WPG Documentation

Release 0.14

Alexey Buzmakov (WPG), Liubov Samoilova (WPG), Oleg Chubar

Dec 08, 2019

Contents

We present a new interactive framework package for coherent and partially coherent X-ray wavefront propagation simulations – "WaveProperGator" (WPG). The package has been developed at European XFEL to facilitate for the end users (beamline scientists and XFEL users) the designing, optimizing and improving X-ray optics to meet their experimental requirements. Our package uses SRW C/C++ library and its Python binding for wavefront propagation simulations. The tool allows for changing source and optics parameters and visualizing the results interactively. The framework is cross-platform: it runs reliably on Linux, MS Windows 7, and Mac OS X. Several application examples, specific for XFEL, will be presented.

CHAPTER 1

Contents:

1.1 About wpg

The simulations based on wave optics have become indispensable for beamline design for highly coherent novel X-ray sources such as X-ray Free Electron Lasers (XFEL).

We present a new interactive framework package for coherent and partially coherent X-ray wavefront propagation simulations – "WaveProperGator" (WPG).

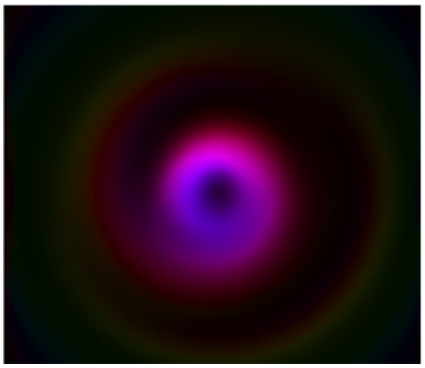
The package has been developed at European XFEL to facilitate for the end users (beamline scientists and XFEL users) the designing, optimizing and improving X-ray optics to meet their experimental requirements. Our package uses SRW C/C++ library and its Python binding for wavefront propagation simulations. The tool allows for changing source and optics parameters and visualizing the results interactively.

The framework is cross-platform: it runs reliably on Linux, MS Windows 7, and Mac OS X. Using IPython as a web-frontend make the users possible run the code at a remote server as well as at their local personal computer. One can use popular Python libraries (such as scipy, numpy, matplotlib) for pre- and post-processing as well as for visualization of the simulation results.

The wavefronts are saved in hdf5 format for the eventual further processing and start-to-end simulations of experiments. The HDF5 format allows for keeping the calculation history within a single file, thus facilitating communication between various scientific groups, as well as cross-checking with other simulation results. The WPG source code together with guidelines for installation and application examples are available on the web.

Several application examples, specific for XFEL, will be presented.

1.2 Getting started



1.2.1 Installation

On Ubuntu Desktop

Install dependencies

```
sudo add-apt-repository -y ppa:jtaylor/ipython
sudo apt-get update
sudo apt-get install -y build-essential python-dev unzip python-numpy python-
↪matplotlib
sudo apt-get install -y python-pip python-scipy python-h5py ipython-notebook
```

Select the directory, in which WPG will be located

```
cd <your_working_directory>
```

Download and build library

```
wget http://github.com/samoylv/WPG/archive/master.zip
```

Extract package:

```
unzip master.zip
```

Change the directory:

```
cd WPG-master
```

Build library. This will download and build FFTW2 and SRW:

```
make all
```

Run web interface.

```
cd samples
ipython notebook
```

If web page not pop up automatically, open your browser in http://localhost:8888

Mac OS X

Install dependencies. You should have XCode and MacPorts installed.

```
sudo port install py27-numpy py27-h5py py27-matplotlib
```

For ipython notebook:

```
sudo port install py27-tornado py27-zmq py27-nose py27-jinja2
py27-sphinx py27-pygments py27-readline py27-ipython py27-scipy
```

For wget automatic downloading:

```
sudo port install wget
```

Select the directory, in which WPG will be located

```
sh cd
```

Download and build library

```
wget --no-check-certificate http://github.com/samoylv/WPG/archive/master.zip
```

Extract package

```
unzip master.zip
```

Change the directory

```
cd WPG-master
```

Build library. This will download and build FFTW2 and SRW

```
make all
```

Run web interface.

```
cd samples
ipython notebook
```

- If web page not pop up automatically, open your browser in http://localhost:8888

If ypu have some errors runing IPython notebook, which ends with

```
ValueError: unknown locale: UTF-8
```

, please add to file *~/.profile* the following commands and try restart your session or reboot:

```
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
```

On xfel server

You can directly run web interface using YOUR_UNIQUE_PORT_NUMBER, and specifiy the path to the WPG installation on the XFEL server.

YOUR_UNIQUE_PORT_NUMBER should be greater than 1024.

Please visit Simulation web-site to register your port number in the table and get further information.

```
ipython notebook --no-browser --port=YOUR_UNIQUE_PORT_NUMBER --notebook-dir=
↪<your_working_directory> &
```

Setup ssh tunnel to the server. **Please use another LOCAL terminal window!**

```
#If you have Linux or Mac OS X local machine, use the following command

ssh -l <your_account> -f -N <server_name>  -LYOUR_UNIQUE_PORT_
↪NUMBER:localhost:YOUR_UNIQUE_PORT_NUMBER
```

On Windows you can use putty and setup ssh tunnel in the following way: putty.pdf

Open your local browser with the following web address: http://localhost:YOUR_UNIQUE_PORT_NUMBER

On MS Windows

You should have installed python2.7 with modules numpy, matplotlib, h5py and ipython. If these modules have not been installed yet, you can download a free python bundle with preinstalled packages here

Download WPG package and unpack it.

Download SRW library and unpack it in any folder.

Copy the following files from the SRW folder to WPG folder:

- SRW-master/env/work/srw_python/*.py to WGP/wpg
- SRW-master/env/work/srw_python/lib/srwlpy2_x64.pyd to WGP/wpg

Rename srwlpy2_x64.pyd to srwlpy.pyd

Run `ipython notebook --pylab=inline` in `WGP/samples`

If you have SRW already installed

Just copy all pyhon files `SRW-master/env/work/srw_python/*.py` and `srwlpy.so` in 'wpg' folder

1.2.2 Useful links

For visualization and browsing HDF5 files.

Please download HDFVeiw tool from [[http://www.hdfgroup.org/hdf-java-html/hdfview/]]

1.3 wpg module

This module contains utils and classes for X-ray wavefront propagation.

The most propagation methods based on SRW library.

1.4 wpg.wavefront module

This module contains base wrapper for SRWLWfr (Wavefront). It's implement numpy inter operations to SRWLWfr structure, serialization to HDF5, visualization tools, etc.

class wpg.wavefront.**Wavefront**(*srwl_wavefront=None*)
Bases: object

This is base class for manipulation with wavefronts in wpg module.

One of most important field is _srwl_wf (instance of srwlib.SRWLWfr). SEtting and getting this field allows to call all SRWLpy functions.

Create wavefront instance.

The most important wavefront fields dynamically initialize from wpg.glossry

Parameters **srwl_wavefront** (srwlib.SRWLWfr) – if present, wavefront inits with it's parameters

Returns Wavefront instance.

get_imag_part(*slice_number=None*, *polarization=None*)
Return imaginary part of wavefront.

Parameters

- **polarization** (*string*) – 'total' or 'horizontal' or 'vertical'
- **slice_number** (*int or range*) – slice number ti return, if None - get 3D array (all slices)

Returns array of imaginary parts

get_intensity(*slice_number=None*, *polarization=None*)
Return intensity of wavefront

Parameters

- **polarization** (*string*) – 'total' or 'horizontal' or 'vertical'
- **slice_number** (*int or range*) – slice number ti return, if None - get 3D array (all slices)

Returns array of intensities

get_limits(*axis='z'*)
Get wavefront mesh limits [xmin, xmax, ...].

Used in 2D visualization tools (as pylab.imshow(wfr_data, extends=wrf.get_limits()))

Params axis 'x','y' or 'z'

Returns list of integers

get_phase(*slice_number=None*, *polarization=None*)
Return phase of wavefront.

Parameters

- **polarization** (*string*) – 'total' or 'horizontal' or 'vertical'
- **slice_number** (*int or range*) – slice number ti return, if None - get 3D array (all slices)

Returns array of phases

get_real_part (*slice_number=None*, *polarization=None*)
Return real part of wavefront.

Parameters

- **polarization** (*string*) – 'total' or 'horizontal' or 'vertical'
- **slice_number** (*int or range*) – slice number ti return, if None - get 3D array (all slices)

Returns array of real parts

load_hdf5 (*file_name*)
Load wavefront from HDF5 file.

Parameters **file_name** (*string*) – output HDF5 file name

srw_info ()
Print self._srwl_wf string representation. Used for debugging.

Returns string

store_hdf5 (*file_name*)
Store wavefront to HDF5 file (attributes and values).

Parameters **file_name** (*string*) – output HDF5 file name

1.5 Glossary definition

Current glossary contains the list of dynamically loading fields of wavefront. It is used for mapping srwlib.SRWLWfr to wpg.Wavefront. Currently this class support the following fields (output of wpg.glossary.print_glossary()):

params/wSpace - Real space or q-space WF presentation [string] - *string*

params/nval - complex electric field nval==2 - **

params/Mesh/ny - Numbers of points, vertical - **

params/Mesh/nx - Numbers of points, horizontal - **

params/Mesh/nSlices - Numbers of points vs photon energy/time for the pulse - **

data/arrEhor - EM field (Re, Im) pairs written in 3D array, slice number changes first. Horizontal polarization - **

data/arrEver - EM field (Re, Im) pairs written in 3D array, slice number changes first. Vertical polarization - **

params/dRx - Error of wavefront horizontal radius [m] - *m*

params/dRy - Error of wavefront horizontal radius [m] - *m*

params/Mesh/qxMax - Maximum of horizontal frequency [1/m] - *1/m*

params/Mesh/qxMin - Minimum of horizontal frequency [1/m] - *1/m*

params/Mesh/qyMax - Maximum of vertical frequency [1/m] - *1/m*

params/Mesh/qyMin - Minimum of vertical frequency [1/m] - *1/m*

params/Mesh/sliceMax - Max value of time [s] or energy [ev] for pulse (fragment) - *s or ev*

params/Mesh/sliceMin - Min value of time [s] or energy [ev] for pulse (fragment) - *s or ev*

params/Mesh/xMax - Maximum of horizontal range [m] - *m*

params/Mesh/xMin - Minimum of horizontal range [m] - *m*

params/Mesh/yMax - Maximum of vertical range [m] - *m*

params/Mesh/yMin - Minimum of vertical range [m] - *m*

params/Mesh/zCoord - Longitudinal position [m], Fast data: length of active undulator, Gaussian source: distance to waist - *m*

params/photonEnergy - Average photon energy [ev] - *ev*

params/Rx - Instantaneous horizontal wavefront radius [m] - *m*

params/Ry - Instantaneous vertical wavefront radius [m] - *m*

params/wDomain - WF in time or frequency (photon energy) domain [string] - *string*

params/wEFieldUnit - Electric field units [string] - *string*

params/wFloatType - Electric field numerical type [string] - *string*

params/xCentre - Horizontal transverse coordinates of wavefront instant 'source center' [m] - *m*

params/yCentre - Vertical transverse coordinates of wavefront instant 'source center' [m] - *m*

version - Hdf5 format version (glossary) - **

1.5.1 `wpg.glossary` module

This module contains definitions (glossary) of Wavefront fields. Described mapping fields SRWLWfr <-> wpg.Wavefront

class wpg.glossary.**RadiationField**(*wf*)

Bases: object

This is base class for all Wavefront fileds.

Used for map values to Wavefront. Also map description string from docstrings and attributes.

Parameters **wf** (*wpg.Wavefront*) – Wavefront

find_units_label()

Search [units] in field docstring

Returns units string

glossary_name = None

keys_chain

Split field name to the parts.

Returns tpule of strings

value

Property where value stored.

class wpg.glossary.**WFDataArrEhor**(*wf*)

Bases: *wpg.glossary.RadiationField*

EM field (Re, Im) pairs written in 3D array, slice number changes first. Horizontal polarization

data/arrEhor field

glossary_name = 'data/arrEhor'

value

EM field (Re, Im) pairs written in 3D array, slice number changes first. Horizontal polarization

class wpg.glossary.**WFDataArrEver**(*wf*)
Bases: *wpg.glossary.RadiationField*

EM field (Re, Im) pairs written in 3D array, slice number changes first. Vertical polarization

data/arrEver field

glossary_name = 'data/arrEver'

value
EM field (Re, Im) pairs written in 3D array, slice number changes first. Vertical polarization

class wpg.glossary.**WFRadiationDRx**(*wf*)
Bases: *wpg.glossary.RadiationField*

Error of wavefront horizontal radius [m]

params/dRx field

glossary_name = 'params/dRx'

value
Error of wavefront horizontal radius [m]

class wpg.glossary.**WFRadiationDRy**(*wf*)
Bases: *wpg.glossary.RadiationField*

Error of wavefront horizontal radius [m]

params/dRy field

glossary_name = 'params/dRy'

value
Error of wavefront horizontal radius [m]

class wpg.glossary.**WFRadiationMeshHvx**(*wf*)
Bases: *wpg.glossary.RadiationField*

Lab-frame horizontal base vector of the observation plane (/ surface in its center)

params/Mesh/hvx field

glossary_name = 'params/Mesh/hvx'

value
Numbers of points, vertical

class wpg.glossary.**WFRadiationMeshHvy**(*wf*)
Bases: *wpg.glossary.RadiationField*

Lab-frame horizontal base vector of the observation plane (/ surface in its center)

params/Mesh/hvy field

glossary_name = 'params/Mesh/hvy'

value
Numbers of points, vertical

class wpg.glossary.**WFRadiationMeshHvz**(*wf*)
Bases: *wpg.glossary.RadiationField*

Lab-frame horizontal base vector of the observation plane (/ surface in its center)

params/Mesh/hvz field

glossary_name = 'params/Mesh/hvz'

value
Numbers of points, vertical

class wpg.glossary.**WFRadiationMeshNSlices**(*wf*)
Bases: *wpg.glossary.RadiationField*

Numbers of points vs photon energy/time for the pulse

params/Mesh/nSlices field

glossary_name = 'params/Mesh/nSlices'

value
Numbers of points vs photon energy/time for the pulse

class wpg.glossary.**WFRadiationMeshNvx**(*wf*)
Bases: *wpg.glossary.RadiationField*

Lab-frame coordinate of the inner normal to observation plane (/ surface in its center)

params/Mesh/nvx field

glossary_name = 'params/Mesh/nvx'

value
Numbers of points, vertical

class wpg.glossary.**WFRadiationMeshNvy**(*wf*)
Bases: *wpg.glossary.RadiationField*

Lab-frame coordinate of the inner normal to observation plane (/ surface in its center)

params/Mesh/nvy field

glossary_name = 'params/Mesh/nvy'

value
Numbers of points, vertical

class wpg.glossary.**WFRadiationMeshNvz**(*wf*)
Bases: *wpg.glossary.RadiationField*

Lab-frame coordinate of the inner normal to observation plane (/ surface in its center)

params/Mesh/nvz field

glossary_name = 'params/Mesh/nvz'

value
Numbers of points, vertical

class wpg.glossary.**WFRadiationMeshNx**(*wf*)
Bases: *wpg.glossary.RadiationField*

Numbers of points, horizontal

params/Mesh/nx field

glossary_name = 'params/Mesh/nx'

value
Numbers of points, horizontal

class wpg.glossary.**WFRadiationMeshNy**(*wf*)
Bases: *wpg.glossary.RadiationField*

Numbers of points, vertical

params/Mesh/ny field

glossary_name = 'params/Mesh/ny'

value
Numbers of points, vertical

class wpg.glossary.**WFRadiationMeshQxMax**(*wf*)
Bases: *wpg.glossary.RadiationField*

Maximum of horizontal frequency [1/m]

params/Mesh/qxMax field

glossary_name = 'params/Mesh/qxMax'

value
Maximum of horizontal frequency [1/m]

class wpg.glossary.**WFRadiationMeshQxMin**(*wf*)
Bases: *wpg.glossary.RadiationField*

Minimum of horizontal frequency [1/m]

params/Mesh/qxMin field

glossary_name = 'params/Mesh/qxMin'

value
Minimum of horizontal frequency [1/m]

class wpg.glossary.**WFRadiationMeshQyMax**(*wf*)
Bases: *wpg.glossary.RadiationField*

Maximum of vertical frequency [1/m]

params/Mesh/qyMax field

glossary_name = 'params/Mesh/qyMax'

value
Maximum of vertical frequency [1/m]

class wpg.glossary.**WFRadiationMeshQyMin**(*wf*)
Bases: *wpg.glossary.RadiationField*

Minimum of vertical frequency [1/m]

params/qyMin field

glossary_name = 'params/Mesh/qyMin'

value
Minimum of vertical frequency [1/m]

class wpg.glossary.**WFRadiationMeshSliceMax**(*wf*)
Bases: *wpg.glossary.RadiationField*

Max value of time [s] or energy [ev] for pulse (fragment)

params/Mesh/sliceMax field

glossary_name = 'params/Mesh/sliceMax'

value
Max value of time [s] or energy [ev] for pulse (fragment)

class wpg.glossary.**WFRadiationMeshSliceMin**(*wf*)
Bases: *wpg.glossary.RadiationField*

Min value of time [s] or energy [ev] for pulse (fragment)

params/Mesh/sliceMin field

glossary_name = 'params/Mesh/sliceMin'

value
Min value of time [s] or energy [ev] for pulse (fragment)

class wpg.glossary.**WFRadiationMeshXMax**(*wf*)
Bases: *wpg.glossary.RadiationField*

Maximum of horizontal range [m]

params/Mesh/xMax field

glossary_name = 'params/Mesh/xMax'

value
Maximum of horizontal range [m]

class wpg.glossary.**WFRadiationMeshXMin**(*wf*)
Bases: *wpg.glossary.RadiationField*

Minimum of horizontal range [m]

params/Mesh/xMin field

glossary_name = 'params/Mesh/xMin'

value
Minimum of horizontal range [m]

class wpg.glossary.**WFRadiationMeshYMax**(*wf*)
Bases: *wpg.glossary.RadiationField*

Maximum of vertical range [m]

params/Mesh/yMax field

glossary_name = 'params/Mesh/yMax'

value
Maximum of vertical range [m]

class wpg.glossary.**WFRadiationMeshYMin**(*wf*)
Bases: *wpg.glossary.RadiationField*

Minimum of vertical range [m]

params/Mesh/yMin field

glossary_name = 'params/Mesh/yMin'

value
Minimum of vertical range [m]

class wpg.glossary.**WFRadiationMeshZCoord**(*wf*)
Bases: *wpg.glossary.RadiationField*

Longitudinal position [m], Fast data: length of active undulator, Gaussian source: distance to waist

params/Mesh/zCoord field

glossary_name = 'params/Mesh/zCoord'

value
Longitudinal position, for fast data - length of active undulator [m]

class wpg.glossary.**WFRadiationNval**(*wf*)
Bases: *wpg.glossary.RadiationField*

complex electric field nval==2

params/nval field

glossary_name = 'params/nval'

value
complex electric field nval==2

class wpg.glossary.**WFRadiationPhotonEnergy**(*wf*)
Bases: *wpg.glossary.RadiationField*

Average photon energy [ev]

params/photonEnergy field.

Parameters **wf** (*wpg.Wavefront*) – Wavefront

glossary_name = 'params/photonEnergy'

value
Average photon energy [ev]

class wpg.glossary.**WFRadiationRx**(*wf*)
Bases: *wpg.glossary.RadiationField*

Instantaneous horizontal wavefront radius [m]

params/Rx field

glossary_name = 'params/Rx'

value
Instantaneous horizontal wavefront radius [m]

class wpg.glossary.**WFRadiationRy**(*wf*)
Bases: *wpg.glossary.RadiationField*

Instantaneous vertical wavefront radius [m]

params/Ry field

glossary_name = 'params/Ry'

value
Instantaneous vertical wavefront radius [m]

class wpg.glossary.**WFRadiationWDomain**(*wf*)
Bases: *wpg.glossary.RadiationField*

WF in time or frequency (photon energy) domain [string]

params/wDomain field

glossary_name = 'params/wDomain'

value
WF in time or frequency (photon energy) domain

class wpg.glossary.**WFRadiationWEFieldUnit**(*wf*)
Bases: *wpg.glossary.RadiationField*

Electric field units [string]

params/wEFieldUnit field

glossary_name = 'params/wEFieldUnit'

value
Electric field units

class wpg.glossary.**WFRadiationWFloatType**(*wf*)
Bases: *wpg.glossary.RadiationField*

Electric field numerical type [string]

params/wFloatType field

glossary_name = 'params/wFloatType'

value
Electric field numerical type

class wpg.glossary.**WFRadiationWSpace**(*wf*)
Bases: *wpg.glossary.RadiationField*

Real space or q-space WF presentation [string]

params/wSpace field

glossary_name = 'params/wSpace'

value
Real space or q-space WF presentation

class wpg.glossary.**WFRadiationXCentre**(*wf*)
Bases: *wpg.glossary.RadiationField*

Horizontal transverse coordinates of wavefront instant 'source center' [m]

params/xCentre field

glossary_name = 'params/xCentre'

value
Horizontal transverse coordinates of wavefront instant 'source center' [m]

class wpg.glossary.**WFRadiationYCentre**(*wf*)
Bases: *wpg.glossary.RadiationField*

Vertical transverse coordinates of wavefront instant 'source center' [m]

params/yCentre field

glossary_name = 'params/yCentre'

value
Vertical transverse coordinates of wavefront instant 'source center' [m]

class wpg.glossary.**WFVersion**(*wf*)
Bases: *wpg.glossary.RadiationField*

Hdf5 format version (glossary)

Version field.

Parameters **wf** (*wpg.Wavefront*) – Wavefront

glossary_name = 'version'

value
Hdf5 format version (glossary)

wpg.glossary.**get_glosary_info**()

Returns dictionary field_name -> doc

wpg.glossary.**get_wf_fields**()
Return fields in proper order to map it in Wavefront

Returns iterator over wavefront fields in glossary

wpg.glossary.**print_glossary**()
Print glossary docs

wpg.glossary.**print_glossary_html**()
Print glossry docsas html table. Used to build alfresco documentaion page.

1.6 wpg.srwlib module

class wpg.srwlib.**SRWLGsnBm**(*_x=0*, *_y=0*, *_z=0*, *_xp=0*, *_yp=0*, *_avgPhotEn=1*, *_pulseEn=1*, *_repRate=1*, *_polar=1*, *_sigX=1e-05*, *_sigY=1e-05*, *_sigT=1e-15*, *_mx=0*, *_my=0*)

Bases: object

Gaussian Beam

Parameters

- **_x** – average horizontal coordinates of waist [m]
- **_y** – average vertical coordinates of waist [m]
- **_z** – average longitudinal coordinate of waist [m]
- **_xp** – average horizontal angle at waist [rad]
- **_yp** – average verical angle at waist [rad]
- **_avgPhotEn** – average photon energy [eV]
- **_pulseEn** – energy per pulse [J]
- **_repRate** – rep. rate [Hz]
- **_polar** – polarization 1- lin. hor., 2- lin. vert., 3- lin. 45 deg., 4- lin.135 deg., 5- circ. right, 6- circ. left
- **_sigX** – rms beam size vs horizontal position [m] at waist (for intensity)
- **_sigY** – rms beam size vs vertical position [m] at waist (for intensity)
- **_sigT** – rms pulse duration [s] (for intensity)
- **_mx** – transverse Gauss-Hermite mode order in horizontal direction
- **_my** – transverse Gauss-Hermite mode order in vertical direction

class wpg.srwlib.**SRWLKickM**(*_arKickMx=None*, *_arKickMy=None*, *_order=2*, *_nx=0*, *_ny=0*, *_nz=0*, *_rx=0*, *_ry=0*, *_rz=0*, *_x=0*, *_y=0*, *_z=0*)

Bases: object

Kick Matrix (for fast trajectory calculation)

Parameters

- **_arKickMx** – horizontal kick-matrix (tabulated on the same transverse grid vs x and y as vertical kick-matrix)
- **_arKickMy** – vertical kick-matrix (tabulated on the same transverse grid vs x and y as horizontal kick-matrix)
- **_order** – kick order: 1- first order (in this case kick matrix data is assumed to be in [T*m]), 2- second order (kick matrix data is assumed to be in [T^2*m^2])
- **_nx** – numbers of points in kick matrices in horizontal direction
- **_ny** – numbers of points in kick matrices in vertical direction
- **_nz** – number of steps in longitudinal direction
- **_rx** – range covered by kick matrices in horizontal direction [m]
- **_ry** – range covered by kick matrices in vertical direction [m]
- **_rz** – extension in longitudinal direction [m]
- **_x** – horizontal coordinate of center point [m]
- **_y** – vertical coordinate of center point [m]
- **_z** – longitudinal coordinate of center point [m]

class wpg.srwlib.**SRWLMagFld**

Bases: object

Magnetic Field (base class)

class wpg.srwlib.**SRWLMagFld3D**(*_arBx=None*, *_arBy=None*, *_arBz=None*, *_nx=0*, *_ny=0*, *_nz=0*, *_rx=0*, *_ry=0*, *_rz=0*, *_nRep=1*, *_interp=1*, *_arX=None*, *_arY=None*, *_arZ=None*)

Bases: *wpg.srwlib.SRWLMagFld*

Magnetic Field: Arbitrary 3D

Parameters

- **_arBx** – horizontal magnetic field component array [T]
- **_arBy** – vertical magnetic field component array [T]
- **_arBz** – longitudinal magnetic field component array [T]
- **_nx** – number of magnetic field data points in the horizontal direction
- **_ny** – number of magnetic field data points in the vertical direction
- **_nz** – number of magnetic field data points in the longitudinal direction
- **_rx** – range of horizontal coordinate for which the field is defined [m]
- **_ry** – range of vertical coordinate for which the field is defined [m]
- **_rz** – range of longitudinal coordinate for which the field is defined [m]
- **_nRep** – "number of periods", i.e. number of times the field is "repeated" in the longitudinal direction
- **_interp** – interpolation method to use (e.g. for trajectory calculation), 1- bi-linear (3D), 2- (bi-)quadratic (3D), 3- (bi-)cubic (3D)
- **_arX** – optional array of horizontal transverse coordinate of an irregular 3D mesh (if this array is defined, rx will be ignored)

- **_arY** – optional array of vertical transverse coordinate of an irregular 3D mesh (if this array is defined, ry will be ignored)
- **_arZ** – optional array of longitudinal coordinate of an irregular 3D mesh (if this array is defined, rz will be ignored)

add_const(*_bx=0*, *_by=0*, *_bz=0*)
Adds constant magnetic field to the entire tabulated field (to simulate e.g. background magnetic field effects) :param _bx: horizontal magnetic field component to add [T] :param _by: vertical magnetic field component to add [T] :param _bz: longitudinal magnetic field component to add [T]

save_ascii(*_file_path*, *_xc=0*, *_yc=0*, *_zc=0*)
Auxiliary function to write tabulated Arbitrary 3D Magnetic Field data to ASCII file

class wpg.srwlib.**SRWLMagFldC**(*_arMagFld=None*, *_arXc=None*, *_arYc=None*, *_arZc=None*, *_arVx=None*, *_arVy=None*, *_arVz=None*, *_arAng=None*)
Bases: *wpg.srwlib.SRWLMagFld*

Magnetic Field: Container

Parameters

- **_arMagFld** – magnetic field structures array
- **_arXc** – horizontal center positions of magnetic field elements in arMagFld array [m]
- **_arYc** – vertical center positions of magnetic field elements in arMagFld array [m]
- **_arZc** – longitudinal center positions of magnetic field elements in arMagFld array [m]
- **_arVx** – horizontal components of axis vectors of magnetic field elements in arMagFld array [rad]
- **_arVy** – vertical components of axis vectors of magnetic field elements in arMagFld array [rad]
- **_arVz** – longitudinal components of axis vectors of magnetic field elements in arMagFld array [rad]
- **_arAng** – rotation angles of magnetic field elements about their axes [rad]

add(*_mag*, *_xc=None*, *_yc=None*, *_zc=None*, *_vx=None*, *_vy=None*, *_vz=None*, *_ang=None*)
Adds magnetic element to container :param _mag: magnetic element (or array of elements) to be added :param _xc: horizontal center position (or array of center positions) of magnetic field element to be added [m] :param _yc: vertical center positions (or array of center positions) of magnetic field element to be added [m] :param _zc: longitudinal center positions (or array of center positions) of magnetic field element to be added [m] :param _vx: horizontal component of axis vectors of magnetic field element to be added [rad] :param _vy: vertical component of axis vectors of magnetic field element to be added [rad] :param _vz: longitudinal components of axis vector of magnetic field element to be added [rad] :param _ang: rotation angle about axis [rad]

allocate(*_nElem*)

class wpg.srwlib.**SRWLMagFldH**(*_n=1*, *_h_or_v='v'*, *_B=0*, *_ph=0*, *_s=1*, *_a=1*)
Bases: *wpg.srwlib.SRWLMagFld*

Magnetic Field: Undulator Harmonic

Parameters

- **_n** – harmonic number
- **_h_or_v** – magnetic field plane horzontal ('h') or vertical ('v')
- **_B** – magnetic field amplitude [T]

- **_ph** – initial phase [rad]
- **_s** – symmetry vs longitudinal position 1 - symmetric (B ~ cos(2*Pi*n*z/per + ph)) , -1 - anti-symmetric (B ~ sin(2*Pi*n*z/per + ph))
- **_a** – coefficient for transverse depenednce B*cosh(2*Pi*n*a*y/per)*cos(2*Pi*n*z/per + ph)

class wpg.srwlib.**SRWLMagFldM**(*_G=0*, *_m=2*, *_n_or_s='n'*, *_Leff=0*, *_Ledge=0*, *_R=0*)
Bases: *wpg.srwlib.SRWLMagFld*

Magnetic Field: Multipole Magnet

Parameters

- **_G** – field parameter [T] for dipole, [T/m] for quadrupole (negative means defocusing for x), [T/m^2] for sextupole, [T/m^3] for octupole
- **_m** – multipole order 1 for dipole, 2 for quadrupoole, 3 for sextupole, 4 for octupole
- **_n_or_s** – normal ('n') or skew ('s')
- **_Leff** – effective length [m]
- **_Ledge** – "soft" edge length for field variation from 10% to 90% [m]; G/(1 + ((z-zc)/d)^2)^2 fringe field dependence is assumed
- **_R** – radius of curvature of central trajectory [m] (for simulating e.g. quadrupole component integrated to a bending magnet; effective if > 0)

class wpg.srwlib.**SRWLMagFldS**(*_B=0*, *_Leff=0*)
Bases: *wpg.srwlib.SRWLMagFld*

Magnetic Field: Solenoid

Parameters

- **_B** – magnetic field [T]
- **_Leff** – effective length [m]

class wpg.srwlib.**SRWLMagFldU**(*_arHarm=None*, *_per=0*, *_nPer=0*)
Bases: *wpg.srwlib.SRWLMagFld*

Magnetic Field: Undulator

Parameters

- **_arHarm** – array of field harmonics
- **_per** – period length [m]
- **_nPer** – number of periods (will be rounded to integer)

E1_2_B(*_e1*, *_en_elec=3.0*)
Estimate deflection parameter from :param _e1: fundamental photon energy [eV] :param _en_elec: electron energy [GeV] :return: magnetic field amplitude [T]

E1_2_K(*_e1*, *_en_elec=3.0*)
Estimate deflection parameter from :param _e1: fundamental photon energy [eV] :param _en_elec: electron energy [GeV] :return: deflection parameter

allocate(*_nHarm*)

get_E1(*_en_elec=3.0*, *_unit='eV'*)
Estimate fundamental photon energy :param _en_elec: electron energy [GeV] :return: fundamental photon energy [eV]

get_K()
Estimate K (deflection parameter) value

set_sin(*_per=0.02*, *_len=1*, *_bx=0*, *_by=0*, *_phx=0*, *_phy=0*, *_sx=1*, *_sy=1*)
Setup basic undulator with sinusoidal magnetic field :param _per: period length [m] :param _len: undulator length [m] :param _bx: horizontal magnetic field amplitude [m] :param _by: vertical magnetic field amplitude [m] :param _phx: initial phase of the horizontal magnetic field [rad] :param _phx: initial phase of the vertical magnetic field [rad] :param _sx: symmetry of the horizontal magnetic field vs longitudinal position 1 - symmetric (B ~ cos(2*Pi*n*z/per + ph)) , -1 - anti-symmetric (B ~ sin(2*Pi*n*z/per + ph)) :param _sy: symmetry of the vertical magnetic field vs longitudinal position

class wpg.srwlib.**SRWLOpt**
Bases: object

Optical Element (base class)

class wpg.srwlib.**SRWLOptA**(*_shape='r'*, *_ap_or_ob='a'*, *_Dx=0*, *_Dy=0*, *_x=0*, *_y=0*)
Bases: *wpg.srwlib.SRWLOpt*

Optical Element: Aperture / Obstacle

Parameters

- **_shape** – 'r' for rectangular, 'c' for circular
- **_ap_or_ob** – 'a' for aperture, 'o' for obstacle
- **_Dx** – horizontal transverse dimension [m]; in case of circular aperture, only Dx is used for diameter
- **_Dy** – vertical transverse dimension [m]; in case of circular aperture, Dy is ignored
- **_x** – horizontal transverse coordinate of center [m]
- **_y** – vertical transverse coordinate of center [m]

class wpg.srwlib.**SRWLOptAng**(*_ang_x=0*, *_ang_y=0*)
Bases: *wpg.srwlib.SRWLOpt*

Optical Element: Angle

Parameters

- **_ang_x** – horizontal angle [rad]
- **_ang_y** – vertical angle [rad]

class wpg.srwlib.**SRWLOptC**(*_arOpt=None*, *_arProp=None*)
Bases: *wpg.srwlib.SRWLOpt*

Optical Element: Container

Parameters

- **_arOpt** – optical element structures list (or array)
- **_arProp** – list of lists of propagation parameters to be used for each individual optical element Each element _arProp[i] is a list in which elements mean: [0]: Auto-Resize (1) or not (0) Before propagation [1]: Auto-Resize (1) or not (0) After propagation [2]: Relative Precision for propagation with Auto-Resizing (1. is nominal) [3]: Allow (1) or not (0) for semi-analytical treatment of the quadratic (leading) phase terms at the propagation [4]: Do any Resizing on Fourier side, using FFT, (1) or not (0) [5]: Horizontal Range modification factor at Resizing (1. means no modification) [6]: Horizontal Resolution modification factor at Resizing (1. means no modification) [7]: Vertical Range modification factor at

Resizing (1. means no modification) [8]: Vertical Resolution modification factor at Resizing (1. means no modification) [9]: Optional: Type of wavefront Shift before Resizing (vs which coordinates; to be implemented) [10]: Optional: New Horizontal wavefront Center position after Shift (to be implemented) [11]: Optional: New Vertical wavefront Center position after Shift (to be implemented) [12]: Optional: Orientation of the Output Optical Axis vector in the Incident Beam Frame: Horizontal Coordinate [13]: Optional: Orientation of the Output Optical Axis vector in the Incident Beam Frame: Vertical Coordinate [14]: Optional: Orientation of the Output Optical Axis vector in the Incident Beam Frame: Longitudinal Coordinate [15]: Optional: Orientation of the Horizontal Base vector of the Output Frame in the Incident Beam Frame: Horizontal Coordinate [16]: Optional: Orientation of the Horizontal Base vector of the Output Frame in the Incident Beam Frame: Vertical Coordinate

allocate(*_nElem*)

class wpg.srwlib.**SRWLOptCryst**(*_d_sp*, *_psi0r*, *_psi0i*, *_psi_hr*, *_psi_hi*, *_psi_hbr*, *_psi_hbi*, *_tc*, *_ang_as*, *_nvx=0*, *_nvy=0*, *_nvz=-1*, *_tvx=1*, *_tvy=0*)

Bases: *wpg.srwlib.SRWLOpt*

Optical Element: Ideal Crystal

Parameters

- **_d_sp** – (_d_space) crystal reflecting planes d-spacing (John's dA) [A]
- **_psi0r** – real part of 0-th Fourier component of crystal polarizability (John's psi0c.real) (units?)
- **_psi0i** – imaginary part of 0-th Fourier component of crystal polarizability (John's psi0c.imag) (units?)
- **_psi_hr** – (_psiHr) real part of H-th Fourier component of crystal polarizability (John's psihc.real) (units?)
- **_psi_hi** – (_psiHi) imaginary part of H-th Fourier component of crystal polarizability (John's psihc.imag) (units?)
- **_psi_hbr** – (_psiHBr:) real part of -H-th Fourier component of crystal polarizability (John's psimhc.real) (units?)
- **_psi_hbi** – (_psiHBi:) imaginary part of -H-th Fourier component of crystal polarizability (John's psimhc.imag) (units?)
- **_tc** – crystal thickness [m] (John's thicum)
- **_ang_as** – (_Tasym) asymmetry angle [rad] (John's alphdg)
- **_nvx** – horizontal coordinate of outward normal to crystal surface (John's angles: thdg, chidg, phidg)
- **_nvy** – vertical coordinate of outward normal to crystal surface (John's angles: thdg, chidg, phidg)
- **_nvz** – longitudinal coordinate of outward normal to crystal surface (John's angles: thdg, chidg, phidg)
- **_tvx** – horizontal coordinate of central tangential vector (John's angles: thdg, chidg, phidg)
- **_tvy** – vertical coordinate of central tangential vector (John's angles: thdg, chidg, phidg)

find_orient(*_en*, *_ang_dif_pl=0*)

Finds optimal crystal orientation in the input beam frame (i.e. surface normal and tangential vectors) and the orientation of the output beam frame (i.e. coordinates of the longitudinal and horizontal vectors in the input beam frame) :param _en: photon energy [eV] :param _ang_dif_pl: diffraction plane angle (0

corresponds to the vertical deflection; pi/2 to the horizontal deflection; any value in between is allowed) :return: list of two triplets of vectors:

out[0] is the list of 3 base vectors [tangential, saggitan, normal] defining the crystal orientation out[1] is the list of 3 base vectors [ex, ey, ez] defining orientation of the output beam frame the cartesian coordinates of all these vectors are given in the frame of the input beam

set_orient(*_nvx=0*, *_nvy=0*, *_nvz=-1*, *_tvx=1*, *_tvy=0*)

Defines Crystal Orientation in the frame of the Incident Photon beam :param _nvx: horizontal coordinate of normal vector :param _nvy: vertical coordinate of normal vector :param _nvz: longitudinal coordinate of normal vector :param _tvx: horizontal coordinate of tangential vector :param _tvy: vertical coordinate of tangential vector

class wpg.srwlib.**SRWLOptD**(*_L=0*, *_treat=0*)

Bases: *wpg.srwlib.SRWLOpt*

Optical Element: Drift Space

Parameters

- **_L** – Length [m]
- **_treat** – switch specifying whether the absolute optical path should be taken into account in radiation phase (=1) or not (=0, default)

class wpg.srwlib.**SRWLOptG**(*_mirSub*, *_m=1*, *_grDen=100*, *_grDen1=0*, *_grDen2=0*, *_grDen3=0*, *_grDen4=0*, *_grAng=0*)

Bases: *wpg.srwlib.SRWLOpt*

Optical Element: Grating

Parameters

- **_mirSub** – SRWLOptMir (or derived) type object defining substrate of the grating
- **_m** – output (diffraction) order
- **_grDen** – groove density [lines/mm] (coefficient a0 in the polynomial groove density: a0 + a1*y + a2*y^2 + a3*y^3 + a4*y^4)
- **_grDen1** – groove density polynomial coefficient a1 [lines/mm^2]
- **_grDen2** – groove density polynomial coefficient a2 [lines/mm^3]
- **_grDen3** – groove density polynomial coefficient a3 [lines/mm^4]
- **_grDen4** – groove density polynomial coefficient a4 [lines/mm^5]
- **_grAng** – angle between the grove direction and the saggital direction of the substrate [rad] (by default, groves are made along saggital direction (_grAng=0))

class wpg.srwlib.**SRWLOptL**(*_Fx=1e+23*, *_Fy=1e+23*, *_x=0*, *_y=0*)

Bases: *wpg.srwlib.SRWLOpt*

Optical Element: Thin Lens

Parameters

- **_Fx** – focal length in horizontal plane [m]
- **_Fy** – focal length in vertical plane [m]
- **_x** – horizontal coordinate of center [m]
- **_y** – vertical coordinate of center [m]

class wpg.srwlib.**SRWLOptMir**

Bases: *wpg.srwlib.SRWLOpt*

Optical Element: Mirror (focusing)

set_all(*_size_tang=1*, *_size_sag=1*, *_ap_shape='r'*, *_sim_meth=2*, *_npt=100*, *_nps=100*, *_treat_in_out=1*, *_ext_in=0*, *_ext_out=0*, *_nvx=0*, *_nvy=0*, *_nvz=-1*, *_tvx=1*, *_tvy=0*, *_x=0*, *_y=0*, *_refl=1*, *_n_ph_en=1*, *_n_ang=1*, *_n_comp=1*, *_ph_en_start=1000.0*, *_ph_en_fin=1000.0*, *_ph_en_scale_type='lin'*, *_ang_start=0*, *_ang_fin=0*, *_ang_scale_type='lin'*)

Parameters

- **_size_tang** – size in tangential direction [m]
- **_size_sag** – size in sagital direction [m]
- **_ap_shape** – shape of aperture in local frame ('r' for rectangular, 'e' for elliptical)
- **_sim_meth** – simulation method (1 for "thin" approximation, 2 for "thick" approximation)
- **_treat_in_out** – switch specifying how to treat input and output wavefront before and after the main propagation through the optical element: 0- assume that the input wavefront is defined in the plane before the optical element, and the output wavefront is required in a plane just after the element; 1- assume that the input wavefront is defined in the plane at the optical element center and the output wavefront is also required at the element center; 2- assume that the input wavefront is defined in the plane at the optical element center and the output wavefront is also required at the element center; however, before the propagation though the optical element, the wavefront should be propagated through a drift back to a plane just before the optical element, then a special propagator will bring the wavefront to a plane at the optical element exit, and after that the wavefront will be propagated through a drift back to the element center;
- **_ext_in** – optical element extent on the input side, i.e. distance between the input plane and the optical center (positive, in [m]) to be used at wavefront propagation manipulations; if 0, this extent will be calculated internally from optical element parameters
- **_ext_out** – optical element extent on the output side, i.e. distance between the optical center and the output plane (positive, in [m]) to be used at wavefront propagation manipulations; if 0, this extent will be calculated internally from optical element parameters
- **_nvx** – horizontal coordinate of central normal vector
- **_nvy** – vertical coordinate of central normal vector
- **_nvz** – longitudinal coordinate of central normal vector
- **_tvx** – horizontal coordinate of central tangential vector
- **_tvy** – vertical coordinate of central tangential vector
- **_x** – horizontal position of mirror center [m]
- **_y** – vertical position of mirror center [m]
- **_refl** – reflectivity coefficient to set (can be one number or C-aligned flat complex array vs photon energy vs grazing angle vs component (sigma, pi))
- **_n_ph_en** – number of photon energy values for which the reflectivity coefficient is specified
- **_n_ang** – number of grazing angle values for which the reflectivity coefficient is specified

- **_n_comp** – number of electric field components for which the reflectivity coefficient is specified (can be 1 or 2)
- **_ph_en_start** – initial photon energy value for which the reflectivity coefficient is specified
- **_ph_en_fin** – final photon energy value for which the reflectivity coefficient is specified
- **_ph_en_scale_type** – photon energy sampling type ('lin' for linear, 'log' for logarithmic)
- **_ang_start** – initial grazing angle value for which the reflectivity coefficient is specified
- **_ang_fin** – final grazing angle value for which the reflectivity coefficient is specified
- **_ang_scale_type** – angle sampling type ('lin' for linear, 'log' for logarithmic)

set_dim_sim_meth(*_size_tang=1*, *_size_sag=1*, *_ap_shape='r'*, *_sim_meth=2*, *_npt=500*, *_nps=500*, *_treat_in_out=1*, *_ext_in=0*, *_ext_out=0*)

Sets Mirror Dimensions, Aperture Shape and its simulation method :param _size_tang: size in tangential direction [m] :param _size_sag: size in sagital direction [m] :param _ap_shape: shape of aperture in local frame ('r' for rectangular, 'e' for elliptical) :param _sim_meth: simulation method (1 for "thin" approximation, 2 for "thick" approximation) :param _npt: number of mesh points to represent mirror in tangential direction (used for "thin" approximation) :param _nps: number of mesh points to represent mirror in sagital direction (used for "thin" approximation) :param _treat_in_out: switch specifying how to treat input and output wavefront before and after the main propagation through the optical element:

0- assume that the input wavefront is defined in the plane before the optical element, and the output wavefront is required in a plane just after the element; 1- assume that the input wavefront is defined in the plane at the optical element center and the output wavefront is also required at the element center; 2- assume that the input wavefront is defined in the plane at the optical element center and the output wavefront is also required at the element center; however, before the propagation though the optical element, the wavefront should be propagated through a drift back to a plane just before the optical element, then a special propagator will bring the wavefront to a plane at the optical element exit, and after this the wavefront will be propagated through a drift back to the element center;

Parameters

- **_ext_in** – optical element extent on the input side, i.e. distance between the input plane and the optical center (positive, in [m]) to be used at wavefront propagation manipulations; if 0, this extent will be calculated internally from optical element parameters
- **_ext_out** – optical element extent on the output side, i.e. distance between the optical center and the output plane (positive, in [m]) to be used at wavefront propagation manipulations; if 0, this extent will be calculated internally from optical element parameters

set_orient(*_nvx=0*, *_nvy=0*, *_nvz=-1*, *_tvx=1*, *_tvy=0*, *_x=0*, *_y=0*)

Defines Mirror Orientation in the frame of the incident photon beam :param _nvx: horizontal coordinate of central normal vector :param _nvy: vertical coordinate of central normal vector :param _nvz: longitudinal coordinate of central normal vector :param _tvx: horizontal coordinate of central tangential vector :param _tvy: vertical coordinate of central tangential vector :param _x: horizontal position of mirror center [m] :param _y: vertical position of mirror center [m]

set_reflect(*_refl=1*, *_n_ph_en=1*, *_n_ang=1*, *_n_comp=1*, *_ph_en_start=0*, *_ph_en_fin=0*, *_ph_en_scale_type='lin'*, *_ang_start=0*, *_ang_fin=0*, *_ang_scale_type='lin'*)

Sets Mirror Reflectivity :param _refl: reflectivity coefficient to set (can be one number or C-aligned flat array complex array vs photon energy vs grazing angle vs component (sigma, pi)) :param _n_ph_en: number of photon energy values for which the reflectivity coefficient is specified :param _n_ang: number of

grazing angle values for which the reflectivity coefficient is specified :param _n_comp: number of electric field components for which the reflectivity coefficient is specified (can be 1 or 2) :param _ph_en_start: initial photon energy value for which the reflectivity coefficient is specified :param _ph_en_fin: final photon energy value for which the reflectivity coefficient is specified :param _ph_en_scale_type: photon energy sampling type ('lin' for linear, 'log' for logarithmic) :param _ang_start: initial grazing angle value for which the reflectivity coefficient is specified :param _ang_fin: final grazing angle value for which the reflectivity coefficient is specified :param _ang_scale_type: angle sampling type ('lin' for linear, 'log' for logarithmic)

class wpg.srwlib.**SRWLOptMirEl**(*_p=1*, *_q=1*, *_ang_graz=0.001*, *_r_sag=1e+23*, *_size_tang=1*, *_size_sag=1*, *_ap_shape='r'*, *_sim_meth=2*, *_npt=500*, *_nps=500*, *_treat_in_out=1*, *_ext_in=0*, *_ext_out=0*, *_nvx=0*, *_nvy=0*, *_nvz=-1*, *_tvx=1*, *_tvy=0*, *_x=0*, *_y=0*, *_refl=1*, *_n_ph_en=1*, *_n_ang=1*, *_n_comp=1*, *_ph_en_start=1000.0*, *_ph_en_fin=1000.0*, *_ph_en_scale_type='lin'*, *_ang_start=0*, *_ang_fin=0*, *_ang_scale_type='lin'*)

Bases: *wpg.srwlib.SRWLOptMir*

Optical Element: Mirror: Ellipsoid

Parameters

- **_p** – distance from first focus ("source") to mirror center [m]
- **_q** – distance from mirror center to second focus ("image") [m]
- **_ang_graz** – grazing angle at mirror center at perfect orientation [rad]
- **_r_sag** – sagital radius of curvature at mirror center [m]
- **_size_tang** – size in tangential direction [m]
- **_size_sag** – size in sagital direction [m]
- **_ap_shape** – shape of aperture in local frame ('r' for rectangular, 'e' for elliptical)
- **_sim_meth** – simulation method (1 for "thin" approximation, 2 for "thick" approximation)
- **_npt** – number of mesh points to represent mirror in tangential direction (used for "thin" approximation)
- **_nps** – number of mesh points to represent mirror in sagital direction (used for "thin" approximation)
- **_treat_in_out** – switch specifying how to treat input and output wavefront before and after the main propagation through the optical element: 0- assume that the input wavefront is defined in the plane before the optical element, and the output wavefront is required in a plane just after the element; 1- assume that the input wavefront is defined in the plane at the optical element center and the output wavefront is also required at the element center; 2- assume that the input wavefront is defined in the plane at the optical element center and the output wavefront is also required at the element center; however, before the propagation though the optical element, the wavefront should be propagated through a drift back to a plane just before the optical element, then a special propagator will bring the wavefront to a plane at the optical element exit, and after this the wavefront will be propagated through a drift back to the element center;
- **_ext_in** – optical element extent on the input side, i.e. distance between the input plane and the optical center (positive, in [m]) to be used at wavefront propagation manipulations; if 0, this extent will be calculated internally from optical element parameters

- **_ext_out** – optical element extent on the output side, i.e. distance between the optical center and the output plane (positive, in [m]) to be used at wavefront propagation manipulations; if 0, this extent will be calculated internally from optical element parameters
- **_nvx** – horizontal coordinate of central normal vector
- **_nvy** – vertical coordinate of central normal vector
- **_nvz** – longitudinal coordinate of central normal vector
- **_tvx** – horizontal coordinate of central tangential vector
- **_tvy** – vertical coordinate of central tangential vector
- **_x** – horizontal position of mirror center [m]
- **_y** – vertical position of mirror center [m]
- **_refl** – reflectivity coefficient to set (can be one number or C-aligned flat complex array vs photon energy vs grazing angle vs component (sigma, pi))
- **_n_ph_en** – number of photon energy values for which the reflectivity coefficient is specified
- **_n_ang** – number of grazing angle values for which the reflectivity coefficient is specified
- **_n_comp** – number of electric field components for which the reflectivity coefficient is specified (can be 1 or 2)
- **_ph_en_start** – initial photon energy value for which the reflectivity coefficient is specified
- **_ph_en_fin** – final photon energy value for which the reflectivity coefficient is specified
- **_ph_en_scale_type** – photon energy sampling type ('lin' for linear, 'log' for logarithmic)
- **_ang_start** – initial grazing angle value for which the reflectivity coefficient is specified
- **_ang_fin** – final grazing angle value for which the reflectivity coefficient is specified
- **_ang_scale_type** – angle sampling type ('lin' for linear, 'log' for logarithmic)

class wpg.srwlib.**SRWLOptMirPl**(*_size_tang=1*, *_size_sag=1*, *_ap_shape='r'*, *_sim_meth=2*, *_npt=100*, *_nps=100*, *_treat_in_out=1*, *_ext_in=0*, *_ext_out=0*, *_nvx=0*, *_nvy=0*, *_nvz=-1*, *_tvx=1*, *_tvy=0*, *_x=0*, *_y=0*, *_refl=1*, *_n_ph_en=1*, *_n_ang=1*, *_n_comp=1*, *_ph_en_start=1000.0*, *_ph_en_fin=1000.0*, *_ph_en_scale_type='lin'*, *_ang_start=0*, *_ang_fin=0*, *_ang_scale_type='lin'*)

Bases: *wpg.srwlib.SRWLOptMir*

Optical Element: Mirror: Plane

Parameters

- **_size_tang** – size in tangential direction [m]
- **_size_sag** – size in sagital direction [m]
- **_ap_shape** – shape of aperture in local frame ('r' for rectangular, 'e' for elliptical)
- **_sim_meth** – simulation method (1 for "thin" approximation, 2 for "thick" approximation)
- **_treat_in_out** – switch specifying how to treat input and output wavefront before and after the main propagation through the optical element: 0- assume that the input wavefront is defined in the plane before the optical element, and the output wavefront is required in a plane just after the element; 1- assume that the input wavefront is defined in the plane at

the optical element center and the output wavefront is also required at the element center; 2- assume that the input wavefront is defined in the plane at the optical element center and the output wavefront is also required at the element center; however, before the propagation though the optical element, the wavefront should be propagated through a drift back to a plane just before the optical element, then a special propagator will bring the wavefront to a plane at the optical element exit, and after that the wavefront will be propagated through a drift back to the element center;

- **_ext_in** – optical element extent on the input side, i.e. distance between the input plane and the optical center (positive, in [m]) to be used at wavefront propagation manipulations; if 0, this extent will be calculated internally from optical element parameters
- **_ext_out** – optical element extent on the output side, i.e. distance between the optical center and the output plane (positive, in [m]) to be used at wavefront propagation manipulations; if 0, this extent will be calculated internally from optical element parameters
- **_nvx** – horizontal coordinate of central normal vector
- **_nvy** – vertical coordinate of central normal vector
- **_nvz** – longitudinal coordinate of central normal vector
- **_tvx** – horizontal coordinate of central tangential vector
- **_tvy** – vertical coordinate of central tangential vector
- **_x** – horizontal position of mirror center [m]
- **_y** – vertical position of mirror center [m]
- **_refl** – reflectivity coefficient to set (can be one number or C-aligned flat complex array vs photon energy vs grazing angle vs component (sigma, pi))
- **_n_ph_en** – number of photon energy values for which the reflectivity coefficient is specified
- **_n_ang** – number of grazing angle values for which the reflectivity coefficient is specified
- **_n_comp** – number of electric field components for which the reflectivity coefficient is specified (can be 1 or 2)
- **_ph_en_start** – initial photon energy value for which the reflectivity coefficient is specified
- **_ph_en_fin** – final photon energy value for which the reflectivity coefficient is specified
- **_ph_en_scale_type** – photon energy sampling type ('lin' for linear, 'log' for logarithmic)
- **_ang_start** – initial grazing angle value for which the reflectivity coefficient is specified
- **_ang_fin** – final grazing angle value for which the reflectivity coefficient is specified
- **_ang_scale_type** – angle sampling type ('lin' for linear, 'log' for logarithmic)

class wpg.srwlib.**SRWLOptMirSph**(*_r=1.0*, *_size_tang=1*, *_size_sag=1*, *_ap_shape='r'*, *_sim_meth=2*, *_npt=500*, *_nps=500*, *_treat_in_out=1*, *_ext_in=0*, *_ext_out=0*, *_nvx=0*, *_nvy=0*, *_nvz=-1*, *_tvx=1*, *_tvy=0*, *_x=0*, *_y=0*, *_refl=1*, *_n_ph_en=1*, *_n_ang=1*, *_n_comp=1*, *_ph_en_start=1000.0*, *_ph_en_fin=1000.0*, *_ph_en_scale_type='lin'*, *_ang_start=0*, *_ang_fin=0*, *_ang_scale_type='lin'*)

Bases: *wpg.srwlib.SRWLOptMir*

Optical Element: Mirror: Spherical

Parameters

- **_r** – radius of surface curvature [m]
- **_size_tang** – size in tangential direction [m]
- **_size_sag** – size in sagital direction [m]
- **_ap_shape** – shape of aperture in local frame ('r' for rectangular, 'e' for elliptical)
- **_sim_meth** – simulation method (1 for "thin" approximation, 2 for "thick" approximation)
- **_npt** – number of mesh points to represent mirror in tangential direction (used for "thin" approximation)
- **_nps** – number of mesh points to represent mirror in sagital direction (used for "thin" approximation)
- **_treat_in_out** – switch specifying how to treat input and output wavefront before and after the main propagation through the optical element: 0- assume that the input wavefront is defined in the plane before the optical element, and the output wavefront is required in a plane just after the element; 1- assume that the input wavefront is defined in the plane at the optical element center and the output wavefront is also required at the element center; 2- assume that the input wavefront is defined in the plane at the optical element center and the output wavefront is also required at the element center; however, before the propagation though the optical element, the wavefront should be propagated through a drift back to a plane just before the optical element, then a special propagator will bring the wavefront to a plane at the optical element exit, and after this the wavefront will be propagated through a drift back to the element center;
- **_ext_in** – optical element extent on the input side, i.e. distance between the input plane and the optical center (positive, in [m]) to be used at wavefront propagation manipulations; if 0, this extent will be calculated internally from optical element parameters
- **_ext_out** – optical element extent on the output side, i.e. distance between the optical center and the output plane (positive, in [m]) to be used at wavefront propagation manipulations; if 0, this extent will be calculated internally from optical element parameters
- **_nvx** – horizontal coordinate of central normal vector
- **_nvy** – vertical coordinate of central normal vector
- **_nvz** – longitudinal coordinate of central normal vector
- **_tvx** – horizontal coordinate of central tangential vector
- **_tvy** – vertical coordinate of central tangential vector
- **_x** – horizontal position of mirror center [m]
- **_y** – vertical position of mirror center [m]
- **_refl** – reflectivity coefficient to set (can be one number or C-aligned flat complex array vs photon energy vs grazing angle vs component (sigma, pi))
- **_n_ph_en** – number of photon energy values for which the reflectivity coefficient is specified
- **_n_ang** – number of grazing angle values for which the reflectivity coefficient is specified
- **_n_comp** – number of electric field components for which the reflectivity coefficient is specified (can be 1 or 2)
- **_ph_en_start** – initial photon energy value for which the reflectivity coefficient is specified

- **_ph_en_fin** – final photon energy value for which the reflectivity coefficient is specified
- **_ph_en_scale_type** – photon energy sampling type ('lin' for linear, 'log' for logarithmic)
- **_ang_start** – initial grazing angle value for which the reflectivity coefficient is specified
- **_ang_fin** – final grazing angle value for which the reflectivity coefficient is specified
- **_ang_scale_type** – angle sampling type ('lin' for linear, 'log' for logarithmic)

class wpg.srwlib.**SRWLOptMirTor**(*_rt=1*, *_rs=1*, *_size_tang=1*, *_size_sag=1*, *_ap_shape='r'*, *_sim_meth=2*, *_npt=500*, *_nps=500*, *_treat_in_out=1*, *_ext_in=0*, *_ext_out=0*, *_nvx=0*, *_nvy=0*, *_nvz=-1*, *_tvx=1*, *_tvy=0*, *_x=0*, *_y=0*, *_refl=1*, *_n_ph_en=1*, *_n_ang=1*, *_n_comp=1*, *_ph_en_start=1000.0*, *_ph_en_fin=1000.0*, *_ph_en_scale_type='lin'*, *_ang_start=0*, *_ang_fin=0*, *_ang_scale_type='lin'*)

Bases: *wpg.srwlib.SRWLOptMir*

Optical Element: Mirror: Toroid (to be developed)

Parameters

- **_rt** – tangential (major) radius [m]
- **_rs** – sagittal (minor) radius [m]
- **_size_tang** – size in tangential direction [m]
- **_size_sag** – size in sagital direction [m]
- **_ap_shape** – shape of aperture in local frame ('r' for rectangular, 'e' for elliptical)
- **_sim_meth** – simulation method (1 for "thin" approximation, 2 for "thick" approximation)
- **_npt** – number of mesh points to represent mirror in tangential direction (used for "thin" approximation)
- **_nps** – number of mesh points to represent mirror in sagital direction (used for "thin" approximation)
- **_treat_in_out** – switch specifying how to treat input and output wavefront before and after the main propagation through the optical element: 0- assume that the input wavefront is defined in the plane before the optical element, and the output wavefront is required in a plane just after the element; 1- assume that the input wavefront is defined in the plane at the optical element center and the output wavefront is also required at the element center; 2- assume that the input wavefront is defined in the plane at the optical element center and the output wavefront is also required at the element center; however, before the propagation though the optical element, the wavefront should be propagated through a drift back to a plane just before the optical element, then a special propagator will bring the wavefront to a plane at the optical element exit, and after this the wavefront will be propagated through a drift back to the element center;
- **_ext_in** – optical element extent on the input side, i.e. distance between the input plane and the optical center (positive, in [m]) to be used at wavefront propagation manipulations; if 0, this extent will be calculated internally from optical element parameters
- **_ext_out** – optical element extent on the output side, i.e. distance between the optical center and the output plane (positive, in [m]) to be used at wavefront propagation manipulations; if 0, this extent will be calculated internally from optical element parameters
- **_nvx** – horizontal coordinate of central normal vector

- **_nvy** – vertical coordinate of central normal vector
- **_nvz** – longitudinal coordinate of central normal vector
- **_tvx** – horizontal coordinate of central tangential vector
- **_tvy** – vertical coordinate of central tangential vector
- **_x** – horizontal position of mirror center [m]
- **_y** – vertical position of mirror center [m]
- **_refl** – reflectivity coefficient to set (can be one number or C-aligned flat complex array vs photon energy vs grazing angle vs component (sigma, pi))
- **_n_ph_en** – number of photon energy values for which the reflectivity coefficient is specified
- **_n_ang** – number of grazing angle values for which the reflectivity coefficient is specified
- **_n_comp** – number of electric field components for which the reflectivity coefficient is specified (can be 1 or 2)
- **_ph_en_start** – initial photon energy value for which the reflectivity coefficient is specified
- **_ph_en_fin** – final photon energy value for which the reflectivity coefficient is specified
- **_ph_en_scale_type** – photon energy sampling type ('lin' for linear, 'log' for logarithmic)
- **_ang_start** – initial grazing angle value for which the reflectivity coefficient is specified
- **_ang_fin** – final grazing angle value for which the reflectivity coefficient is specified
- **_ang_scale_type** – angle sampling type ('lin' for linear, 'log' for logarithmic)

class wpg.srwlib.**SRWLOptShift**(*_shift_x=0*, *_shift_y=0*)

Bases: *wpg.srwlib.SRWLOpt*

Optical Element: Shirt

Parameters

- **_shift_x** – horizontal shift [m]
- **_shift_y** – vertical shift [m]

class wpg.srwlib.**SRWLOptT**(*_nx=1*, *_ny=1*, *_rx=0.001*, *_ry=0.001*, *_arTr=None*, *_extTr=0*, *_Fx=1e+23*, *_Fy=1e+23*, *_x=0*, *_y=0*, *_ne=1*, *_eStart=0*, *_eFin=0*)

Bases: *wpg.srwlib.SRWLOpt*

Optical Element: Transmission (generic)

Parameters

- **_nx** – number of transmission data points in the horizontal direction
- **_ny** – number of transmission data points in the vertical direction
- **_rx** – range of the horizontal coordinate [m] for which the transmission is defined
- **_ry** – range of the vertical coordinate [m] for which the transmission is defined
- **_arTr** – complex C-aligned data array (of 2*ne*nx*ny length) storing amplitude transmission and optical path difference as function of transverse coordinates
- **_extTr** – transmission outside the grid/mesh is zero (0), or it is same as on boundary (1)

- **_Fx** – estimated focal length in the horizontal plane [m]
- **_Fy** – estimated focal length in the vertical plane [m]
- **_x** – horizontal transverse coordinate of center [m]
- **_y** – vertical transverse coordinate of center [m]
- **_ne** – number of transmission data points vs photon energy
- **_eStart** – initial value of photon energy
- **_eFin** – final value of photon energy

allocate(*_ne*, *_nx*, *_ny*)

get_data(*_typ*, *_dep=3*, *_e=0*, *_x=0*, *_y=0*)

Returns Transmission Data Characteristic :param _typ: type of transmission characteristic to extract: 1- amplitude transmission, 2- intensity transmission, 3- optical path difference :param _dep: type of dependence to extract: 0- vs photon energy, 1- vs horizontal position, 2- vs vertical position, 3- vs hor. & vert. positions :param _e: photon energy [eV] (to keep fixed) :param _x: horizontal position [m] (to keep fixed) :param _y: vertical position [m] (to keep fixed)

class wpg.srwlib.**SRWLOptWG**(*_L=1*, *_Dx=0.01*, *_Dy=0.01*, *_x=0*, *_y=0*)

Bases: *wpg.srwlib.SRWLOpt*

Optical Element: Waveguide

Parameters

- **_L** – length [m]
- **_Dx** – horizontal transverse dimension [m]
- **_Dy** – vertical transverse dimension [m]
- **_x** – horizontal transverse coordinate of center [m]
- **_y** – vertical transverse coordinate of center [m]

class wpg.srwlib.**SRWLOptZP**(*_nZones=100*, *_rn=0.0001*, *_thick=1e-05*, *_delta1=1e-06*, *_atLen1=0.1*, *_delta2=0*, *_atLen2=1e-06*, *_x=0*, *_y=0*)

Bases: *wpg.srwlib.SRWLOpt*

Optical Element: Thin Lens

Parameters

- **_nZones** – total number of zones
- **_rn** – auter zone radius [m]
- **_thick** – thickness [m]
- **_delta1** – refractuve index decrement of the “main” material
- **_atLen1** – attenuation length [m] of the “main” material
- **_delta2** – refractuve index decrement of the “complementary” material
- **_atLen2** – attenuation length [m] of the “complementary” material
- **_x** – horizontal transverse coordinate of center [m]
- **_y** – vertical transverse coordinates of center [m]

class wpg.srwlib.**SRWLPartBeam**(*_Iavg=0*, *_nPart=0*, *_partStatMom1=None*, *_arStatMom2=None*)

Bases: object

Particle Beam

Parameters

- **_Iavg** – average current [A]
- **_nPart** – number of electrons (in a bunch)
- **_partStatMom1** – particle type and 1st order statistical moments
- **_arStatMom2** – 2nd order statistical moments [0]: <(x-x0)^2> [1]: <(x-x0)*(xp-xp0)> [2]: <(xp-xp0)^2> [3]: <(y-y0)^2> [4]: <(y-y0)*(yp-yp0)> [5]: <(yp-yp0)^2> [6]: <(x-x0)*(y-y0)> [7]: <(xp-xp0)*(y-y0)> [8]: <(x-x0)*(yp-yp0)> [9]: <(xp-xp0)*(yp-yp0)> [10]: <(E-E0)^2>/E0^2 [11]: <(s-s0)^2> [12]: <(s-s0)*(E-E0)>/E0 [13]: <(x-x0)*(E-E0)>/E0 [14]: <(xp-xp0)*(E-E0)>/E0 [15]: <(y-y0)*(E-E0)>/E0 [16]: <(yp-yp0)*(E-E0)>/E0 [17]: <(x-x0)*(s-s0)> [18]: <(xp-xp0)*(s-s0)> [19]: <(y-y0)*(s-s0)> [20]: <(yp-yp0)*(s-s0)>

drift(*_dist*)

Propagates particle beam statistical moments over a distance in free space :param _dist: distance the beam has to be propagated over [m]

from_RMS(*_Iavg=0*, *_e=0*, *_sig_e=0*, *_sig_x=0*, *_sig_x_pr=0*, *_m_xx_pr=0*, *_sig_y=0*, *_sig_y_pr=0*, *_m_yy_pr=0*)

Sets up particle (electron) beam internal data from Twiss parameters :param _Iavg: average current [A] :param _e: energy [GeV] :param _sig_e: RMS energy spread :param _sig_x: horizontal RMS size [m] :param _sig_x_pr: horizontal RMS divergence [rad] :param _m_xx_pr: <(x-<x>)(x'-<x'>)> [m] :param _sig_y: vertical RMS size [m] :param _sig_y_pr: vertical RMS divergence [rad] :param _m_yy_pr: <(y-<y>)(y'-<y'>)> [m]

from_Twiss(*_Iavg=0*, *_e=0*, *_sig_e=0*, *_emit_x=0*, *_beta_x=0*, *_alpha_x=0*, *_eta_x=0*, *_eta_x_pr=0*, *_emit_y=0*, *_beta_y=0*, *_alpha_y=0*, *_eta_y=0*, *_eta_y_pr=0*)

Sets up particle (electron) beam internal data from Twiss parameters :param _Iavg: average current [A] :param _e: energy [GeV] :param _sig_e: RMS energy spread :param _emit_x: horizontal emittance [m] :param _beta_x: horizontal beta-function [m] :param _alpha_x: horizontal alpha-function [rad] :param _eta_x: horizontal dispersion function [m] :param _eta_x_pr: horizontal dispersion function derivative [rad] :param _emit_y: vertical emittance [m] :param _beta_y: vertical beta-function [m] :param _alpha_y: vertical alpha-function [rad] :param _eta_y: vertical dispersion function [m] :param _eta_y_pr: vertical dispersion function derivative [rad]

class wpg.srwlib.**SRWLParticle**(*_x=0*, *_y=0*, *_z=0*, *_xp=0*, *_yp=0*, *_gamma=1*, *_relE0=1*, *_nq=-1*)

Bases: object

Charged Particle

Parameters

- **_x** – instant coordinates [m]
- **_y** – instant coordinates [m]
- **_z** – instant coordinates [m]
- **_xp** – instant transverse velocity component btx = vx/c (angles for relativistic particle)
- **_yp** – instant transverse velocity component bty = vy/c (angles for relativistic particle)
- **_gamma** – relative energy

- **_relE0** – rest mass (energy) in units of electron rest mass, e.g. 1 for electron, 1836.1526988 (=938.272013/0.510998902) for proton
- **_nq** – charge of the particle related to absolute value of electron charge, -1 for electron, 1 for positron and for proton

drift(*_dist*)

Propagates particle beam statistical moments over a distance in free space :param _dist: distance the beam has to be propagated over [m]

get_E(*_unit='GeV'*)

class wpg.srwlib.**SRWLPrtTrj**(*_arX=None*, *_arXp=None*, *_arY=None*, *_arYp=None*, *_arZ=None*, *_arZp=None*, *_arBx=None*, *_arBy=None*, *_arBz=None*, *_np=0*, *_ctStart=0*, *_ctEnd=0*, *_partInitCond=None*)

Bases: object

Charged Particle Trajectory

Parameters

- **_arX** – array of horizontal position [m]
- **_arXp** – array of horizontal relative velocity (trajectory angle) [rad]
- **_arY** – array of vertical position [m]
- **_arYp** – array of vertical relative velocity (trajectory angle) [rad]
- **_arZ** – array of longitudinal positions [m]
- **_arZp** – array of longitudinal relative velocity [rad]
- **_arBx** – array of horizontal magnetic field component "seen" by particle [T]
- **_arBy** – array of vertical magnetic field component "seen" by particle [T]
- **_arBz** – array of longitudinal magnetic field component "seen" by particle [T]
- **_np** – number of trajectory points
- **_ctStart** – start value of independent variable (c*t) for which the trajectory should be (/is) calculated (is constant step enough?)
- **_ctEnd** – end value of independent variable (c*t) for which the trajectory should be (/is) calculated (is constant step enough?)
- **_partInitCond** – particle type and initial conditions for which the trajectory should be (/is) calculated

allocate(*_np*, *_allB=False*)

save_ascii(*_file_path*)

Auxiliary function to write tabulated Trajectory data to ASCII file

class wpg.srwlib.**SRWLRadMesh**(*_eStart=0*, *_eFin=0*, *_ne=1*, *_xStart=0*, *_xFin=0*, *_nx=1*, *_yStart=0*, *_yFin=0*, *_ny=1*, *_zStart=0*, *_nvx=0*, *_nvy=0*, *_nvz=1*, *_hvx=1*, *_hvy=0*, *_hvz=0*, *_arSurf=None*)

Bases: object

Radiation Mesh (Sampling)

Parameters

- **_eStart** – initial value of photon energy (/time)
- **_eFin** – final value of photon energy (/time)

- **_ne** – number of points vs photon energy (/time)
- **_xStart** – initial value of horizontal position (/angle)
- **_xFin** – final value of horizontal position (/angle)
- **_nx** – number of points vs horizontal position (/angle)
- **_yStart** – initial value of vertical position (/angle)
- **_yFin** – final value of vertical position (/angle)
- **_ny** – number of points vs vertical position (/angle)
- **_zStart** – longitudinal position
- **_nvx** – horizontal lab-frame coordinate of inner normal to observation plane (/ surface in its center)
- **_nvy** – vertical lab-frame coordinate of inner normal to observation plane (/ surface in its center)
- **_nvz** – longitudinal lab-frame coordinate of inner normal to observation plane (/ surface in its center)
- **_hvx** – horizontal lab-frame coordinate of the horizontal base vector of the observation plane (/ surface in its center)
- **_hvy** – vertical lab-frame coordinate of the horizontal base vector of the observation plane (/ surface in its center)
- **_hvz** – longitudinal lab-frame coordinate of the horizontal base vector of the observation plane (/ surface in its center)
- **_arSurf** – array defining the observation surface (as function of 2 variables - x & y - on the mesh given by _xStart, _xFin, _nx, _yStart, _yFin, _ny; to be used in case this surface differs from plane)

set_from_other(*_mesh*)

class wpg.srwlib.**SRWLStokes**(*_arS=None*, *_typeStokes='f'*, *_eStart=0*, *_eFin=0*, *_ne=0*, *_xStart=0*, *_xFin=0*, *_nx=0*, *_yStart=0*, *_yFin=0*, *_ny=0*, *_mutual=0*)

Bases: object

Radiation Stokes Parameters

Parameters

- **_arS** – flat C-aligned array of all Stokes components (outmost loop over Stokes parameter number); NOTE: only 'f' (float) is supported for the moment (Jan. 2012)
- **_typeStokes** – numerical type: 'f' (float) or 'd' (double, not supported yet)
- **_eStart** – initial value of photon energy (/time)
- **_eFin** – final value of photon energy (/time)
- **_ne** – numbers of points vs photon energy
- **_xStart** – initial value of horizontal position
- **_xFin** – final value of photon horizontal position
- **_nx** – numbers of points vs horizontal position
- **_yStart** – initial value of vertical position
- **_yFin** – final value of vertical position

- **_ny** – numbers of points vs vertical position
- **_mutual** – mutual Stokes components (4*(_ne*_nx*_ny_)^2 values)

add_stokes(*_st*, *_n_comp=4*, *_mult=1*, *_meth=0*)
Add Another Stokes structure :param _st: Stokes structure to be added :param _n_comp: number of components to treat :param _mult: multiplier :param _meth: method of adding the Stokes structure _st: 0- simple addition assuming _wfr to have same mesh as this wavefront 1- add using bilinear interpolation (taking into account meshes of the two wavefronts) 2- add using bi-quadratic interpolation (taking into account meshes of the two wavefronts) 3- add using bi-cubic interpolation (taking into account meshes of the two wavefronts)

allocate(*_ne*, *_nx*, *_ny*, *_typeStokes='f'*, *_mutual=0*)

avg_update_interp(*_more_stokes*, *_iter*, *_ord*, *_n_stokes_comp=4*, *_mult=1.0*)
Update this Stokes data structure with new data, contained in the _more_stokes structure, calculated on a different 2D mesh, so that it would represent estimation of average of (_iter + 1) structures :param _more_stokes: Stokes data structure to "add" to the estimation of average :param _iter: number of Stokes structures already "added" previously :param _ord: order of 2D interpolation to use (1- bilinear, . . . , 3- bi-cubic) :param _n_stokes_comp: number of Stokes components to treat (1 to 4) :param _mult: optional multiplier of the _more_stokes

avg_update_interp_mutual(*_more_stokes*, *_iter*, *_n_stokes_comp=4*, *_mult=1.0*)
Update this Stokes data structure with new data, contained in the _more_stokes structure, calculated on a different 2D mesh, so that it would represent estimation of average of (_iter + 1) structures :param _more_stokes: Stokes data structure to "add" to the estimation of average :param _iter: number of Stokes structures already "added" previously :param _n_stokes_comp: number of Stokes components to treat (1 to 4) :param _mult: optional multiplier of the _more_stokes

avg_update_same_mesh(*_more_stokes*, *_iter*, *_n_stokes_comp=4*, *_mult=1.0*)
Update this Stokes data structure with new data, contained in the _more_stokes structure, calculated on the same mesh, so that this structure would represent estimation of average of (_iter + 1) structures :param _more_stokes: Stokes data structure to "add" to the estimation of average :param _iter: number of Stokes structures already "added" previously :param _n_stokes_comp: number of Stokes components to treat (1 to 4) :param _mult: optional multiplier of the _more_stokes

to_int(*_pol=6*)
Calculates / "extracts" intensity at a given polarization from the Stokes components :param _pol: polarization component to extract:

0- Linear Horizontal; 1- Linear Vertical; 2- Linear 45 degrees; 3- Linear 135 degrees; 4- Circular Right; 5- Circular Left; 6- Total

Returns 1D array with (C-aligned) resulting intensity data

class wpg.srwlib.**SRWLWfr**(*_arEx=None*, *_arEy=None*, *_typeE='f'*, *_eStart=0*, *_eFin=0*, *_ne=0*, *_xStart=0*, *_xFin=0*, *_nx=0*, *_yStart=0*, *_yFin=0*, *_ny=0*, *_zStart=0*, *_partBeam=None*)
Bases: object

Radiation Wavefront (Electric Field)

Parameters

- **_arEx** – horizontal complex electric field component array; NOTE: only 'f' (float) is supported for the moment (Jan. 2011)
- **_arEy** – vertical complex electric field component array
- **_typeE** – electric field numerical type: 'f' (float) or 'd' (double)

- **_eStart** – initial value of photon energy (/time)
- **_eFin** – final value of photon energy (/time)
- **_ne** – numbers of points vs photon energy
- **_xStart** – initial value of horizontal positions
- **_xFin** – final value of horizontal positions
- **_nx** – numbers of points vs horizontal positions
- **_yStart** – initial vertical positions
- **_yFin** – final value of vertical positions
- **_ny** – numbers of points vs vertical positions
- **_zStart** – longitudinal position
- **_partBeam** – particle beam source; strictly speaking, it should be just SRWLParticle; however, "multi-electron" information can appear useful for those cases when "multi-electron intensity" can be deduced from the "single-electron" one by convolution

Some additional parameters, that are not included in constructor arguments: Rx, Ry: instant wavefront radii dRx, dRy: error of wavefront radii xc, yc: transverse coordinates of wavefront instant "source center" avgPhotEn: average photon energy for time-domain simulations presCA: presentation/domain: 0- coordinates, 1- angles presFT: presentation/domain: 0- frequency (photon energy), 1- time unitElFld: electric field units: 0- arbitrary, 1- sqrt(Phot/s/0.1%bw/mm^2) arElecPropMatr: effective 1st order "propagation matrix" for electron beam parameters arMomX, arMomY: statistical moments (of Wigner distribution); to check the exact number of moments required arWfrAuxData: array of auxiliary wavefront data

addE(*_wfr*, *_meth=0*)

Add Another Electric Field Wavefront :param _wfr: wavefront to be added :param _meth: method of adding the wavefront _wfr: 0- simple addition assuming _wfr to have same mesh as this wavefront 1- add using bilinear interpolation (taking into account meshes of the two wavefronts) 2- add using bi-quadratic interpolation (taking into account meshes of the two wavefronts) 3- add using bi-cubic interpolation (taking into account meshes of the two wavefronts)

allocate(*_ne*, *_nx*, *_ny*, *_EXNeeded=1*, *_EYNeeded=1*, *_typeE='f'*, *_backupNeeded=0*)

Allocate Electric Field data :param _ne: number of points vs photon energy / time :param _nx: number of points vs horizontal position / angle :param _ny: number of points vs vertical position / angle :param _EXNeeded: switch specifying whether Ex data is necessary or not (1 or 0) :param _EYNeeded: switch specifying whether Ey data is necessary or not (1 or 0) :param _typeE: numerical type of Electric Field data: float (single precision) or double ('f' or 'd'); double is not yet supported :param _backupNeeded: switch specifying whether backup of Electric Field data (arExAux, arEyAux) should be created or not (1 or 0)

calc_stokes(*_stokes*)

Calculate Stokes parameters from Electric Field

delE(*_type=0*, *_treatEX=1*, *_treatEY=1*)

Delete Electric Field data :param _type: type of data to be deleted: 0- arEx, arEy, arExAux, arEyAux; 1- arEx, arEy only; 2- arExAux, arEyAux only :param _treatEX: switch specifying whether Ex data should be deleted or not (1 or 0) :param _treatEY: switch specifying whether Ey data should be deleted or not (1 or 0)

wpg.srwlib.**srwl_opt_setup_CRL**(*_foc_plane*, *_delta*, *_atten_len*, *_shape*, *_apert_h*, *_apert_v*, *_r_min*, *_n*, *_wall_thick*, *_xc*, *_yc*, *_void_cen_rad=None*, *_e_start=0*, *_e_fin=0*, *_nx=1001*, *_ny=1001*)

Setup Transmission type Optical Element which simulates Compound Refractive Lens (CRL) :param _foc_plane: plane of focusing: 1- horizontal, 2- vertical, 3- both :param _delta: refractive index decrement

(can be one number of array vs photon energy) :param _atten_len: attenuation length [m] (can be one number of array vs photon energy) :param _shape: 1- parabolic, 2- circular (spherical) :param _apert_h: horizontal aperture size [m] :param _apert_v: vertical aperture size [m] :param _r_min: radius (on tip of parabola for parabolic shape) [m] :param _n: number of lenses (/"holes") :param _wall_thick: min. wall thickness between "holes" [m] :param _xc: horizontal coordinate of center [m] :param _yc: vertical coordinate of center [m] :param _void_cen_rad: flat array/list of void center coordinates and radii: [x1, y1, r1, x2, y2, r2,...] :param _e_start: initial photon energy :param _e_fin: final photon energy :return: transmission (SRWLOptT) type optical element which simulates CRL

wpg.srwlib.**srwl_opt_setup_cyl_fiber**(*_foc_plane*, *_delta_ext*, *_delta_core*, *_atten_len_ext*, *_atten_len_core*, *_diam_ext*, *_diam_core*, *_xc*, *_yc*)

Setup Transmission type Optical Element which simulates Cylindrical Fiber :param _foc_plane: plane of focusing: 1- horizontal (i.e. fiber is parallel to vertical axis), 2- vertical (i.e. fiber is parallel to horizontal axis) :param _delta_ext: refractive index decrement of extenal layer :param _delta_core: refractive index decrement of core :param _atten_len_ext: attenuation length [m] of external layer :param _atten_len_core: attenuation length [m] of core :param _diam_ext: diameter [m] of external layer :param _diam_core: diameter [m] of core :param _xc: horizontal coordinate of center [m] :param _yc: vertical coordinate of center [m] :return: transmission (SRWLOptT) type optical element which simulates Cylindrical Fiber

wpg.srwlib.**srwl_opt_setup_surf_height_1d**(*_height_prof_data*, *_dim*, *_ang*, *_ang_r=0*, *_amp_coef=1*, *_ar_arg_long=None*, *_nx=0*, *_ny=0*, *_size_x=0*, *_size_y=0*, *_xc=0*, *_yc=0*)

Setup Transmission type optical element with 1D (mirror or grating) surface Heght Profile data :param _height_prof_data: two- or one-column table containing, in case of two columns: longitudinal position in [m] (1st column) and the Height Profile in [m] (2nd column) data; in case of one column, it contains the Height Profile data :param _dim: orientation of the reflection (deflection) plane; can be 'x' or 'y' :param _ang: grazing angle (between input optical axis and mirror/grating plane) :param _ang_r: reflection angle (between output optical axis and mirror/grating plane) :param _amp_coef: height profile "amplification coefficient" :param _ar_arg_long: optional array of longitudinal position (along mirror/grating) in [m]; if _ar_arg_long != None, any longitudinal position contained in _height_prof_data is ignored :param _nx: optional number of points in horizontal dimension of the output transmission optical element :param _ny: optional number of points in vertical dimension of the output transmission optical element :param _size_x: optional horizontal transverse size of the output transmission optical element (if <=0: _height_prof_data, _dim, _ar_arg_long, _ar_arg_tr data is used) :param _size_y: optional vertical transverse size of the output transmission optical element (if <=0: _height_prof_data, _dim, _ar_arg_long, _ar_arg_tr data is used) :param _xc: optional horizontal center position of the output transmission optical element :param _yc: optional vertical center position of the output transmission optical element :return: transmission (SRWLOptT) type optical element which simulates the effect of surface height error

wpg.srwlib.**srwl_opt_setup_surf_height_2d**(*_height_prof_data*, *_dim*, *_ang*, *_ang_r=0*, *_amp_coef=1*, *_ar_arg_long=None*, *_ar_arg_tr=None*, *_nx=0*, *_ny=0*, *_size_x=0*, *_size_y=0*)

Setup Transmission type optical element with 2D (mirror or grating) surface Heght Profile data :param _height_prof_data: a matrix (2D array) containing the Height Profile data in [m]; if _ar_height_prof_x==None and _ar_height_prof_y==None: the first column in _height_prof_data is assumed to be the "longitudinal" position [m] and first row the "transverse" position [m], and _height_prof_data[0][0] is not used; otherwise the "longitudinal" and "transverse" positions on the surface are assumed to be given by _ar_height_prof_x, _ar_height_prof_y :param _dim: orientation of the reflection (deflection) plane; can be 'x' or 'y' :param _ang: grazing angle (between input optical axis and mirror/grating plane) :param _ang_r: reflection angle (between output optical axis and mirror/grating plane) :param _amp_coef: height profile "amplification coefficient" :param _ar_arg_long: optional array of longitudinal position (along mirror/grating) in [m] :param _ar_arg_tr: optional array of transverse position on mirror/grating surface in [m] :param _nx: optional number of points in horizontal dimension of the output transmission optical element :param _ny: optional number of points in vertical dimension of the output transmission optical element :param _size_x: optional horizontal transverse size of the output transmission optical element (if <=0: _height_prof_data, _dim, _ar_arg_long, _ar_arg_tr

data is used) :param _size_y: optional vertical transverse size of the output transmission optical element (if <=0: _height_prof_data, _dim, _ar_arg_long, _ar_arg_tr data is used) :return: transmission (SRWLOptT) type optical element which simulates the effect of surface height error

wpg.srwlib.**srwl_uti_array_alloc**(*_type*, *_n*)

wpg.srwlib.**srwl_uti_math_seq_halton**(*i*, *base=2*)

wpg.srwlib.**srwl_uti_num_round**(*_x*, *_ndig=8*)

wpg.srwlib.**srwl_uti_ph_en_conv**(*_x*, *_in_u='keV'*, *_out_u='nm'*)
Photon Energy <-> Wavelength conversion :param _x: value to be converted :param _in_u: input unit :param _out_u: output unit :return: value in the output units

wpg.srwlib.**srwl_uti_proc_is_master**()
Check if process is Master (in parallel processing sense)

wpg.srwlib.**srwl_uti_rand_fill_vol**(*_np*, *_x_min*, *_x_max*, *_nx*, *_ar_y_vs_x_min*, *_ar_y_vs_x_max*, *_y_min*, *_y_max*, *_ny*, *_ar_z_vs_xy_min*, *_ar_z_vs_xy_max*)
Generate coordinates of ponts randomly filling 3D volume limited by two arbitrary curves (defining base) and two surfaces :param _np: number of random points in rectangular parallelepiped to try :param _x_min: min. x coordinate :param _x_max: max. x coordinate :param _nx: number of points vs x coord. :param _ar_y_vs_x_min: min. y vs x array :param _ar_y_vs_x_max: max. y vs x array :param _y_min: min. y coordinate :param _y_max: max. y coordinate :param _ny: number of points vs y coord. :param _ar_z_vs_xy_min: min. z vs x and y flat 2D array :param _ar_z_vs_xy_max: max. z vs x and y flat 2D array :return: flat array of point coordinates: array('d', [x1,y1,z1,x2,y2,z2,...])

wpg.srwlib.**srwl_uti_read_data_cols**(*_file_path*, *_str_sep*, *_i_col_start=0*, *_i_col_end=-1*, *_n_line_skip=0*)
Auxiliary function to read-in data comumns from ASCII file (2D table) :param _file_path: full path (including file name) to the file :param _str_sep: column separation symbol(s) (string) :param _i_col_start: initial data column to read :param _i_col_end: final data column to read :param _n_line_skip: number of lines to skip in the beginning of the file :return: 2D list containing data columns read

wpg.srwlib.**srwl_uti_read_intens_ascii**(*_file_path*, *_num_type='f'*)

wpg.srwlib.**srwl_uti_read_mag_fld_3d**(*_fpath*, *_scom='#'*)

wpg.srwlib.**srwl_uti_save_intens_ascii**(*_ar_intens, _mesh, _file_path, _n_stokes=1, _arLabels=['Photon Energy', 'Horizontal Position', 'Vertical Position', 'Intensity'], _arUnits=['eV', 'm', 'm', 'ph/s/.1%bw/mm^2'], _mutual=0*)

wpg.srwlib.**srwl_uti_save_text**(*_text*, *_file_path*)

wpg.srwlib.**srwl_uti_write_data_cols**(*_file_path*, *_cols*, *_str_sep*, *_str_head=None*, *_i_col_start=0*, *_i_col_end=-1*)
Auxiliary function to write tabulated data (columns, i.e 2D table) to ASCII file :param _file_path: full path (including file name) to the file to be (over-)written :param _cols: array of data columns to be saves to file :param _str_sep: column separation symbol(s) (string) :param _str_head: header (string) to write before data columns :param _i_col_start: initial data column to write :param _i_col_end: final data column to write

wpg.srwlib.**srwl_wfr_emit_prop_multi_e**(*_e_beam*, *_mag*, *_mesh*, *_sr_meth*, *_sr_rel_prec*, *_n_part_tot*, *_n_part_avg_proc=1*, *_n_save_per=100*, *_file_path=None*, *_sr_samp_fact=-1*, *_opt_bl=None*, *_pres_ang=0*, *_char=0*, *_x0=0*, *_y0=0*, *_e_ph_integ=0*, *_rand_meth=1*, *_tryToUseMPI=True*)
Calculate Stokes Parameters of Emitted (and Propagated, if beamline is defined) Partially-Coherent SR :param _e_beam: Finite-Emittance e-beam (SRWLPartBeam type) :param _mag: Magnetic Field container (magFldCnt

type) :param _mesh: mesh vs photon energy, horizontal and vertical positions (SRWLRadMesh type) on which initial SR should be calculated :param _sr_meth: SR Electric Field calculation method to be used (0- "manual", 1- "auto-undulator", 2- "auto-wiggler") :param _sr_rel_prec: relative precision for SR Electric Field calculation (usually 0.01 is OK, the smaller the more accurate) :param _n_part_tot: total number of "macro-electrons" to be used in the calculation :param _n_part_avg_proc: number of "macro-electrons" to be used in calculation at each "slave" before sending Stokes data to "master" (effective if the calculation is run via MPI) :param _n_save_per: periodicity of saving intermediate average Stokes data to file by master process :param _file_path: path to file for saving intermediate average Stokes data by master process :param _sr_samp_fact: oversampling factor for calculating of initial wavefront for subsequent propagation (effective if >0) :param _opt_bl: optical beamline (container) to propagate the radiation through (SRWLOptC type) :param _pres_ang: switch specifying presentation of the resulting Stokes parameters: coordinate (0) or angular (1) :param _char: radiation characteristic to calculate: 0- Intensity (s0); 1- Four Stokes components; 2- Mutual Intensity Cut vs X; 3- Mutual Intensity Cut vs Y; 4- Mutual Intensity Cut vs X & Y; 10- Flux :param _x0: horizontal center position for mutual intensity calculation :param _y0: vertical center position for mutual intensity calculation :param _e_ph_integ: integration over photon energy is required (1) or not (0); if the integration is required, the limits are taken from _mesh :param _rand_meth: method for generation of pseudo-random numbers for e-beam phase-space integration:

1- standard pseudo-random number generator 2- Halton sequences 3- LPtau sequences (to be implemented)

Parameters **_tryToUseMPI** – switch specifying whether MPI should be attempted to be used

1.7 wpg.beamline module

class wpg.beamline.**Beamline**(*srwl_beamline=None*)

Bases: object

Set of optical elements and propagation parameters.

Init beamline.

Params srwl_beamline if present will used for initialization.

append(*optical_element*, *propagation_parameters*)

Appends optical element and propagation propagation parameters to the end of beamline

Parameters

- **optical_element** – SRW or wpg optical element
- **propagation_parameters** – SRW propagation parameters list or wpg.optical_elements.UsePP object

propagate(*wfr*)

Propagate wavefront through beamline.

Parameters **wfr** (wpg.wavefront.Wavefront) – Input wavefront (will be re-writed after propagation)

1.8 wpg.optical_elements module

This module contains definitions custom optical elements.

Described mapping (or aliases) of some of SRW optical elements (SRWLOpt* <-> wpg)

wpg.optical_elements.**Aperture**(*shape*, *ap_or_ob*, *Dx*, *Dy=1e+23*, *x=0*, *y=0*)

Defining an aperture/obstacle propagator: A wrapper to a SRWL function SRWLOptA()

Parameters

- **shape** – 'r' for rectangular, 'c' for circular
- **ap_or_ob** – 'a' for aperture, 'o' for obstacle
- **Dy** (*Dx,*) – transverse dimensions [m]; in case of circular aperture, only Dx is used for diameter
- **y** (*x,*) – transverse coordinates of center [m]

Returns opAp - aperture propagator, `struct SRWLOptA`

`wpg.optical_elements.`**CRL**(*_foc_plane*, *_delta*, *_atten_len*, *_shape*, *_apert_h*, *_apert_v*, *_r_min*, *_n*, *_wall_thick*, *_xc*, *_yc*, *_void_cen_rad=None*, *_e_start=0*, *_e_fin=0*, *_nx=1001*, *_ny=1001*)

Setup Transmission type Optical Element which simulates Compound Refractive Lens (CRL).

Parameters

- **_foc_plane** – plane of focusing: 1- horizontal, 2- vertical, 3- both
- **_delta** – refractive index decrement (can be one number of array vs photon energy)
- **_atten_len** – attenuation length [m] (can be one number of array vs photon energy)
- **_shape** – 1- parabolic, 2- circular (spherical)
- **_apert_h** – horizontal aperture size [m]
- **_apert_v** – vertical aperture size [m]
- **_r_min** – radius (on tip of parabola for parabolic shape) [m]
- **_n** – number of lenses (/"holes")
- **_wall_thick** – min. wall thickness between "holes" [m]
- **_xc** – horizontal coordinate of center [m]
- **_yc** – vertical coordinate of center [m]
- **_void_cen_rad** – flat array/list of void center coordinates and radii: [x1, y1, r1, x2, y2, r2,...]
- **_e_start** – initial photon energy
- **_e_fin** – final photon energy

Returns transmission (SRWLOptT) type optical element which simulates CRL

class `wpg.optical_elements.`**Empty**

Bases: *wpg.optical_elements.WPGOpticalElement*

Optical element: Empty. This is empty propagator used for sampling and zooming wavefront

propagate(*wfr*, *propagation_parameters*)

Propagate wavefront through empty propagator, used for sampling and resizing wavefront

`wpg.optical_elements.`**Mirror_elliptical**(*orient*, *p*, *q*, *thetaE*, *theta0*, *length*)

Defining a plane elliptical focusing mirror propagator: A wrapper to a SRWL function SRWLOptMirEl()

Parameters

- **orient** – mirror orientation, 'x' (horizontal) or 'y' (vertical)
- **p** – distance to one ellipsis center (source), [m]
- **q** – distance to the other ellipsis center (focus), [m]

- **thetaE** – design incidence angle in the center of mirror, [rad]
- **theta0** – "real" incidence angle in the center of mirror, [rad]
- **length** – mirror length, [m]

Returns opEFM - elliptical mirror propagator, `struct SRWLOptMirEl`

wpg.optical_elements.**Mirror_plane**(*orient*, *theta*, *length*, *range_xy*, *filename*, *scale=1*, *delim='\t'*)

Defining a plane mirror propagator with taking into account surface height errors

Parameters

- **orient** – mirror orientation, 'x' (horizontal) or 'y' (vertical)
- **theta** – incidence angle [rad]
- **length** – mirror length, [m]

Range_xy range in which the incident WF defined [m]

Filename full file name with mirror profile of two columns, x and h(x) - heigh errors [m]

Scale scale factor, optical path difference OPD = 2*h*scale*sin(theta)

Delim delimiter between data columns

Returns opIPM - imperfect plane mirror propagator

class wpg.optical_elements.**Use_PP**(*auto_resize_before=None*, *auto_resize_after=None*, *releative_precision=None*, *semi_analytical_treatment=None*, *fft_resizing=None*, *zoom=None*, *zoom_h=None*, *zoom_v=None*, *sampling=None*, *sampling_h=None*, *sampling_v=None*, *srw_pp=None*)

Bases: `object`

Short version of propagation parameters. Should be used with *wpg.beamline.Beamline*

Params srw_pp propagation parameters in srw style

auto_resize_after
Auto-Resize (1) or not (0) After propagation

auto_resize_before
Auto-Resize (1) or not (0) Before propagation

fft_resizing
Do any Resizing on Fourier side, using FFT, (1) or not (0)

get_srw_pp()
Return SRW propagation parameters list. Useful for interoperations with SRW tools.

Returns list of floats (propagation parameters)

releative_precision
Relative Precision for propagation with Auto-Resizing (1. is nominal)

sampling
Resolution modification factor at Resizing

sampling_h
Horizontal Resolution modification factor at Resizing

sampling_v
Vertical Resolution modification factor at Resizing

semi_analytical_treatment
Allow (1) or not (0) for semi-analytical treatment of quadratic phase terms at propagation

zoom
Range modification factor at Resizing (1. means no modification)

zoom_h
Horizontal Range modification factor at Resizing (1. means no modification)

zoom_v
Vertical Range modification factor at Resizing

wpg.optical_elements.**VLS_grating**(*_mirSub*, *_m=1*, *_grDen=100*, *_grDen1=0*, *_grDen2=0*, *_grDen3=0*, *_grDen4=0*, *_grAng=0*)
Optical Element: Grating.

param _mirSub: SRWLOptMir (or derived) type object defining substrate of the grating :param _m: output (diffraction) order :param _grDen: groove density [lines/mm] (coefficient a0 in the polynomial groove density: a0 + a1*y + a2*y^2 + a3*y^3 + a4*y^4) :param _grDen1: groove density polynomial coefficient a1 [lines/mm^2] :param _grDen2: groove density polynomial coefficient a2 [lines/mm^3] :param _grDen3: groove density polynomial coefficient a3 [lines/mm^4] :param _grDen4: groove density polynomial coefficient a4 [lines/mm^5] :param _grAng: angle between the grove direction and the saggital direction of the substrate [rad] (by default, groves are made along saggital direction (_grAng=0))

wpg.optical_elements.**WF_dist**(*nx*, *ny*, *Dx*, *Dy*)
Create a 'phase screen' propagator for wavefront distortions: A wrapper to SRWL struct SRWLOptT

Params nx number of points in horizontal direction

Params ny number of points in vertical direction

Params Dx size in m

Params Dy size in

class wpg.optical_elements.**WPGOpticalElement**
Bases: object

Base class for optical elements.

wpg.optical_elements.**Xtal**(*_d_sp=5.4309*, *_psi0r=-1.446e-05*, *_psi0i=3.202e-07*, *_psi_hr=8.8004e-06*, *_psi_hi=3.0808e-07*, *_psi_hbr=8.8004e-06*, *_psi_hbi=3.0808e-07*, *_tc=0.0001*, *_ang_as=0*)
Optical Element: Crystal.

Parameters

- **_d_sp** – (_d_space) crystal reflecting planes d-spacing (John's dA) [A]
- **_psi0r** – real part of 0-th Fourier component of crystal polarizability (John's psi0c.real) (units?)
- **_psi0i** – imaginary part of 0-th Fourier component of crystal polarizability (John's psi0c.imag) (units?)
- **_psi_hr** – (_psiHr) real part of H-th Fourier component of crystal polarizability (John's psihc.real) (units?)
- **_psi_hi** – (_psiHi) imaginary part of H-th Fourier component of crystal polarizability (John's psihc.imag) (units?)
- **_psi_hbr** – (_psiHBr:) real part of -H-th Fourier component of crystal polarizability (John's psimhc.real) (units?)

- **_psi_hbi** – (_psiHBi:) imaginary part of -H-th Fourier component of crystal polarizability (John's psimhc.imag) (units?)
- **_tc** – crystal thickness [m] (John's thicum)
- **_ang_as** – (_Tasym) asymmetry angle [rad] (John's alphdg)
- **_nvx** – horizontal coordinate of outward normal to crystal surface (John's angles: thdg, chidg, phidg)
- **_nvy** – vertical coordinate of outward normal to crystal surface (John's angles: thdg, chidg, phidg)
- **_nvz** – longitudinal coordinate of outward normal to crystal surface (John's angles: thdg, chidg, phidg)
- **_tvx** – horizontal coordinate of central tangential vector (John's angles: thdg, chidg, phidg)
- **_tvy** – vertical coordinate of central tangential vector (John's angles: thdg, chidg, phidg)

wpg.optical_elements.**append_Xtal**(*bl*, *xtal='C'*, *h=1*, *k=1*, *l=1*, *tc=0.0001*, *_dE=0.0*, *doPrint=False*)

Append to a beamline Xtal propagator

Parameters **bl** – Beamline() structure

wpg.optical_elements.**calculateOPD**(*wf_dist*, *mdatafile*, *ncol*, *delim*, *Orient*, *theta*, *scale=1.0*, *stretching=1.0*)

Calculates optical path difference (OPD) from mirror profile and fills the struct wf_dist (`struct SRWLOptT`) for wavefront distortions

Params wf_dist struct SRWLOptT

Params mdatafile an ascii file with mirror profile data

Params ncol number of columns in the file

Params delim delimiter between numbers in an row, can be space (' '), tab ' ', etc

Params orient mirror orientation, 'x' (horizontal) or 'y' (vertical)

Params theta incidence angle

Params scale scaling factor for the mirror profile

Parameters **stretching** – scaling factor for the mirror profile x-axis (@TODO a hack, should be removed ASAP)

:return filled

wpg.optical_elements.**define_Xtal**(*xtal='C'*, *h=4*, *k=0*, *l=0*, *tc=0.0001*, *idx=0*, *_dE=0.0*, *doPrint=False*)

Parameters

- **xtal** – crystal type (now only 'Si' and 'C' are suppoted)
- **h,k,l** – Miller indices
- **tc** – crystal thickness, [m]
- **idx** – the given photon energy line number in crystal parameters tables (ascii files '<xtal>_xih_<hkl>.dat' and '<xtal>_xi0.dat')
- **_dE** – photon energy offset, eV
- **doPrint** – if True additional printouts for crystal orientation matrixes

:return

wpg.optical_elements.**Drift**
alias of *wpg.srwlib.SRWLOptD*

wpg.optical_elements.**Lens**
alias of *wpg.srwlib.SRWLOptL*

class wpg.optical_elements.**Aperture**
Defining an aperture/obstacle propagator: A wrapper to a SRWL function SRWLOptA()

Parameters

- **shape** – 'r' for rectangular, 'c' for circular
- **ap_or_ob** – 'a' for aperture, 'o' for obstacle
- **Dy** (*Dx,*) – transverse dimensions [m]; in case of circular aperture, only Dx is used for diameter
- **y** (*x,*) – transverse coordinates of center [m]

Returns opAp - aperture propagator, `struct SRWLOptA`

class wpg.optical_elements.**WF_dist**
Create a 'phase screen' propagator for wavefront distortions: A wrapper to SRWL struct SRWLOptT

Params nx number of points in horizontal direction

Params ny number of points in vertical direction

Params Dx size in m

Params Dy size in

1.9 wpg.generators module

wpg.generators.**build_gauss_wavefront**(*nx*, *ny*, *nz*, *ekev*, *xMin*, *xMax*, *yMin*, *yMax*, *tau*, *sigX*, *sigY*, *d2waist*, *pulseEn=None*, *pulseRange=None*, *_mx=None*, *_my=None*)
Build 3D Gaussian beam.

Parameters

- **nx** – Number of point along x-axis
- **ny** – Number of point along y-axis
- **nz** – Number of point along z-axis (slices)
- **ekev** – Energy in kEv
- **xMin** – Initial Horizontal Position [m]
- **xMax** – Final Horizontal Position [m]
- **yMin** – Initial Vertical Position [m]
- **yMax** – Final Vertical Position [m]
- **tau** – Pulse duration [s]
- **sigX** – Horiz. RMS size at Waist [m]
- **sigY** – Vert. RMS size at Waist [m]
- **d2waist** – Distance to Gaussian waist

- **pulseEn** – Energy per Pulse [J]
- **pulseRange** – pulse duration range in sigT's
- **_mx** – transverse Gauss-Hermite mode order in horizontal direction
- **_my** – transverse Gauss-Hermite mode order in vertical direction

Returns wpg.Wavefront structure

wpg.generators.**build_gauss_wavefront_xy**(*nx*, *ny*, *ekev*, *xMin*, *xMax*, *yMin*, *yMax*, *sigX*, *sigY*, *d2waist*, *xoff=0.0*, *yoff=0.0*, *tiltX=0.0*, *tiltY=0.0*, *pulseEn=None*, *pulseTau=None*, *repRate=None*, *_mx=None*, *_my=None*)

Build 2D Gaussian beam.

Parameters

- **nx** – Number of point along x-axis
- **ny** – Number of point along y-axis
- **nz** – Number of point along z-axis (slices)
- **ekev** – Energy in kEv
- **xMin** – Initial Horizontal Position [m]
- **xMax** – Final Horizontal Position [m]
- **yMin** – Initial Vertical Position [m]
- **yMax** – Final Vertical Position [m]
- **sigX** – Horiz. RMS size at Waist [m]
- **sigY** – Vert. RMS size at Waist [m]
- **d2waist** – distance to Gaussian waist
- **xoff** – Horizonal Coordinate of Gaussian Beam Center at Waist [m]
- **yoff** – Vertical Coordinate of Gaussian Beam Center at Waist [m]
- **tiltX** – Average Angle of Gaussian Beam at Waist in Horizontal plane [rad]
- **tiltY** – Average Angle of Gaussian Beam at Waist in Vertical plane [rad]
- **pulseEn** – Energy per Pulse [J]
- **pulseTau** – Coherence time [s] to get proper BW
- **_mx** – transverse Gauss-Hermite mode order in horizontal direction
- **_my** – transverse Gauss-Hermite mode order in vertical direction

Returns wpg.Wavefront structure

wpg.generators.**build_gauss_wavefront_xy_**(*xoff*, *yoff*, *tiltX*, *tiltY*, *nx*, *ny*, *ekev*, *xMin*, *xMax*, *yMin*, *yMax*, *sigX*, *sigY*, *d2waist*)

This function depricated and will removed in next releases, use build_gauss_wavefront_xy instead.

1.10 WPG tutorials

1.10.1 Wavefront propagation simulation tutorial

L.Samoylova liubov.samoylova@xfel.eu, A.Buzmakov buzmakov@gmail.com

Tutorial course on Wavefront Propagation Simulations, 28/11/2013, European XFEL, Hamburg.

Wave optics software is based on SRW core library https://github.com/ochubar/SRW, available through WPG interactive framework https://github.com/samoylv/WPG

Propagation Gaussian through horizontal offset mirror (HOM)

Import modules

```
%matplotlib inline
```

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

#Importing necessary modules:
import os
import sys
sys.path.insert(0,os.path.join('..', '..'))

import time
import copy
import numpy as np
import pylab as plt


#import SRW core functions
from wpg.srwlib import srwl,SRWLOptD,SRWLOptA,SRWLOptC,SRWLOptT,SRWLOptL

#import SRW helpers functions
from wpg.useful_code.srwutils import AuxTransmAddSurfHeightProfileScaled

#import some helpers functions
from wpg.useful_code.wfrutils import calculate_fwhm_x, plot_wfront, calculate_fwhm_y,␣
↪print_beamline, get_mesh

#Import base wavefront class
from wpg import Wavefront

#Gaussian beam generator
from wpg.generators import build_gauss_wavefront_xy

plt.ion()
```

Define auxiliary functions

```
#Plotting
def plot_1d(profile, title_fig, title_x, title_y):
    plt.plot(profile[0], profile[1])
    plt.xlabel(title_x)
    plt.ylabel(title_y)
    plt.title(title_fig)
    plt.grid(True)


def plot_2d(amap, xmin, xmax, ymin, ymax, title_fig, title_x, title_y):
    plt.imshow(amap, extent=(ymin, ymax, xmin, xmax))
    plt.colorbar()
    plt.xlabel(title_x)
    plt.ylabel(title_y)
    plt.title(title_fig)
```

```
#calculate source size from photon energy and FWHM angular divergence
def calculate_source_fwhm(ekev, theta_fwhm):
    wl = 12.39e-10/ekev
    k = 2 * np.sqrt(2*np.log(2))
    theta_sigma = theta_fwhm /k
    sigma0 = wl /(2*np.pi*theta_sigma)
    return sigma0*k
```

```
#calculate angular divergence using formula from CDR2011
def calculate_theta_fwhm_cdr(ekev,qnC):
    theta_fwhm = (17.2 - 6.4 * np.sqrt(qnC))*1e-6/ekev**0.85
    return theta_fwhm
```

```
#define optical path difference (OPD) from mirror profile, i.e.
#fill the struct opTrErMirr
#input:
#    mdatafile: an ascii file with mirror profile data
#    ncol:      number of columns in the file
#    delim:     delimiter between numbers in an row, can be space (' '), tab '\t', etc
#    Orient:    mirror orientation, 'x' (horizontal) or 'y' (vertical)
#    theta:     incidence angle
#    scale:     scaling factor for the mirror profile
def defineOPD(opTrErMirr, mdatafile, ncol, delim, Orient, theta, scale):
    heightProfData = np.loadtxt(mdatafile).T
    AuxTransmAddSurfHeightProfileScaled(opTrErMirr, heightProfData, Orient, theta,␣
↪scale)
    plt.figure()
    plot_1d(heightProfData,'profile from ' + mdatafile,'x (m)', 'h (m)') #@todo add␣
↪the func def in on top of example
```

Defining initial wavefront and writing electric field data to h5-file

```
# #**********************Input Wavefront Structure and Parameters
print('*****defining initial wavefront and writing electric field data to h5-file...')
strInputDataFolder = 'data_common'  # input data sub-folder name
strOutputDataFolder = 'Tutorial_intro'  # output data sub-folder name
```

(continues on next page)

(continued from previous page)

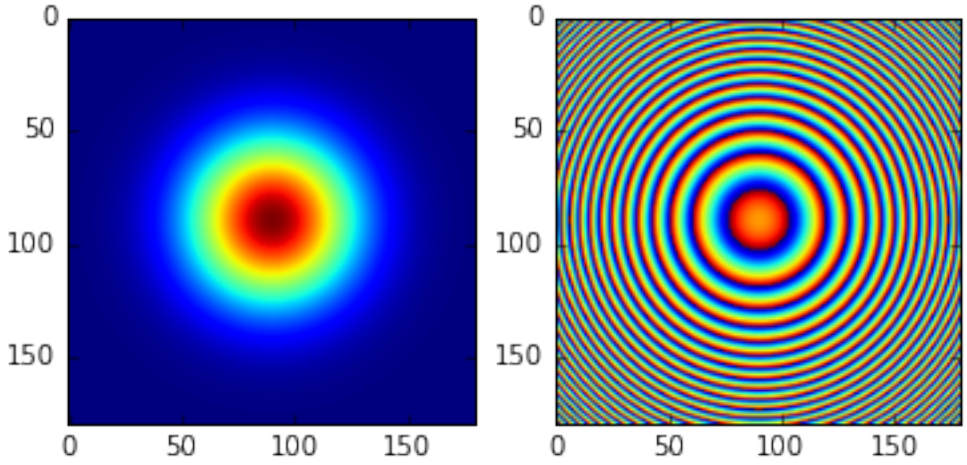
```
#init Gauusian beam parameters
d2m1_sase1 = 246.5
d2m1_sase2 = 290.0
d2m1_sase3 = 281.0
qnC = 0.1                       # e-bunch charge, [nC]
ekev_sase1 = 8.0
thetaOM_sase1 = 2.5e-3          # @check!
ekev_sase3 = 3.0
thetaOM_sase3 = 9.e-3

ekev = ekev_sase1
thetaOM = thetaOM_sase1
d2m1 = d2m1_sase1
#ekev = ekev_sase3
#thetaOM = thetaOM_sase3
#d2m1 = d2m1_sase3
z1 = d2m1
theta_fwhm = calculate_theta_fwhm_cdr(ekev,qnC)
k = 2*np.sqrt(2*np.log(2))
sigX = 12.4e-10*k/(ekev*4*np.pi*theta_fwhm)
print('waist_fwhm [um], theta_fwhms [urad]:', sigX*k*1e6, theta_fwhm*1e6)
#define limits
range_xy = theta_fwhm/k*z1*7. # sigma*7 beam size
npoints=180

#define unique filename for storing results
ip = np.floor(ekev)
frac = np.floor((ekev - ip)*1e3)
fname0 = 'g' + str(int(ip))+'_'+str(int(frac))+'kev'
print('save hdf5: '+fname0+'.h5')
ifname = os.path.join(strOutputDataFolder,fname0+'.h5')

#build SRW gauusian wavefront
wfr0=build_gauss_wavefront_xy(nx=npoints,ny=npoints,ekev=ekev,
                              xMin=-range_xy/2, xMax=range_xy/2,
                              yMin=-range_xy/2, yMax=range_xy/2,
                              sigX=sigX, sigY=sigX, d2waist=z1)



#init WPG Wavefront helper class
mwf = Wavefront(wfr0)

#store wavefront to HDF5 file
mwf.store_hdf5(ifname)

#draw wavefront with common functions
plt.subplot(1,2,1)
plt.imshow(mwf.get_intensity(slice_number=0))
plt.subplot(1,2,2)
plt.imshow(mwf.get_phase(slice_number=0,polarization='horizontal'))
plt.show()

#draw wavefront with cuts
plot_wfront(mwf, title_fig='at '+str(z1)+' m',
            isHlog=False, isVlog=False,
```

(continues on next page)

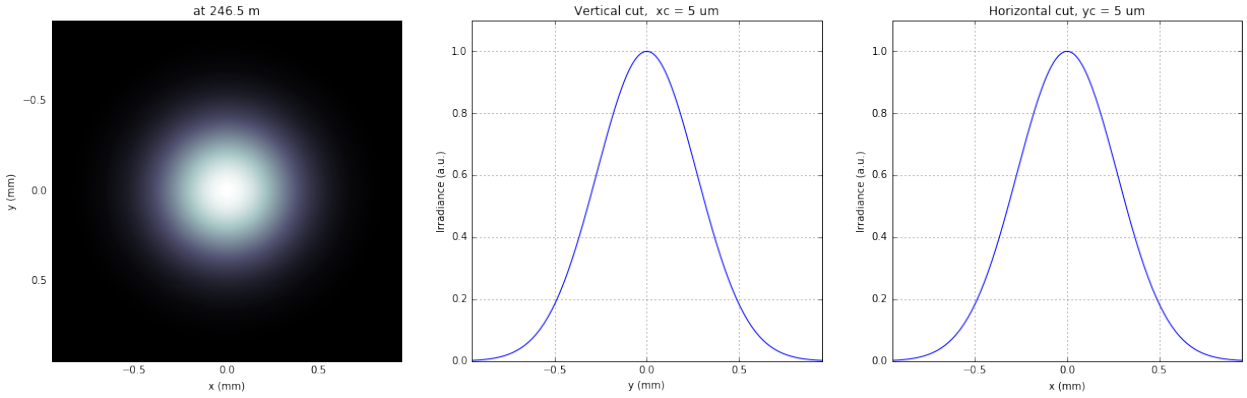
(continued from previous page)

```
            i_x_min=1e-5, i_y_min=1e-5, orient='x', onePlot=True,)

plt.set_cmap('bone') #set color map, 'bone', 'hot', 'jet', etc
fwhm_x = calculate_fwhm_x(mwf)
print('FWHMx [mm], theta_fwhm [urad]:',fwhm_x*1e3,fwhm_x/z1*1e6)
```

```
*****defining initial wavefront and writing electric field data to h5-file...
waist_fwhm [um], theta_fwhms [urad]: 26.3938279331 2.59140266508
save hdf5: g8_0kev.h5
```




```
FWHMx [mm]: 0.625880068386
FWHMy [mm]: 0.625880068386
Coordinates of center, [mm]: 0.00530406837615 0.00530406837615
stepX, stepY [um]: 10.608136752297826 10.608136752297826

R-space
FWHMx [mm], theta_fwhm [urad]: 0.625880068386 2.53906721455
```



Defining optical beamline(s)

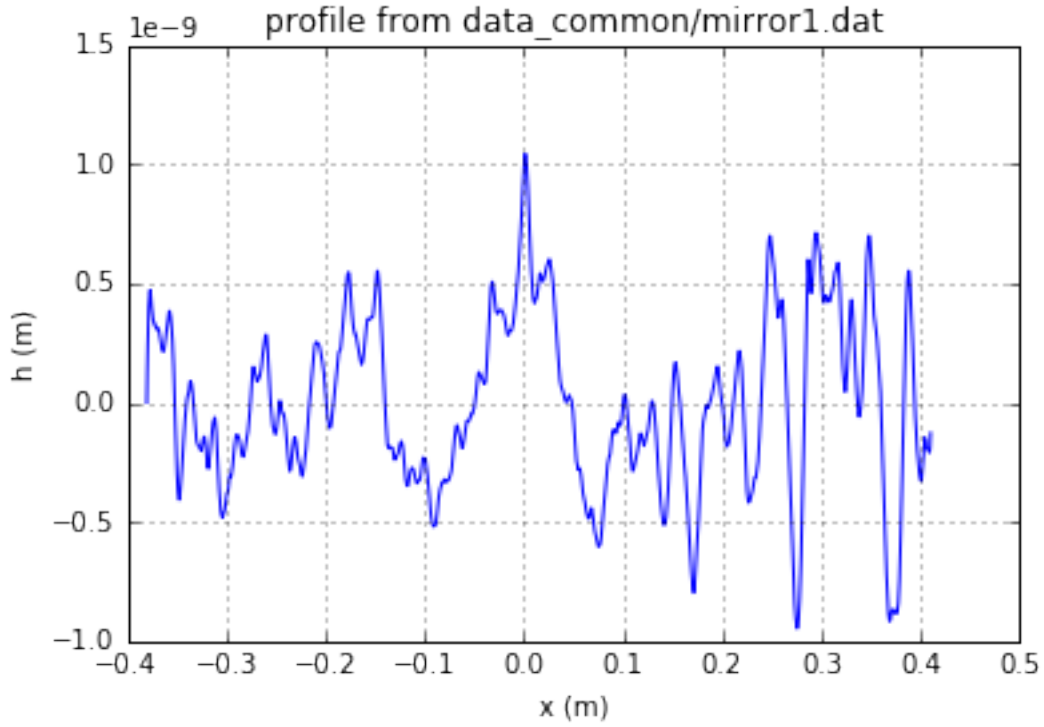
```
print('*****Defining optical beamline(s) ...')
d2exp_sase1 = 904.0
```

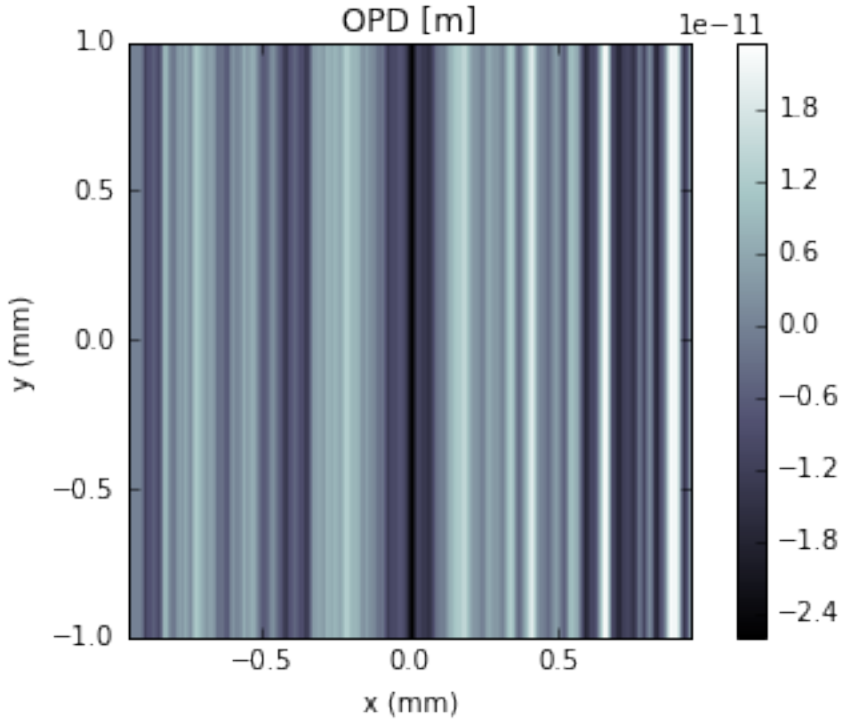
(continues on next page)

(continued from previous page)

```
d2exp_sase3 = 418.0

d2exp = d2exp_sase1
z2 = d2exp - d2m1
DriftM1_Exp = SRWLOptD(z2) #Drift from first offset mirror (M1) to exp hall
horApM1 = 0.8*thetaOM
opApM1 = SRWLOptA('r', 'a', horApM1, range_xy)  # clear aperture of the Offset␣
↪Mirror(s)

#Wavefront Propagation Parameters:
#[0]:  Auto-Resize (1) or not (0) Before propagation
#[1]:  Auto-Resize (1) or not (0) After propagation
#[2]:  Relative Precision for propagation with Auto-Resizing (1. is nominal)
#[3]:  Allow (1) or not (0) for semi-analytical treatment of quadratic phase terms at␣
↪propagation
#[4]:  Do any Resizing on Fourier side, using FFT, (1) or not (0)
#[5]:  Horizontal Range modification factor at Resizing (1. means no modification)
#[6]:  Horizontal Resolution modification factor at Resizing
#[7]:  Vertical Range modification factor at Resizing
#[8]:  Vertical Resolution modification factor at Resizing
#[9]:  Type of wavefront Shift before Resizing (not yet implemented)
#[10]: New Horizontal wavefront Center position after Shift (not yet implemented)
#[11]: New Vertical wavefront Center position after Shift (not yet implemented)
#                 [ 0] [1] [2]  [3] [4] [5]  [6]  [7]  [8]  [9] [10] [11]
ppM1 =            [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppTrErM1 =        [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDriftM1_Exp =   [ 0,  0, 1.0,  1,  0, 2.4, 1.8, 2.4, 1.8,  0,  0,   0]
ppLens =          [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDrift_Foc  =    [ 0,  0, 1.0,  1,  0, 1.0, 1.5, 1.0, 1.5,  0,  0,   0]
ppFin  =          [ 0,  0, 1.0,  0,  0, 0.05,5.0, 0.05,5.0,  0,  0,   0]

optBL0 = SRWLOptC([opApM1,  DriftM1_Exp],
                  [ppM1,ppDriftM1_Exp])

scale = 5     #5 mirror profile scaling factor
print('*****HOM1 data for BL1 beamline ')
opTrErM1 = SRWLOptT(1500, 100, horApM1, range_xy)
defineOPD(opTrErM1, os.path.join(strInputDataFolder,'mirror1.dat'), 2, '\t', 'x', ␣
↪thetaOM, scale)
opdTmp=np.array(opTrErM1.arTr)[1::2].reshape(opTrErM1.mesh.ny,opTrErM1.mesh.nx)
plt.figure()
plot_2d(opdTmp, opTrErM1.mesh.xStart*1e3,opTrErM1.mesh.xFin*1e3,opTrErM1.mesh.
↪yStart*1e3,opTrErM1.mesh.yFin*1e3,
        'OPD [m]', 'x (mm)', 'y (mm)')

optBL1 = SRWLOptC([opApM1,opTrErM1,  DriftM1_Exp],
                  [ppM1,ppTrErM1,ppDriftM1_Exp])

z3 = 30. #distance to focal plane
f_x = 1./(1./(z1+z2)+1./z3)
opLens = SRWLOptL(f_x,f_x,0,0)
Drift_Foc = SRWLOptD(z3)
optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_Exp,opLens,  Drift_Foc],
                  [ppM1,ppTrErM1,ppDriftM1_Exp,ppLens,ppDrift_Foc,ppFin])

optBL20= SRWLOptC([opApM1,  DriftM1_Exp,opLens,  Drift_Foc],
                  [ppM1,ppDriftM1_Exp,ppLens,ppDrift_Foc,ppFin])
```

```
*****Defining optical beamline(s) ...
*****HOM1 data for BL1 beamline
```






```
print_beamline(optBL1)
```

```
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.002
    Dy = 0.00189885647866
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1500
             ny = 100
             xFin = 0.001
             xStart = -0.001
             yFin = 0.000949428239331
             yStart = -0.000949428239331
             zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 657.5
    treat = 0
```

Propagating through BL0 beamline. Ideal mirror: HOM as an aperture

```
print('*****Ideal mirror: HOM as an aperture')
bPlotted = False
isHlog = False
isVlog = False
bSaved = True
optBL = optBL0
strBL = 'bl0'
pos_title = 'at exp hall wall'
print('*****setting-up optical elements, beamline:', strBL)
print_beamline(optBL)
startTime = time.time()

print('*****reading wavefront from h5 file...')
```

(continues on next page)

(continued from previous page)

```
w2 = Wavefront()
w2.load_hdf5(ifname)
wfr = w2._srwl_wf

print('*****propagating wavefront (with resizing)...')
srwl.PropagElecField(wfr, optBL)
mwf = Wavefront(wfr)
print('[nx, ny, xmin, xmax, ymin, ymax]', get_mesh(mwf))
if bSaved:
    print('save hdf5:', fname0+'_'+strBL+'.h5')
    mwf.store_hdf5(os.path.join(strOutputDataFolder, fname0+'_'+strBL+'.h5'))
print('done')
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Ideal mirror: HOM as an aperture
*****setting-up optical elements, beamline: bl0
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.002
    Dy = 0.00189885647866
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 657.5
    treat = 0


*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
[nx, ny, xmin, xmax, ymin, ymax] [780, 780, -0.004307718157368225, 0.
↪004296672726195485, -0.0043295102195176245, 0.004318408911262452]
save hdf5: g8_0kev_bl0.h5
done
propagation lasted: 0.0 min
```

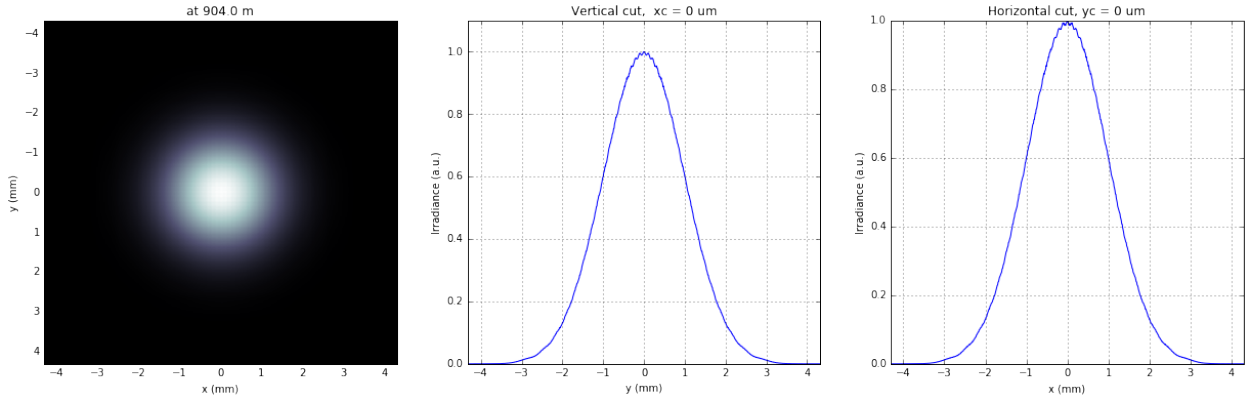
```
print('*****Ideal mirror: HOM as an aperture')
plot_wfront(mwf, 'at '+str(d2exp)+' m',False, False, 1e-5,1e-5,'x', True)
plt.set_cmap('bone') #set color map, 'bone', 'hot', 'jet', etc
plt.axis('tight')
print('FWHMx [mm], theta_fwhm [urad]:',calculate_fwhm_x(mwf)*1e3,calculate_fwhm_
↪x(mwf)/(z1+z2)*1e6)
print('FWHMy [mm], theta_fwhm [urad]:',calculate_fwhm_y(mwf)*1e3,calculate_fwhm_
↪y(mwf)/(z1+z2)*1e6)
```

```
*****Ideal mirror: HOM as an aperture
FWHMx [mm]: 2.31954054628
FWHMy [mm]: 2.33127473359
Coordinates of center, [mm]: -8.67361737988e-16 0.0
stepX, stepY [um]: 11.045431172739036 11.101308255173397

R-space
```

(continues on next page)

(continued from previous page)

```
FWHMx [mm], theta_fwhm [urad]: 2.31954054628 2.56586343615
FWHMy [mm], theta_fwhm [urad]: 2.33127473359 2.57884373184
```



Propagating through BL1 beamline. Imperfect mirror, unfocused beam

```
print('*****Imperfect mirror, unfocused beam')
bPlotted = False
isHlog = True
isVlog = False
bSaved = False
optBL = optBL1
strBL = 'bl1'
pos_title = 'at exp hall wall'
print('*****setting-up optical elements, beamline:', strBL)
print_beamline(optBL)
startTime = time.time()
print('*****reading wavefront from h5 file...')
w2 = Wavefront()
w2.load_hdf5(ifname)
wfr = w2._srwl_wf
print('*****propagating wavefront (with resizing)...')
srwl.PropagElecField(wfr, optBL)
mwf = Wavefront(wfr)
print('[nx, ny, xmin, xmax, ymin, ymax]', get_mesh(mwf))
if bSaved:
    print('save hdf5:', fname0+'_'+strBL+'.h5')
    mwf.store_hdf5(os.path.join(strOutputDataFolder,fname0+'_'+strBL+'.h5'))
print('done')
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Imperfect mirror, unfocused beam
*****setting-up optical elements, beamline: bl1
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.002
    Dy = 0.00189885647866
    ap_or_ob = a
    shape = r
    x = 0
    y = 0
```

(continues on next page)

(continued from previous page)

```
Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1500
             ny = 100
             xFin = 0.001
             xStart = -0.001
             yFin = 0.000949428239331
             yStart = -0.000949428239331
             zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 657.5
    treat = 0


*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
[nx, ny, xmin, xmax, ymin, ymax] [780, 780, -0.007224996707959405, 0.
↪007206471075374892, -0.0043295102195176245, 0.004318408911262452]
done
propagation lasted: 0.0 min
```

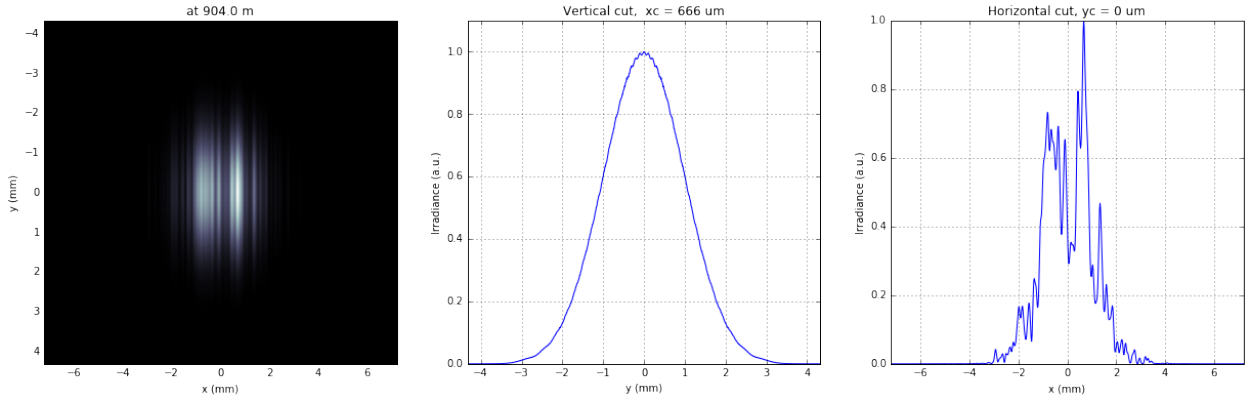
```
print ('*****Imperfect mirror, unfocused beam')
plot_wfront(mwf, 'at '+str(d2exp)+' m',False, False, 1e-5,1e-5,'x', True)
plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [mm], theta_fwhm [urad]:',calculate_fwhm_x(mwf)*1e3,calculate_fwhm_
↪x(mwf)/(z1+z2)*1e6)
print('FWHMy [mm], theta_fwhm [urad]:',calculate_fwhm_y(mwf)*1e3,calculate_fwhm_
↪y(mwf)/(z1+z2)*1e6)
```

```
*****Imperfect mirror, unfocused beam
FWHMx [mm]: 1.88961452362
FWHMy [mm]: 2.33127473359
Coordinates of center, [mm]: 0.666922773042 0.0
stepX, stepY [um]: 18.525632584511293 11.101308255173397

R-space
```

(continues on next page)

(continued from previous page)

```
FWHMx [mm], theta_fwhm [urad]: 1.88961452362 2.09028155268
FWHMy [mm], theta_fwhm [urad]: 2.33127473359 2.57884373184
```



Propagating through BL2 beamline. Focused beam: HOM aperture effect

```
print ('*****Focused beam: Focused beam: HOM aperture effect')
bPlotted = False
isHlog = True
isVlog = False
bSaved = False
optBL = optBL20
strBL = 'bl20'
pos_title = 'at sample'
print('*****setting-up optical elements, beamline:', strBL)
print_beamline(optBL)
startTime = time.time()
print('*****reading wavefront from h5 file...')
w2 = Wavefront()
w2.load_hdf5(ifname)
wfr = w2._srwl_wf
print('*****propagating wavefront (with resizing)...')
srwl.PropagElecField(wfr, optBL)
mwf = Wavefront(wfr)
print('[nx, ny, xmin, xmax, ymin, ymax]', get_mesh(mwf))
if bSaved:
    print('save hdf5:', fname0+'_'+strBL+'.h5')
    mwf.store_hdf5(os.path.join(strOutputDataFolder,fname0+'_'+strBL+'.h5'))
print('done')
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Focused beam: Focused beam: HOM aperture effect
*****setting-up optical elements, beamline: bl20
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.002
    Dy = 0.00189885647866
    ap_or_ob = a
    shape = r
    x = 0
    y = 0
```

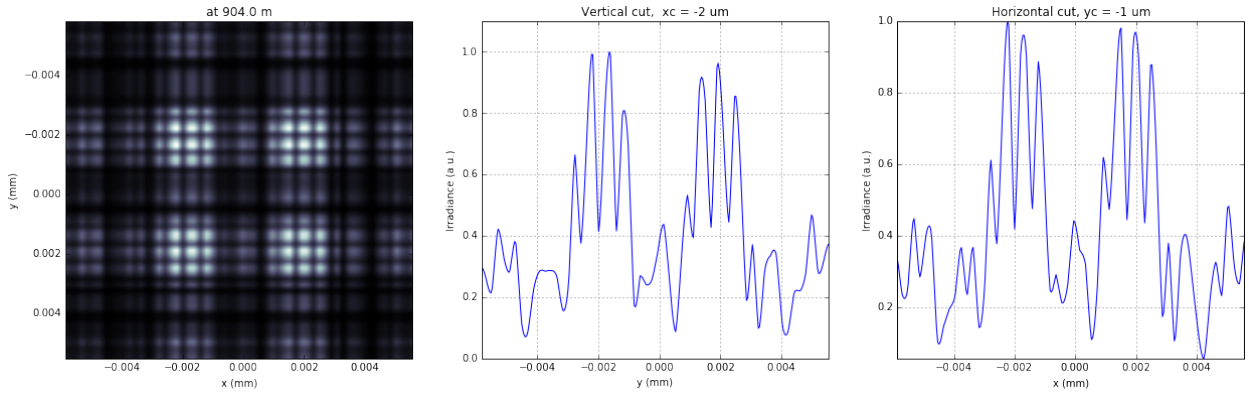
(continues on next page)

(continued from previous page)

```
Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 657.5
    treat = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 29.036402569593147
    Fy = 29.036402569593147
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.5, 1.0, 1.5, 0, 0, 0]
    L = 30.0
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 0, 0.05, 5.0, 0.05, 5.0, 0, 0, 0]


*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
[nx, ny, xmin, xmax, ymin, ymax] [294, 294, -5.863662987539073e-06, 5.
↪5701129005533396e-06, -5.832142801804413e-06, 5.540170696582709e-06]
done
propagation lasted: 0.1 min
```

```
print ('*****Focused beam: HOM aperture effect')
plot_wfront(mwf, 'at '+str(d2exp)+' m',False, False, 1e-5,1e-5,'x', True)
plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [mm], FWHMy [mm]:',calculate_fwhm_x(mwf)*1e3,calculate_fwhm_y(mwf)*1e3)
```

```
*****Focused beam: HOM aperture effect
FWHMx [mm]: 0.0055803070034
FWHMy [mm]: 0.00555031000092
Coordinates of center, [mm]: -0.00223451227903 -0.00164030028363
stepX, stepY [um]: 0.03902312589792632 0.0388133566497854

R-space
FWHMx [mm], FWHMy [mm]: 0.0055803070034 0.00555031000092
```



Propagating through BL3 beamline. Focused beam: Imperfect mirror

```
print ('*****Focused beam: Imperfect mirror')
bPlotted = False
isHlog = True
isVlog = False
bSaved = False
optBL = optBL2
strBL = 'bl2'
pos_title = 'at sample position'
print('*****setting-up optical elements, beamline:', strBL)
print_beamline(optBL)
startTime = time.time()
print('*****reading wavefront from h5 file...')
w2 = Wavefront()
w2.load_hdf5(ifname)
wfr = w2._srwl_wf
print('*****propagating wavefront (with resizing)...')
srwl.PropagElecField(wfr, optBL)
mwf = Wavefront(wfr)
print('[nx, ny, xmin, xmax, ymin, ymax]', get_mesh(mwf))
if bSaved:
    print('save hdf5:', fname0+'_'+strBL+'.h5')
    mwf.store_hdf5(os.path.join(strOutputDataFolder,fname0+'_'+strBL+'.h5'))
print('done')
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Focused beam: Imperfect mirror
*****setting-up optical elements, beamline: bl2
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.002
    Dy = 0.00189885647866
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
```

(continues on next page)

(continued from previous page)

```
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.001
            xStart = -0.001
            yFin = 0.000949428239331
            yStart = -0.000949428239331
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 657.5
    treat = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 29.036402569593147
    Fy = 29.036402569593147
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.5, 1.0, 1.5, 0, 0, 0]
    L = 30.0
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 0, 0.05, 5.0, 0.05, 5.0, 0, 0, 0]


*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
[nx, ny, xmin, xmax, ymin, ymax] [294, 294, -4.012059890591799e-06, 3.
↪811205828484517e-06, -5.832142801804413e-06, 5.540170696582709e-06]
done
propagation lasted: 0.1 min
```

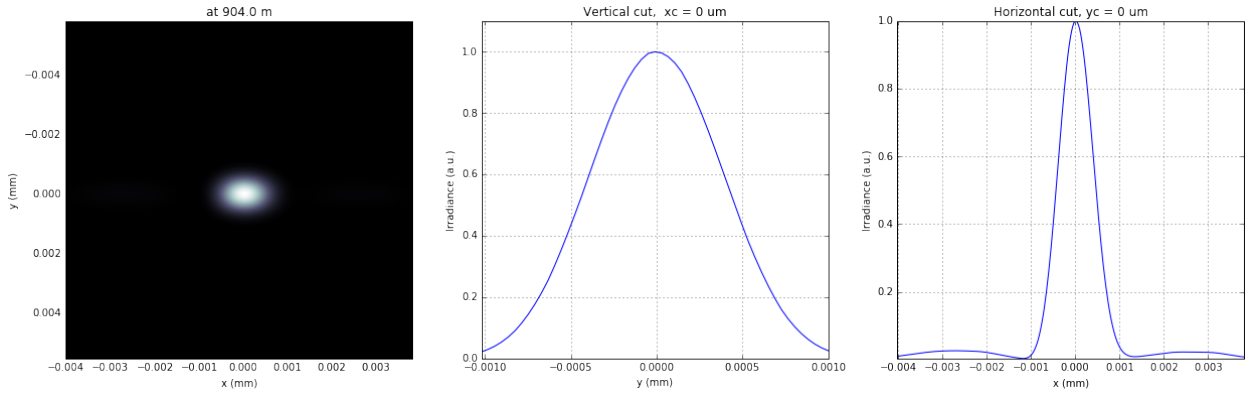
```
print('*****Focused beam: Imperfect mirror')
plot_wfront(mwf, 'at '+str(d2exp)+' m',False, False, 1e-5,1e-5,'x', True)
plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
```

(continues on next page)

(continued from previous page)

```
plt.axis('tight')
print('FWHMx [mm], FWHMy [mm]:',calculate_fwhm_x(mwf)*1e3,calculate_fwhm_y(mwf)*1e3)
#int_x=mwf.get_intensity(slice=0); np.savetxt('bl2.txt',int_x[int_x.shape[1]/2,:])
```

```
*****Focused beam: Imperfect mirror
FWHMx [mm]: 0.000881118664606
FWHMy [mm]: 0.000892707202945
Coordinates of center, [mm]: 1.97255141199e-05 -1.01393043366e-05
stepX, stepY [um]: 0.026700565594117 12 0.0388133566497854

R-space
FWHMx [mm], FWHMy [mm]: 0.000881118664606 0.000892707202945
```



Propagating through BL4 beamline. Focused beam, out of focus

```
print('*****Focused beam, out of focus')
dz = 15.e-3
n = 5
startTime = time.time()
for idx in range(-n,n):
    Drift = SRWLOptD(z3+dz*idx/n)
    optBL = SRWLOptC([opApM1,opTrErM1,  DriftM1_Exp,opLens,  Drift],
                     [ppM1,ppTrErM1,ppDriftM1_Exp,ppLens,ppDrift_Foc,ppFin])
    #print_beamline(optBL4)
    strBL = 'bl'+'_'+str(idx+n)
    print('*****reading wavefront from h5 file...')
    w2 = Wavefront()
    w2.load_hdf5(ifname)
    wfr = w2._srwl_wf
    print('*****propagating wavefront (with resizing)...')
    srwl.PropagElecField(wfr, optBL)
    mwf = Wavefront(wfr)
    print('save hdf5:', fname0+'_'+strBL+'.h5')
    mwf.store_hdf5(os.path.join(strOutputDataFolder,fname0+'_'+strBL+'.h5'))
```

```
*****Focused beam, out of focus
*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
save hdf5: g8_0kev_bl_0.h5
```

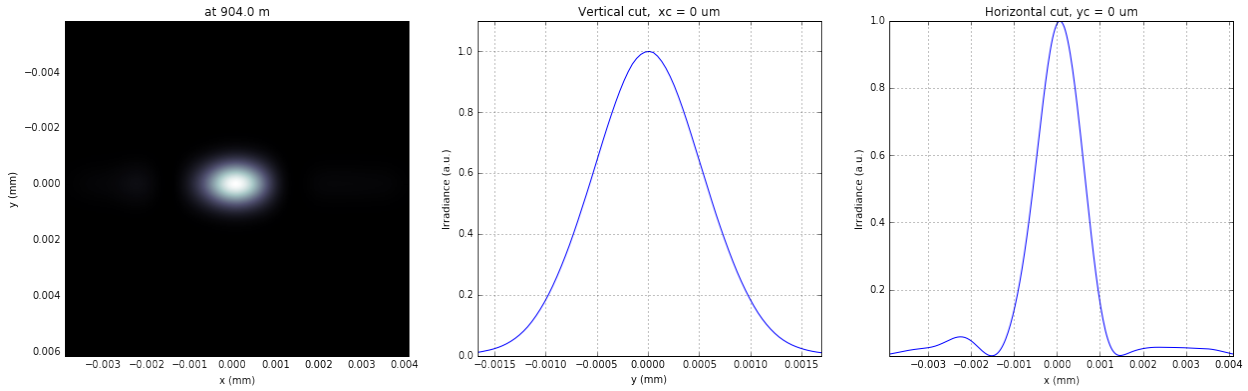
(continues on next page)

(continued from previous page)

```
*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
save hdf5: g8_0kev_bl_1.h5
*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
save hdf5: g8_0kev_bl_2.h5
*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
save hdf5: g8_0kev_bl_3.h5
*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
save hdf5: g8_0kev_bl_4.h5
*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
save hdf5: g8_0kev_bl_5.h5
*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
save hdf5: g8_0kev_bl_6.h5
*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
save hdf5: g8_0kev_bl_7.h5
*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
save hdf5: g8_0kev_bl_8.h5
*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
save hdf5: g8_0kev_bl_9.h5
```

```
print('*****Focused beam, out of focus, last slice')
plot_wfront(mwf, 'at '+str(d2exp)+' m',False, False, 1e-5,1e-5,'x', True)
plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [um], FWHMy [um]:',calculate_fwhm_x(mwf)*1e6,calculate_fwhm_y(mwf)*1e6)
#int_x=mwf.get_intensity(slice=0); np.savetxt('bl2.txt',int_x[int_x.shape[1]/2,:])
```

```
*****Focused beam, out of focus, last slice
FWHMx [mm]: 0.00120037884559
FWHMy [mm]: 0.00122854539774
Coordinates of center, [mm]: 6.16894426568e-05 1.06978600103e-05
stepX, stepY [um]: 0.027281337399753247 0.04095151325811603

R-space
FWHMx [um], FWHMy [um]: 1.20037884559 1.22854539774
```



1.10.2 Wavefront propagation simulation tutorial - Case 1

L.Samoylova liubov.samoylova@xfel.eu, A.Buzmakov buzmakov@gmail.com

Tutorial course on Wavefront Propagation Simulations, 28/11/2013, European XFEL, Hamburg. Updated for using new syntax 28/11/2015

Wave optics software is based on SRW core library https://github.com/ochubar/SRW, available through WPG interactive framework https://github.com/samoylv/WPG

Propagation through CRLs optics

Import modules

```
%matplotlib inline
```

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

#Importing necessary modules:
import os
import sys
import copy

sys.path.insert(0,os.path.join('..','..'))
#sys.path.insert('../..')

import time
import numpy as np
import pylab as plt

if sys.version_info[0] ==3:
    import pickle
else:
    import cPickle as pickle
import errno

#import SRW core functions
```

(continues on next page)

(continued from previous page)

```
from wpg.srwlib import srwl, srwl_opt_setup_CRL, SRWLOptD, SRWLOptA, SRWLOptC,␣
↪SRWLOptT

#import SRW auxiliary functions
from wpg.useful_code.srwutils import AuxTransmAddSurfHeightProfileScaled

#import some helpers functions
from wpg.useful_code.wfrutils import calculate_fwhm_x, plot_wfront, calculate_fwhm_y,␣
↪print_beamline, get_mesh, plot_1d, plot_2d

from wpg.wpg_uti_wf import propagate_wavefront
from wpg.wpg_uti_oe import show_transmission

#from wpg.useful_code.wfrutils import propagate_wavefront

#Import base wavefront class
from wpg import Wavefront

#Import base beamline class and OE wrappers
from wpg.beamline import Beamline
from wpg.optical_elements import Empty, Use_PP
from wpg.optical_elements import         Drift,Aperture,    Lens,Mirror_elliptical,WF_
↪dist,calculateOPD

#Gaussian beam generator
from wpg.generators import build_gauss_wavefront_xy

plt.ion()
```

Use or not new syntax

```
NEW_SYNTAX = True
```

Define auxiliary functions

```
def calculate_source_fwhm(ekev, theta_fwhm):
    """
    Calculate source size from photon energy and FWHM angular divergence

    :param evev: Energy in keV
    :param theta_fwhm: theta_fwhm [units?]
    """
    wl = 12.39e-10/ekev
    k = 2 * np.sqrt(2*np.log(2))
    theta_sigma = theta_fwhm /k
    sigma0 = wl /(2*np.pi*theta_sigma)
    return sigma0*k

def calculate_theta_fwhm_cdr(ekev,qnC):
    """
    Calculate angular divergence using formula from XFEL CDR2011
```

(continues on next page)

(continued from previous page)

```
    :param ekev: Energy in keV
    :param qnC: e-bunch charge, [nC]
    :return: theta_fwhm [units?]
    """
    theta_fwhm = (17.2 - 6.4 * np.sqrt(qnC))*1e-6/ekev**0.85
    return theta_fwhm

def defineOPD(opTrErMirr, mdatafile, ncol, delim, Orient, theta, scale):
    """
    Define optical path difference (OPD) from mirror profile, i.e. ill the struct␣
↪opTrErMirr

    :params mdatafile: an ascii file with mirror profile data
    :params ncol: number of columns in the file
    :params delim: delimiter between numbers in an row, can be space (' '), tab '\t',␣
↪etc
    :params orient: mirror orientation, 'x' (horizontal) or 'y' (vertical)
    :params theta: incidence angle
    :params scale: scaling factor for the mirror profile
    """
    heightProfData = np.loadtxt(mdatafile).T
    AuxTransmAddSurfHeightProfileScaled(opTrErMirr, heightProfData, Orient, theta,␣
↪scale)
    plt.figure()
    plot_1d(heightProfData,'profile from ' + mdatafile,'x (m)', 'h (m)')
```

```
def calc_sampling(zoom,mf):
    """
    This function calculates sampling.
    :param zoom: range zoom
    :param mf: modification factor for step, i.e. dx1=mf*dx0

    :return: sampling.
    """
    sampling = zoom/mf;
    print('zoom:{:.1f}; mod_factor:{:.1f}; sampling:{:.1f}'.format(zoom, mf,␣
↪sampling))
    return sampling
```

```
def _save_object(obj, file_name):
    """
    Save any python object to file.

    :param: obj : - python objest to be saved
    :param: file_name : - output file, wil be overwrite if exists
    """
    with open(file_name,'wb') as f:
        pickle.dump(obj, f)

def _load_object(file_name):
    """
    Save any python object to file.

    :param: file_name : - output file, wil be overwrite if exists
    :return: obj : - loaded pthon object
    """
```

(continues on next page)

(continued from previous page)

```
    res = None
    with open(file_name,'rb') as f:
        res = pickle.load(f)

    return res

def mkdir_p(path):
    """
    Create directory with subfolders (like Linux mkdir -p)

    :param path: Path to be created
    """
    try:
        os.makedirs(path)
    except OSError as exc: # Python >2.5
        if exc.errno == errno.EEXIST and os.path.isdir(path):
            pass
        else: raise

def create_CRL(directory=None, voids_params=None, *args, **keywrds):
    """
    This function build CLR or load it from file if it was created beforehand.
    Out/input filename builded as sequence of function parameters.

    Adiitinal parameters (*args) passed to srwlib.srwl_opt_setup_CRL function

    :param directory: output directory to save file.
    :param voids_params: void params to build CRL and construct unique file name
    :return: SRWL CRL object
    """
    if not isinstance(voids_params,tuple):
        raise TypeError('Voids_params must be tuple')

    file_name = '_'.join([str(a) for a in args[:-1]])
    subdir_name = '_'.join([str(v) for v in voids_params])
    if directory is None:
        full_path = os.path.join(subdir_name,file_name+'.pkl')
    else:
        full_path = os.path.join(directory, subdir_name, file_name+'.pkl')

    if  os.path.isfile(full_path):
        print('Found file {}. CLR will be loaded from file'.format(full_path))
        res = _load_object(full_path)
        return res
    else:
        print('CLR file NOT found. CLR will be recalculated and saved in file {}'.
↪format(full_path))
        res = srwlib.srwl_opt_setup_CRL(*args)
        mkdir_p(os.path.dirname(full_path))
        _save_object(res, full_path)
        return res

def create_CRL1(directory,file_name,*args, **keywrds):
    """
    This function build CLR or load it from file.
    Out/input filename builded as sequence of function parameters.
    Adiitinal parameters (*args) passed to srwlib.srwl_opt_setup_CRL function
```

(continues on next page)

(continued from previous page)

```
    :param directory: output directory
    :param fiel_name: CRL file name
    :return: SRWL CRL object
    """

    full_path = os.path.join(directory, file_name+'.pkl')

    if  os.path.isfile(full_path):
        print('Found file {}. CLR will be loaded from file'.format(full_path))
        res = _load_object(full_path)
        return res
    else:
        print('CLR file NOT found. CLR will be recalculated and saved in file {}'.
↪format(full_path))
        res = srwl_opt_setup_CRL(*args)
        mkdir_p(os.path.dirname(full_path))
        _save_object(res, full_path)
        return res
```

Defining initial wavefront and writing electric field data to h5-file

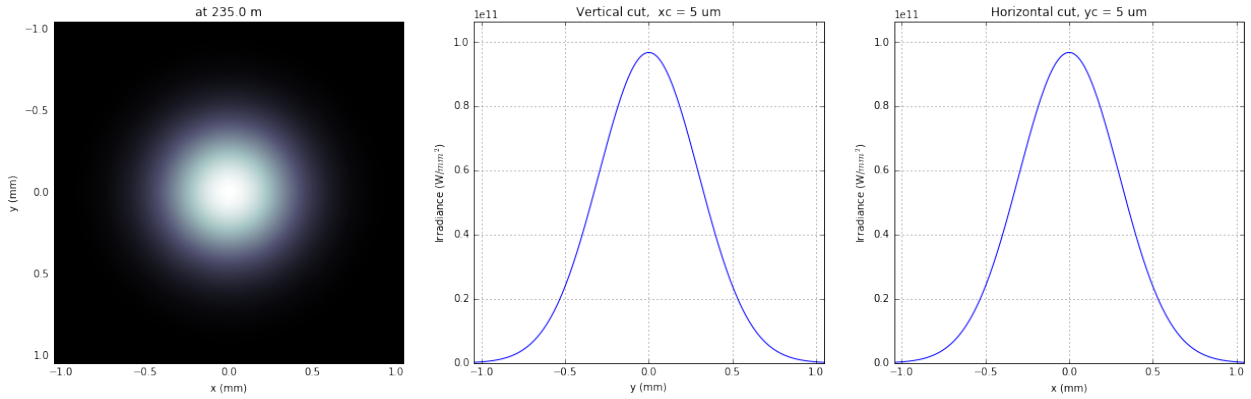
```
print('*****defining initial wavefront and writing electric field data to h5-file...')

strInputDataFolder ='data_common' # sub-folder name for common input  data
strDataFolderName = 'Tutorial_case_1' # output data sub-folder name
if not os.path.exists(strDataFolderName):
    mkdir_p(strDataFolderName)

d2crl1_sase1 = 235.0 # Distance to CRL1 on SASE1 [m]
d2crl1_sase2 = 235.0 # Distance to CRL1 on SASE2 [m]
d2m1_sase1 = 246.5  # Distance to mirror1 on SASE1 [m]
d2m1_sase2 = 290.0  # Distance to mirror1 on SASE2 [m]

ekev = 6.742 # Energy [keV]
thetaOM = 2.5e-3       # @check!

# e-bunch charge, [nC]; total pulse energy, J
#qnC = 0.02;pulse_duration = 1.7e-15;pulseEnergy = 0.08e-3
#coh_time = 0.24e-15

qnC = 0.1; # e-bunch charge, [nC]
pulse_duration = 9.e-15;
pulseEnergy = 0.5e-3; # total pulse energy, J
coh_time = 0.24e-15


d2m1 = d2m1_sase2
d2crl1 = d2crl1_sase2

z1 = d2crl1
theta_fwhm = calculate_theta_fwhm_cdr(ekev,qnC)
k = 2*np.sqrt(2*np.log(2))
sigX = 12.4e-10*k/(ekev*4*np.pi*theta_fwhm)
```

(continues on next page)

(continued from previous page)

```
print('sigX, waist_fwhm [um], far field theta_fwhms [urad]: {}, {},{}'.format(
                            sigX*1e6, sigX*k*1e6, theta_fwhm*1e6)
      )
#define limits
range_xy = theta_fwhm/k*z1*7. # sigma*7 beam size
npoints=180

wfr0 = build_gauss_wavefront_xy(npoints, npoints, ekev, -range_xy/2, range_xy/2,
                                -range_xy/2, range_xy/2 ,sigX, sigX, z1,
                                pulseEn=pulseEnergy, pulseTau=coh_time/np.sqrt(2),
                                repRate=1/(np.sqrt(2)*pulse_duration))

mwf = Wavefront(wfr0)
ip = np.floor(ekev)
frac = np.floor((ekev - ip)*1e3)
ename = str(int(ip))+'_'+str(int(frac))+'kev'
fname0 = 'g' + ename
ifname = os.path.join(strDataFolderName,fname0+'.h5')
print('save hdf5: '+fname0+'.h5')
mwf.store_hdf5(ifname)
print('done')
pow_x=plot_wfront(mwf, 'at '+str(z1)+' m',False, False, 1e-5,1e-5,'x', True, saveDir=
↪'./'+strDataFolderName)
plt.set_cmap('bone') #set color map, 'bone', 'hot', 'jet', etc
fwhm_x = calculate_fwhm_x(mwf);fwhm_y = calculate_fwhm_y(mwf)
print('FWHMx [mm], theta_fwhm=fwhm_x/z1 [urad], distance to waist: {}, {}'.format(
        fwhm_x*1e3,fwhm_x/z1*1e6)
      )
```

```
*****defining initial wavefront and writing electric field data to h5-file...
sigX, waist_fwhm [um], far field theta_fwhms [urad]: 11.499788231945866, 27.
↪079931842197144,2.9970290603902483
save hdf5: g6_742kev.h5
done
FWHMx [mm]: 0.69007789015
FWHMy [mm]: 0.69007789015
Coordinates of center, [mm]: 0.00584811771314 0.00584811771314
stepX, stepY [um]: 11.696235426275774 11.696235426275774

Total power (integrated over full range): 54.4369 [GW]
Peak power calculated using FWHM:         52.4365 [GW]
Max irradiance: 96.7817 [GW/mm^2]
R-space
FWHMx [mm], theta_fwhm=fwhm_x/z1 [urad], distance to waist: 0.6900778901502707, 2.
↪9365016602139176
```




```
print(pow_x[:,1].max())
print ('I_o {} [GW/mm^2]'.format((pow_x[:,1].max()*1e-9)))
print ('peak power {} [GW]'.format((pow_x[:,1].max()*1e-9*1e6*2*np.pi*(fwhm_x/2.
↪35)**2)))
```

```
96781656064.0
I_o 96.781656064 [GW/mm^2]
peak power 52.436466558883836 [GW]
```

Defining optical beamline(s)

```
print('*****Defining optical beamline(s) ...')
#***********CRLs
nCRL1 = 1 #number of lenses, collimating
nCRL2 = 8
delta = 7.511e-06
attenLen = 3.88E-3
diamCRL = 3.58e-03 #CRL diameter
#rMinCRL = 3.3e-03  #CRL radius at the tip of parabola [m]
rMinCRL = 2*delta*z1/nCRL1
wallThickCRL = 30e-06 #CRL wall thickness [m]

#Generating a perfect 2D parabolic CRL:
#opCRL1 = srwlib.srwl_opt_setup_CRL(3, delta, attenLen, 1,
#                                   diamCRL, diamCRL, rMinCRL, nCRL, wallThickCRL, 0,␣
↪0)
opCRL1 = create_CRL1(strDataFolderName,
                     'opd_CRL1_'+str(nCRL1)+'_R'+str(int(rMinCRL*1e6))+'_'+ename,
                     3,delta,attenLen,1,diamCRL,diamCRL,rMinCRL,nCRL1,wallThickCRL,0,
↪0,None)
#opCRL1 = srwl_opt_setup_CRL(3, delta, attenLen, 1,
#                                   diamCRL, diamCRL, rMinCRL, nCRL1, wallThickCRL, 0,␣
↪0)
#Saving transmission data to file
#AuxSaveOpTransmData(opCRL1, 3, os.path.join(os.getcwd(), strDataFolderName, "opt_
↪path_dif_CRL1.dat"))
opCRL2 = create_CRL1(strDataFolderName,
                     'opd_CRL2_'+str(nCRL2)+'_R'+str(int(rMinCRL*1e6))+'_'+ename,
                     3,delta,attenLen,1,diamCRL,diamCRL,rMinCRL,nCRL2,wallThickCRL,0,
↪0,None)
```

(continues on next page)

(continued from previous page)

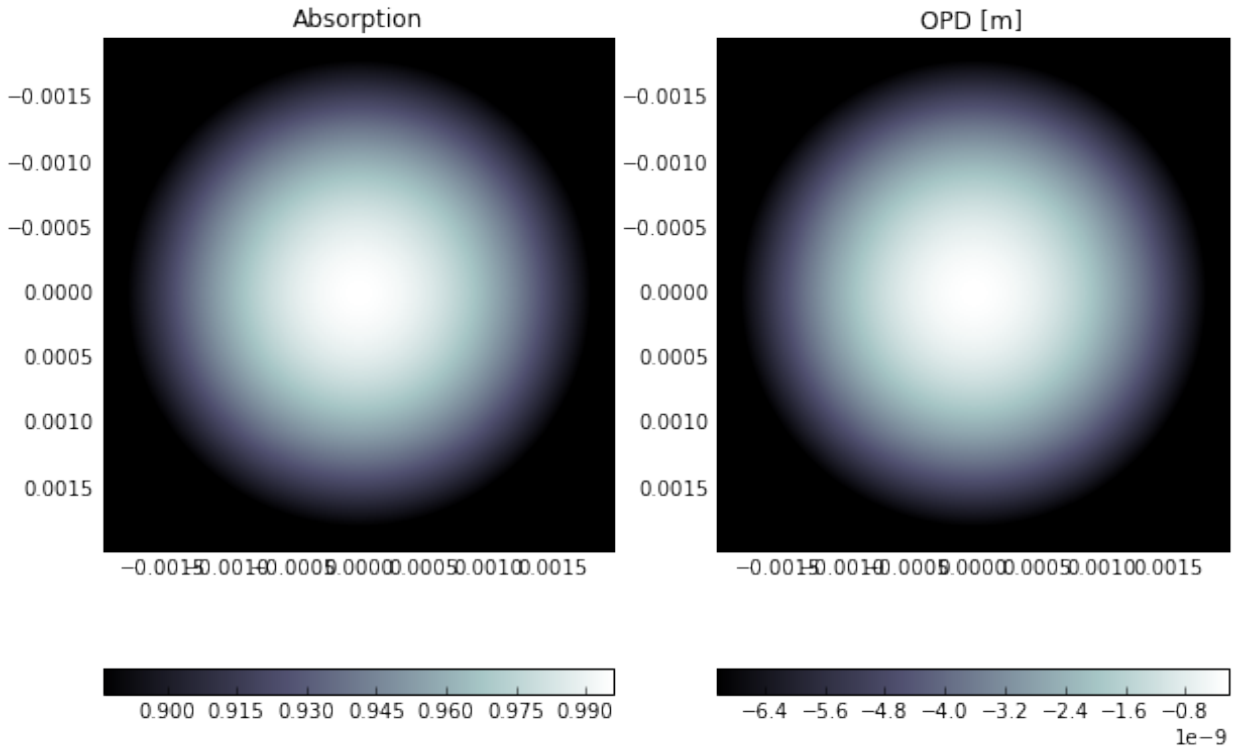
```
scale = 1      #5 mirror profile scaling factor
horApM1 = 0.8*thetaOM

#d2crl2_sase1 = 904.0
d2crl2_sase2 = 931.0

d2exp_sase1 = 904.0
d2exp_sase2 = 942.0

d2crl2 = d2crl2_sase2
d2exp = d2exp_sase2
z2 = d2m1 - d2crl1
z3 = d2crl2 - d2m1
#z3 = d2exp - d2m1
z4 = rMinCRL/(2*delta*nCRL2)

if not NEW_SYNTAX:
    opApCRL1 = SRWLOptA('c','a',range_xy,range_xy)  # circular collimating CRL(s) aperture
    opApM1 = SRWLOptA('r', 'a', horApM1, range_xy)  # clear aperture of the Offset Mirror(s)
    DriftCRL1_M1 = SRWLOptD(z2) #Drift from CRL1 to the first offset mirror (M1)
    DriftM1_Exp  = SRWLOptD(z3) #Drift from M1 to exp hall
    Drift_Sample  = SRWLOptD(z4) #Drift from focusing CRL2 to focal plane

#Wavefront Propagation Parameters:
#[0]:  Auto-Resize (1) or not (0) Before propagation
#[1]:  Auto-Resize (1) or not (0) After propagation
#[2]:  Relative Precision for propagation with Auto-Resizing (1. is nominal)
#[3]:  Allow (1) or not (0) for semi-analytical treatment of quadratic phase terms at propagation
#[4]:  Do any Resizing on Fourier side, using FFT, (1) or not (0)
#[5]:  Horizontal Range modification factor at Resizing (1. means no modification)
#[6]:  Horizontal Resolution modification factor at Resizing
#[7]:  Vertical Range modification factor at Resizing
#[8]:  Vertical Resolution modification factor at Resizing
#[9]:  Type of wavefront Shift before Resizing (not yet implemented)
#[10]: New Horizontal wavefront Center position after Shift (not yet implemented)
#[11]: New Vertical wavefront Center position after Shift (not yet implemented)
#                     [ 0] [1] [2]  [3] [4] [5]  [6]  [7]  [8]  [9] [10] [11]
    ppCRL1 =          [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
    ppDriftCRL1_M1 =  [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
    ppM1 =            [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
    ppDriftM1_Exp  =  [ 0,  0, 1.0,  1,  0, 2.4, 1.8, 2.4, 1.8,  0,  0,   0]
    ppTrErM1 =        [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
    ppCRL2 =          [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
    ppDrift_Sample  = [ 0,  0, 1.0,  1,  0, 1.8, 1.5, 1.8, 1.5,  0,  0,   0]
    ppFin  =          [ 0,  0, 1.0,  0,  0, 0.01, 5.0, 0.01, 5.0,  0,  0,   0]

    optBL0 = SRWLOptC([opCRL1,  DriftCRL1_M1,opApM1,  DriftM1_Exp],
                  [ppCRL1,ppDriftCRL1_M1,  ppM1,ppDriftM1_Exp])

    print('*****HOM1 data for BL1 beamline ')
    opTrErM1 = SRWLOptT(1500, 100, horApM1, range_xy)
    #defineOPD(opTrErM1, os.path.join(strInputDataFolder,'mirror1.dat'), 2, '\t', 'x',
    thetaOM, scale)
    defineOPD(opTrErM1, os.path.join(strInputDataFolder,'mirror2.dat'), 2, ' ', 'x', thetaOM, scale)
```

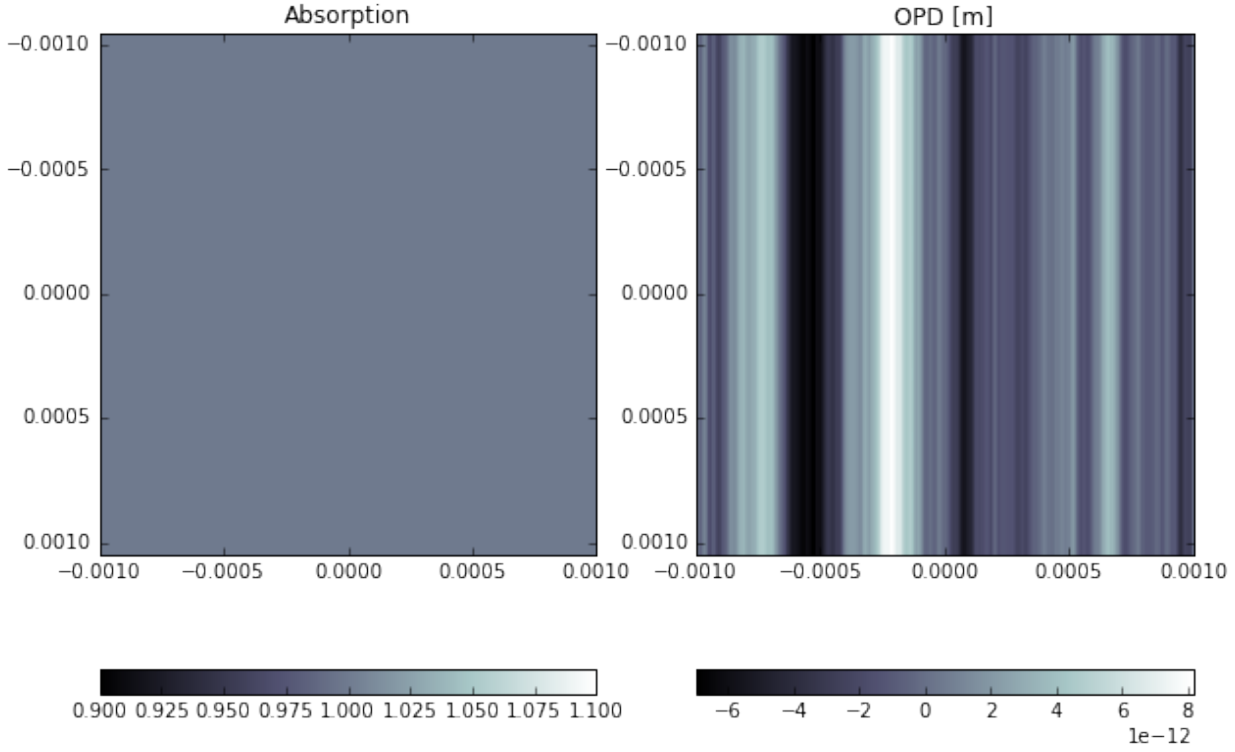
(continues on next page)

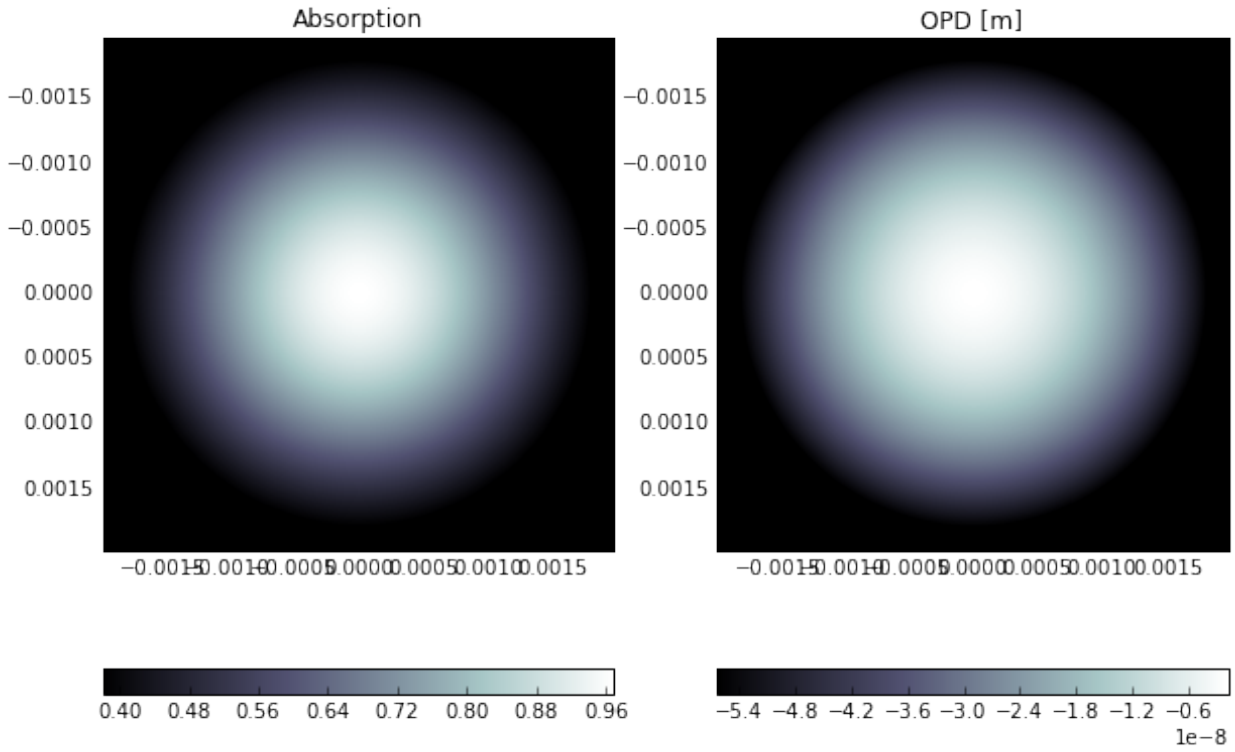
(continued from previous page)

```
    opdTmp=np.array(opTrErM1.arTr)[1::2].reshape(opTrErM1.mesh.ny,opTrErM1.mesh.nx)
    plt.figure()
    plot_2d(opdTmp, opTrErM1.mesh.xStart*1e3,opTrErM1.mesh.xFin*1e3,
            opTrErM1.mesh.yStart*1e3,opTrErM1.mesh.yFin*1e3,'OPD [m]', 'x (mm)', 'y␣
↪(mm)')

    optBL1 = SRWLOptC([opCRL1,  DriftCRL1_M1,opApM1,opTrErM1,  DriftM1_Exp],
                      [ppCRL1,ppDriftCRL1_M1,  ppM1,ppTrErM1,ppDriftM1_Exp])

    optBL2 = SRWLOptC([opCRL1,  DriftCRL1_M1,opApM1,opTrErM1,  DriftM1_Exp, opCRL2,
↪Drift_Sample],
                      [ppCRL1,ppDriftCRL1_M1,  ppM1,ppTrErM1,ppDriftM1_Exp, ppCRL2,␣
↪ppDrift_Sample,ppFin])
else:
    optBL0 = Beamline()
    #optBL0.append(Aperture(shape='c',ap_or_ob='a',Dx=range_xy), Use_PP())# circular␣
↪CRL aperture
    optBL0.append(opCRL1,    Use_PP())
    optBL0.append(Drift(z2), Use_PP(semi_analytical_treatment=1))
    optBL0.append(Aperture(shape='r',ap_or_ob='a',Dx=horApM1,Dy=range_xy),
                           Use_PP())
    optBL0.append(Drift(z3), Use_PP(semi_analytical_treatment=1, zoom=2.4, sampling=1.
↪8))

    show_transmission(opCRL1)
    opOPD_M1 = calculateOPD(WF_dist(nx=1500,ny=100,Dx=horApM1,Dy=range_xy),
                            os.path.join(strInputDataFolder,'mirror2.dat'),
                            2, ' ', 'x',  thetaOM, scale)
    show_transmission(opOPD_M1)
    optBL1 = Beamline()
    #optBL1.append(Aperture(shape='c',ap_or_ob='a',Dx=range_xy), Use_PP())# circular␣
↪CRL aperture
    optBL1.append(opCRL1,    Use_PP())
    optBL1.append(Drift(z2), Use_PP(semi_analytical_treatment=1))
    optBL1.append(Aperture(shape='r',ap_or_ob='a',Dx=horApM1,Dy=range_xy),
                           Use_PP())
    optBL1.append(Aperture(shape='r',ap_or_ob='a',Dx=horApM1,Dy=range_xy),
                  Use_PP())
    optBL1.append(opOPD_M1,Use_PP())
    optBL1.append(Drift(z3),
                  Use_PP(semi_analytical_treatment=1, zoom=2.4, sampling=1.8))

    show_transmission(opCRL2)
    optBL2 = copy.deepcopy(optBL1)
    optBL2.append(opCRL2,     Use_PP())
    optBL2.append(Drift(z4),  Use_PP(semi_analytical_treatment=1, zoom=1.5,␣
↪sampling=1.8))
    zoom=0.02; optBL2.append(Empty(),
                             Use_PP(fft_resizing=1,zoom=zoom, sampling=calc_
↪sampling(zoom=zoom,mf=0.01)))
```

```
*****Defining optical beamline(s) ...
Found file Tutorial_case_1/opd_CRL1_1_R3530_6_742kev.pkl. CLR will be loaded from file
Found file Tutorial_case_1/opd_CRL2_8_R3530_6_742kev.pkl. CLR will be loaded from file
zoom:0.0; mod_factor:0.0; sampling:2.0
```


Absorption
OPD [m]
−0.0015
−0.0010
−0.0005
0.0000
0.0005
0.0010
0.0015
0.900 0.915 0.930 0.945 0.960 0.975 0.990
−6.4 −5.6 −4.8 −4.0 −3.2 −2.4 −1.6 −0.8
1e−9


Absorption
OPD [m]
−0.0010
−0.0005
0.0000
0.0005
0.0010
0.900 0.925 0.950 0.975 1.000 1.025 1.050 1.075 1.100
−6 −4 −2 0 2 4 6 8
1e−12



Propagating through BL0 beamline. Collimating CRL and ideal mirror

```
print('*****Collimating CRL and ideal mirror')
bPlotted = False
isHlog = False
isVlog = False
bSaved = True
optBL = optBL0
strBL = 'bl0'
pos_title = 'at exp hall wall'
print('*****setting-up optical elements, beamline:'+ strBL)

if not NEW_SYNTAX:
    bl = Beamline(optBL)
else:
    bl = optBL
print(bl)

if bSaved:
    out_file_name = os.path.join(strDataFolderName, fname0+'_'+strBL+'.h5')
    print('save hdf5:'+ out_file_name)
else:
    out_file_name = None

startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted: {} min'.format(round((time.time() - startTime) / 6.) / 10.
↪))
```

```
*****Collimating CRL and ideal mirror
*****setting-up optical elements, beamline:bl0
Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 235.0
    Fy = 235.0
    arTr = array of size 2004002
    extTr = 1
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1001
            ny = 1001
            xFin = 0.0019690000000000003
            xStart = -0.0019690000000000003
            yFin = 0.0019690000000000003
            yStart = -0.0019690000000000003
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 55.0
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.002
    Dy = 0.0020936261413
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 641.0
    treat = 0


save hdf5:Tutorial_case_1/g6_742kev_bl0.h5
Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 235.0
    Fy = 235.0
    arTr = array of size 2004002
    extTr = 1
    mesh = Radiation Mesh (Sampling)
            arSurf = None
```

(continues on next page)

(continued from previous page)

```
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1001
            ny = 1001
            xFin = 0.0019690000000000003
            xStart = -0.0019690000000000003
            yFin = 0.0019690000000000003
            yStart = -0.0019690000000000003
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 55.0
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.002
    Dy = 0.0020936261413
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 641.0
    treat = 0


*****reading wavefront from h5 file...
R-space
nx   180  range_x [-1.0e+00, 1.0e+00] mm
ny   180  range_y [-1.0e+00, 1.0e+00] mm
*****propagating wavefront (with resizing)...
save hdf5: Tutorial_case_1/g6_742kev_bl0.h5
done
propagation lasted: 0.0 min
```

```
# bl.propagation_options[0]['optical_elements']
```

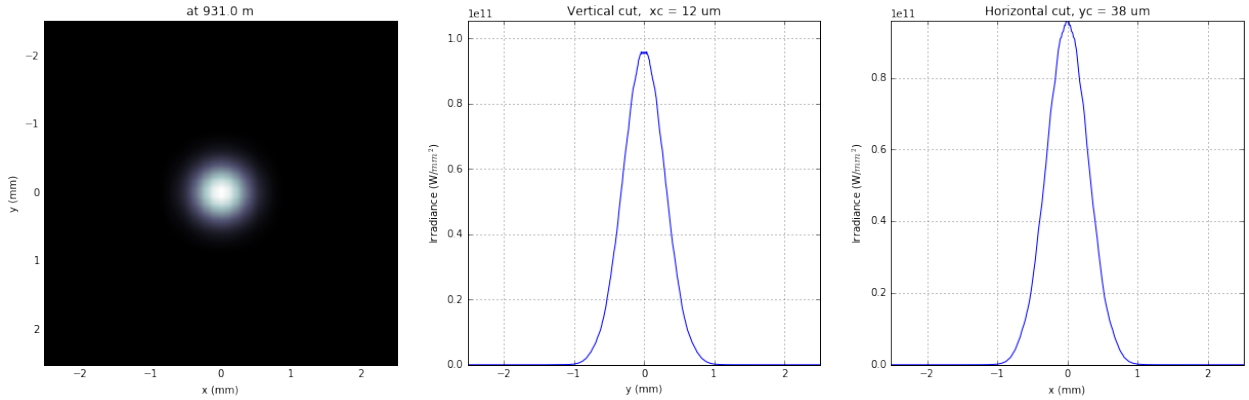
```
print('*****Collimating CRL and ideal mirror')
plot_wfront(mwf, 'at '+str(z1+z2+z3)+' m',False, False, 1e-4,1e-7,'x', True, saveDir=
↪'./'+strDataFolderName)
plt.set_cmap('bone') #set color map, 'bone', 'hot', 'jet', etc
plt.axis('tight')
print('FWHMx [mm], theta_fwhm [urad]: {}, {}'.format(calculate_fwhm_x(mwf)*1e3,␣
↪calculate_fwhm_x(mwf)/(z1+z2)*1e6))
```

(continues on next page)

(continued from previous page)

```
print('FWHMy [mm], theta_fwhm [urad]: {}, {}'.format(calculate_fwhm_y(mwf)*1e3,␣
↪calculate_fwhm_y(mwf)/(z1+z2)*1e6))
```

```
*****Collimating CRL and ideal mirror
FWHMx [mm]: 0.684951762278
FWHMy [mm]: 0.697875380434
Coordinates of center, [mm]: 0.0129236181562 0.0387708544686
stepX, stepY [um]: 6.461809078096801 6.461809078096801

Total power (integrated over full range): 53.3003 [GW]
Peak power calculated using FWHM:         52.1573 [GW]
Max irradiance: 95.9032 [GW/mm^2]
R-space
FWHMx [mm], theta_fwhm [urad]: 0.6849517622782609, 2.3619026285457276
FWHMy [mm], theta_fwhm [urad]: 0.6978753804344545, 2.406466829084326
```



Propagating through BL1 beamline. Collimating CRL and imperfect mirror

```
print ('*****Collimating CRL and imperfect mirror')
bPlotted = False
isHlog = True
isVlog = False
bSaved = False
optBL = optBL1
strBL = 'bl1'
pos_title = 'at exp hall wall'
print('*****setting-up optical elements, beamline:' + strBL)

if not NEW_SYNTAX:
    bl = Beamline(optBL)
else:
    bl = optBL
print(bl)

if bSaved:
    out_file_name = os.path.join(strDataFolderName, fname0+'_'+strBL+'.h5')
    print('save hdf5: '+ out_file_name)
else:
    out_file_name = None
```

(continues on next page)

(continued from previous page)

```
startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted: {} min'.format(round((time.time() - startTime) / 6.) / 10.
↪))
```

```
*****Collimating CRL and imperfect mirror
*****setting-up optical elements, beamline:bl1
Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 235.0
    Fy = 235.0
    arTr = array of size 2004002
    extTr = 1
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1001
             ny = 1001
             xFin = 0.0019690000000000003
             xStart = -0.0019690000000000003
             yFin = 0.0019690000000000003
             yStart = -0.0019690000000000003
             zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 55.0
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.002
    Dy = 0.0020936261413
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.002
    Dy = 0.0020936261413
    ap_or_ob = a
    shape = r
    x = 0
    y = 0
```

(continues on next page)

(continued from previous page)

```
Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.001
            xStart = -0.001
            yFin = 0.00104681307065
            yStart = -0.00104681307065
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 641.0
    treat = 0


Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 235.0
    Fy = 235.0
    arTr = array of size 2004002
    extTr = 1
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1001
            ny = 1001
            xFin = 0.0019690000000000003
            xStart = -0.0019690000000000003
            yFin = 0.0019690000000000003
            yStart = -0.0019690000000000003
            zStart = 0
```

(continues on next page)

(continued from previous page)

```
Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 55.0
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.002
    Dy = 0.0020936261413
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.002
    Dy = 0.0020936261413
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.001
            xStart = -0.001
            yFin = 0.00104681307065
            yStart = -0.00104681307065
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 641.0
    treat = 0


*****reading wavefront from h5 file...
```

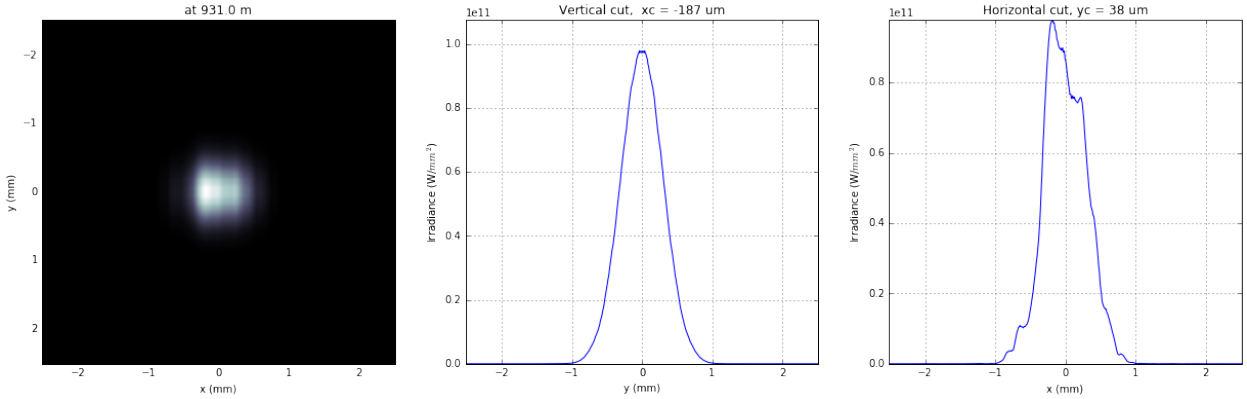
(continues on next page)

(continued from previous page)

```
R-space
nx    180   range_x [-1.0e+00, 1.0e+00] mm
ny    180   range_y [-1.0e+00, 1.0e+00] mm
*****propagating wavefront (with resizing)...
done
propagation lasted: 0.0 min
```

```
print ('*****Collimating CRL and imperfect mirror')
plot_wfront(mwf, 'at '+str(z1+z2+z3)+' m',False, False, 1e-4,1e-7,'x', True, saveDir=
↪'./'+strDataFolderName)
plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [mm], theta_fwhm [urad]: {}, {}'.format(
        calculate_fwhm_x(mwf)*1e3,calculate_fwhm_x(mwf)/(z1+z2)*1e6))
print('FWHMy [mm], theta_fwhm [urad]: {}, {}'.format(
        calculate_fwhm_y(mwf)*1e3,calculate_fwhm_y(mwf)/(z1+z2)*1e6))
```

```
*****Collimating CRL and imperfect mirror
FWHMx [mm]: 0.6784899532
FWHMy [mm]: 0.697875380434
Coordinates of center, [mm]: -0.187392463265 0.0387708544686
stepX, stepY [um]: 6.461809078096801 6.461809078096801

Total power (integrated over full range): 53.3003 [GW]
Peak power calculated using FWHM:         52.7227 [GW]
Max irradiance: 97.8661 [GW/mm^2]
R-space
FWHMx [mm], theta_fwhm [urad]: 0.6784899532001643, 2.3396205282764284
FWHMy [mm], theta_fwhm [urad]: 0.6978753804344545, 2.406466829084326
```



Propagating through BL2 beamline. Collimating CRL1, imperfect mirror, focusing CRL2

```
print ('*****Collimating CRL1, imperfect mirror, focusing CRL2')
bPlotted = False
isHlog = True
isVlog = False
bSaved = False
optBL = optBL2
strBL = 'bl2'
```

(continues on next page)

(continued from previous page)

```
pos_title = 'at sample'
print('*****setting-up optical elements, beamline: {}'.format(strBL))
if not NEW_SYNTAX:
    bl = Beamline(optBL)
else:
    bl = optBL
print(bl)

if bSaved:
    out_file_name = os.path.join(strDataFolderName, fname0+'_'+strBL+'.h5')
    print('save hdf5: {}'.format(out_file_name))
else:
    out_file_name = None

startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted: {} min'.format(round((time.time() - startTime) / 6.) / 10.
↪))
```

```
*****Collimating CRL1, imperfect mirror, focusing CRL2
*****setting-up optical elements, beamline: bl2
Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 235.0
    Fy = 235.0
    arTr = array of size 2004002
    extTr = 1
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1001
             ny = 1001
             xFin = 0.0019690000000000003
             xStart = -0.0019690000000000003
             yFin = 0.0019690000000000003
             yStart = -0.0019690000000000003
             zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 55.0
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.002
    Dy = 0.0020936261413
```

(continues on next page)

(continued from previous page)

```
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.002
    Dy = 0.0020936261413
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1500
             ny = 100
             xFin = 0.001
             xStart = -0.001
             yFin = 0.00104681307065
             yStart = -0.00104681307065
             zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 641.0
    treat = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 29.375
    Fy = 29.375
    arTr = array of size 2004002
    extTr = 1
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
```

(continues on next page)

(continued from previous page)

```
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1001
            ny = 1001
            xFin = 0.0019690000000000003
            xStart = -0.0019690000000000003
            yFin = 0.0019690000000000003
            yStart = -0.0019690000000000003
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.5, 1.8, 1.5, 1.8, 0, 0, 0]
    L = 29.375
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 1, 0.02, 2.0, 0.02, 2.0, 0, 0, 0]


Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 235.0
    Fy = 235.0
    arTr = array of size 2004002
    extTr = 1
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1001
            ny = 1001
            xFin = 0.0019690000000000003
            xStart = -0.0019690000000000003
            yFin = 0.0019690000000000003
            yStart = -0.0019690000000000003
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 55.0
    treat = 0

Optical Element: Aperture / Obstacle
```

(continues on next page)

(continued from previous page)

```
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.002
    Dy = 0.0020936261413
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.002
    Dy = 0.0020936261413
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1500
             ny = 100
             xFin = 0.001
             xStart = -0.001
             yFin = 0.00104681307065
             yStart = -0.00104681307065
             zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 641.0
    treat = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 29.375
    Fy = 29.375
    arTr = array of size 2004002
    extTr = 1
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
```

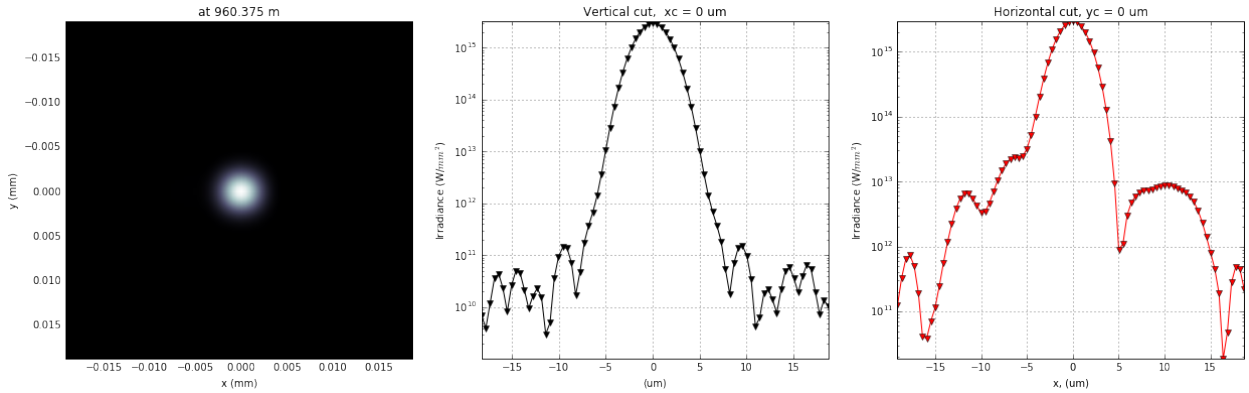
(continues on next page)

(continued from previous page)

```
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1001
            ny = 1001
            xFin = 0.0019690000000000003
            xStart = -0.0019690000000000003
            yFin = 0.0019690000000000003
            yStart = -0.0019690000000000003
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.5, 1.8, 1.5, 1.8, 0, 0, 0]
    L = 29.375
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 1, 0.02, 2.0, 0.02, 2.0, 0, 0, 0]


*****reading wavefront from h5 file...
R-space
nx   180  range_x [-1.0e+00, 1.0e+00] mm
ny   180  range_y [-1.0e+00, 1.0e+00] mm
*****propagating wavefront (with resizing)...
done
propagation lasted: 0.2 min
```

```
print ('*****Collimating CRL1, imperfect mirror, focusing CRL2')
plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m',True, True, 1e-4,1e-6,'x', True, saveDir=
↪'./'+strDataFolderName)
#plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [um], FWHMy [um]: {}, {}'.format(calculate_fwhm_y(mwf)*1e6,calculate_
↪fwhm_y(mwf)*1e6))
```

```
*****Collimating CRL1, imperfect mirror, focusing CRL2
FWHMx[um]: 3.18983482886
FWHMy [um]: 3.6455255187
Coordinates of center, [mm]: 0.0 0.0
stepX, stepY [um]: 0.45569068983763483 0.45569068983763483

Total power (integrated over full range): 44.8729 [GW]
Peak power calculated using FWHM:         38.874 [GW]
Max irradiance: 2.93824e+06 [GW/mm^2]
R-space
FWHMx [um], FWHMy [um]: 3.6455255187010787, 3.6455255187010787
```



1.10.3 Wavefront propagation simulation tutorial - Case 2

L.Samoylova liubov.samoylova@xfel.eu, A.Buzmakov buzmakov@gmail.com

Tutorial course on Wavefront Propagation Simulations, 28/11/2013, European XFEL, Hamburg.

Wave optics software is based on SRW core library https://github.com/ochubar/SRW, available through WPG interactive framework https://github.com/samoylv/WPG

Propagation Gaussian through HOM and KB optics: soft x-ray beamline

Import modules

```
%matplotlib inline
```

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

#Importing necessary modules:
import os
import sys
sys.path.insert(0,os.path.join('..','..'))

import time
import copy
import numpy as np
import pylab as plt


#import SRW core functions
from wpg.srwlib import srwl,SRWLOptD,SRWLOptA,SRWLOptC,SRWLOptT,SRWLOptL,SRWLOptMirEl

#import SRW helpers functions
from wpg.useful_code.srwutils import AuxTransmAddSurfHeightProfileScaled

#import some helpers functions
from wpg.useful_code.wfrutils import calculate_fwhm_x, plot_wfront, calculate_fwhm_y,␣
↪print_beamline, get_mesh, plot_1d, plot_2d
```

(continues on next page)

(continued from previous page)

```
from wpg.useful_code.wfrutils import propagate_wavefront
#Import base wavefront class
from wpg import Wavefront

#Gaussian beam generator
from wpg.generators import build_gauss_wavefront_xy

from wpg import Beamline
from wpg.optical_elements import Empty, Use_PP

# Fix for SRWLib plotting
plt.ion()
```

Define auxiliary functions

```
def calculate_source_fwhm(ekev, theta_fwhm):
    """
    Calculate source size from photon energy and FWHM angular divergence

    :param evev: Energy in keV
    :param theta_fwhm: theta_fwhm [units?]
    """
    wl = 12.39e-10/ekev
    k = 2 * np.sqrt(2*np.log(2))
    theta_sigma = theta_fwhm /k
    sigma0 = wl /(2*np.pi*theta_sigma)
    return sigma0*k

def calculate_theta_fwhm_cdr(ekev,qnC):
    """
    Calculate angular divergence using formula from XFEL CDR2011

    :param ekev: Energy in keV
    :param qnC: e-bunch charge, [nC]
    :return: theta_fwhm [units?]
    """
    theta_fwhm = (17.2 - 6.4 * np.sqrt(qnC))*1e-6/ekev**0.85
    return theta_fwhm

def defineOPD(opTrErMirr, mdatafile, ncol, delim, Orient, theta, scale):
    """
    Define optical path difference (OPD) from mirror profile, i.e. ill the struct␣
↪opTrErMirr

    :params mdatafile: an ascii file with mirror profile data
    :params ncol: number of columns in the file
    :params delim: delimiter between numbers in an row, can be space (' '), tab '\t',␣
↪etc
    :params orient: mirror orientation, 'x' (horizontal) or 'y' (vertical)
    :params theta: incidence angle
    :params scale: scaling factor for the mirror profile
    """
    heightProfData = np.loadtxt(mdatafile).T
    AuxTransmAddSurfHeightProfileScaled(opTrErMirr, heightProfData, Orient, theta,␣
↪scale)
```

(continues on next page)

(continued from previous page)

```
    plt.figure()
    plot_1d(heightProfData,'profile from ' + mdatafile,'x (m)', 'h (m)')
```

Defining initial wavefront and writing electric field data to h5-file

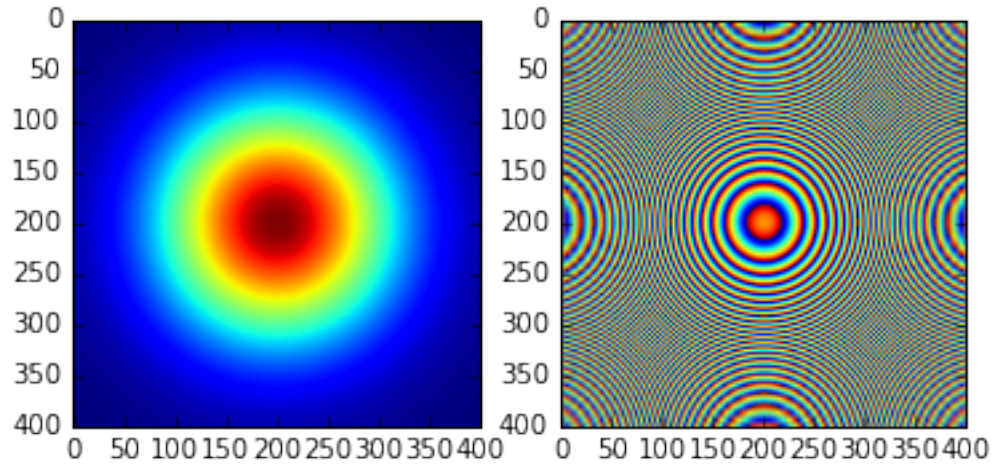
```
# #**********************Input Wavefront Structure and Parameters
print('*****defining initial wavefront and writing electric field data to h5-file...')
strInputDataFolder = 'data_common'  # input data sub-folder name
strOutputDataFolder = 'Tutorial_case_2'  # output data sub-folder name

#init Gauusian beam parameters
d2m1_sase1 = 246.5
d2m1_sase2 = 290.0
d2m1_sase3 = 281.0
d2hkb_sase1 = 904.0
d2hkb_sase3 = 442.3
dHKB_foc_sase3    = 2.715      # nominal focal length for HFM KB
dVKB_foc_sase3    = 1.715      # nominal focal length for VFM KB


qnC = 0.1                     # e-bunch charge, [nC]
ekev_sase3 = 0.8
thetaOM_sase3 = 9.e-3
thetaKB_sase3 = 9.e-3
ekev_sase1 = 8.0
thetaOM_sase1 = 2.5e-3        #
thetaKB_sase1 = 3.5e-3

ekev = ekev_sase3
thetaOM = thetaOM_sase3
d2m1 = d2m1_sase3
d2hkb = d2hkb_sase3
thetaKB = thetaKB_sase3
dhkb_foc = dHKB_foc_sase3      # nominal focal length for HFM KB
dvkb_foc = dVKB_foc_sase3      # nominal focal length for VFM KB
dhkb_vkb = dhkb_foc - dvkb_foc          # distance between centers of HFM and VFM

z1 = d2m1
theta_fwhm = calculate_theta_fwhm_cdr(ekev,qnC)
k = 2*np.sqrt(2*np.log(2))
sigX = 12.4e-10*k/(ekev*4*np.pi*theta_fwhm)
print('waist_fwhm [um], theta_fwhms [urad]:{}, {}'.format(sigX*k*1e6, theta_fwhm*1e6))
#define limits
range_xy = theta_fwhm/k*z1*5. # sigma*4 beam size
npoints=400

#define unique filename for storing results
ip = np.floor(ekev)
frac = np.floor((ekev - ip)*1e3)
fname0 = 'g' + str(int(ip))+'_'+str(int(frac))+'kev'
print('save hdf5: '+fname0+'.h5')
ifname = os.path.join(strOutputDataFolder,fname0+'.h5')

#build SRW gauusian wavefront
```

(continues on next page)

(continued from previous page)

```
wfr0=build_gauss_wavefront_xy(nx=npoints, ny=npoints, ekev=ekev,
                              xMin=-range_xy/2 ,xMax=range_xy/2,
                              yMin=-range_xy/2, yMax=range_xy/2,
                              sigX=sigX, sigY=sigX, d2waist=z1)



#init WPG Wavefront helper class
mwf = Wavefront(wfr0)

#store wavefront to HDF5 file
mwf.store_hdf5(ifname)

#draw wavefront with common functions
plt.subplot(1,2,1)
plt.imshow(mwf.get_intensity(slice_number=0))
plt.subplot(1,2,2)
plt.imshow(mwf.get_phase(slice_number=0,polarization='horizontal'))
plt.show()

#draw wavefront with cuts
plot_wfront(mwf, title_fig='at '+str(z1)+' m',
            isHlog=False, isVlog=False,
            i_x_min=1e-5, i_y_min=1e-5, orient='x', onePlot=True)

plt.set_cmap('bone') #set color map, 'bone', 'hot', 'jet', etc
fwhm_x = calculate_fwhm_x(mwf)
print('FWHMx [mm], theta_fwhm [urad]: {}, {}'.format(fwhm_x*1e3,fwhm_x/z1*1e6))
```

```
*****defining initial wavefront and writing electric field data to h5-file...
waist_fwhm [um], theta_fwhms [urad]:37.282272901778825, 18.34572592382333
save hdf5: g0_800kev.h5
```

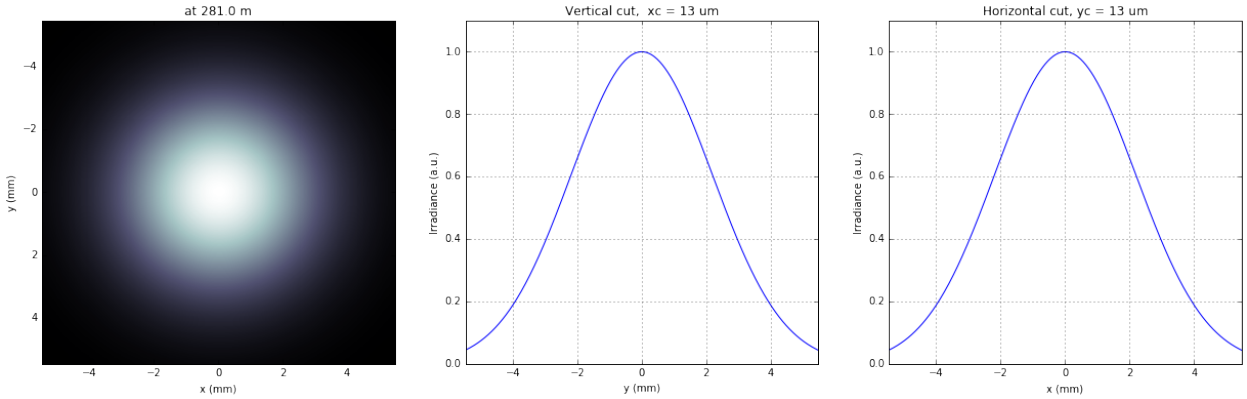



```
FWHMx [mm]: 5.13005725474
FWHMy [mm]: 5.13005725474
Coordinates of center, [mm]: 0.0137167306277 0.0137167306277
stepX, stepY [um]: 27.433461255317237 27.433461255317237
```

(continues on next page)

(continued from previous page)

```
R-space
FWHMx [mm], theta_fwhm [urad]: 5.130057254744322, 18.25643151154563
```



Defining optical beamline(s)

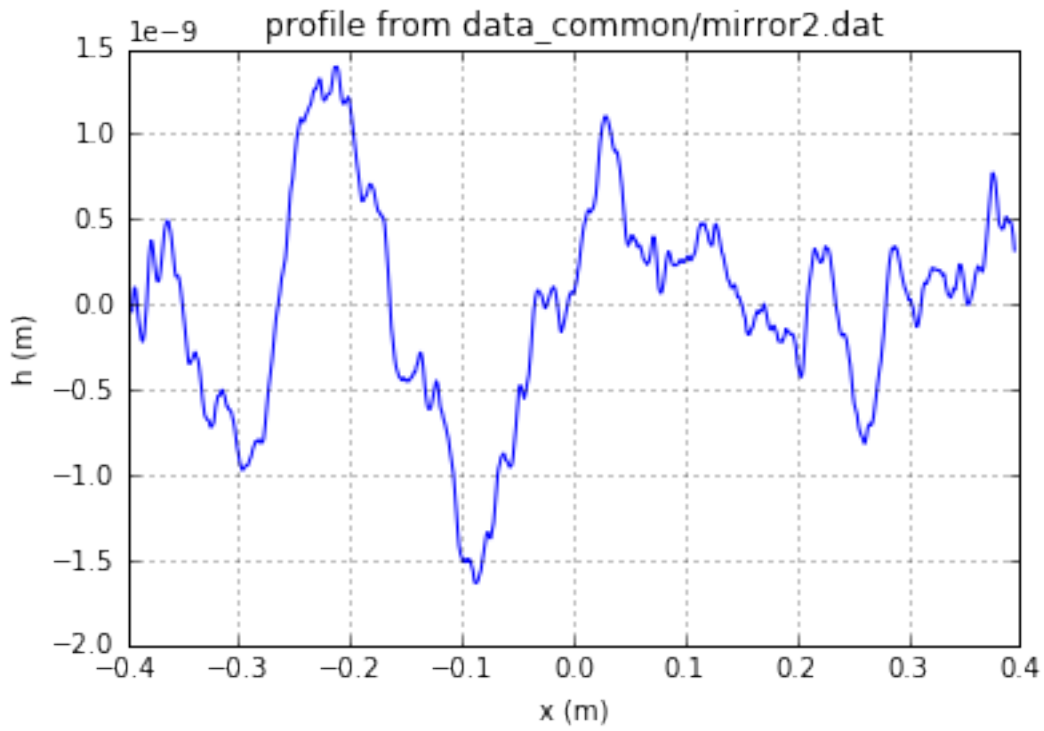
```
print('*****Defining optical beamline(s) ...')

z2 = d2hkb - d2m1

DriftM1_KB = SRWLOptD(z2) #Drift from first offset mirror (M1) to exp hall
horApM1 = 0.8*thetaOM
opApM1 = SRWLOptA('r', 'a', horApM1, range_xy)  # clear aperture of the Offset␣
↪Mirror(s)
horApKB = 0.8 * thetaKB # Aperture of the KB system, CA 0.8 m
opApKB = SRWLOptA('r', 'a', horApKB, horApKB)  # clear aperture of the Offset␣
↪Mirror(s)

#Wavefront Propagation Parameters:
#[0]:  Auto-Resize (1) or not (0) Before propagation
#[1]:  Auto-Resize (1) or not (0) After propagation
#[2]:  Relative Precision for propagation with Auto-Resizing (1. is nominal)
#[3]:  Allow (1) or not (0) for semi-analytical treatment of quadratic phase terms at␣
↪propagation
#[4]:  Do any Resizing on Fourier side, using FFT, (1) or not (0)
#[5]:  Horizontal Range modification factor at Resizing (1. means no modification)
#[6]:  Horizontal Resolution modification factor at Resizing
#[7]:  Vertical Range modification factor at Resizing
#[8]:  Vertical Resolution modification factor at Resizing
#[9]:  Type of wavefront Shift before Resizing (not yet implemented)
#[10]: New Horizontal wavefront Center position after Shift (not yet implemented)
#[11]: New Vertical wavefront Center position after Shift (not yet implemented)
#                  [ 0] [1] [2]  [3] [4] [5]  [6]  [7]  [8]  [9] [10] [11]
ppM1 =             [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppTrErM1 =         [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDriftM1_KB =     [ 0,  0, 1.0,  1,  0, 2.4, 1.8, 2.4, 1.8,  0,  0,   0]
ppApKB =           [ 0,  0, 1.0,  0,  0, 0.6, 8.0, 0.6, 4.0,  0,  0,   0]
ppHKB =            [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppTrErHKB =        [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDrift_HKB_foc =  [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDrift_KB =       [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
```

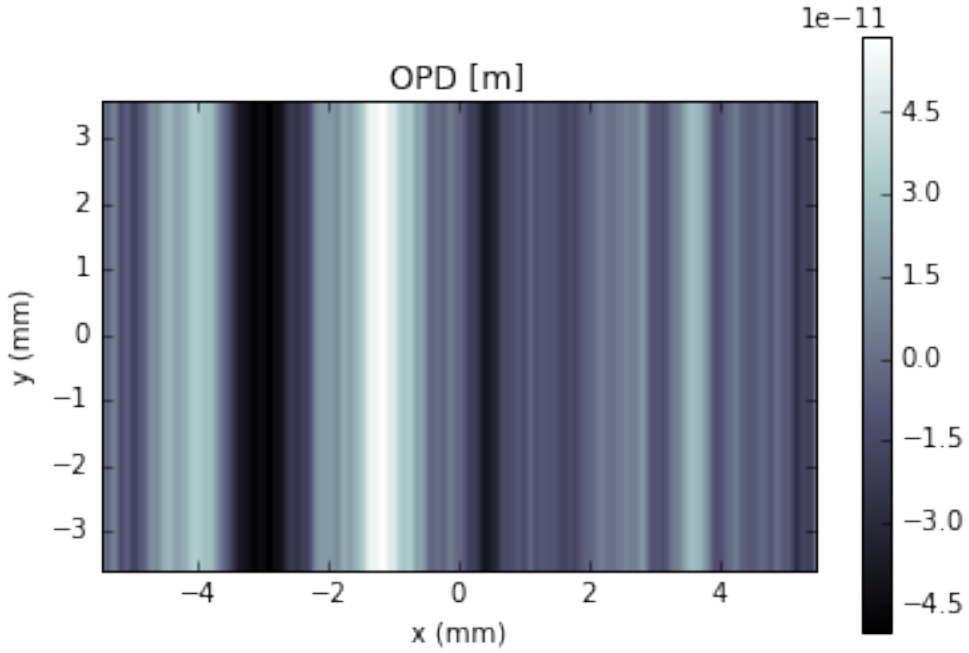
(continues on next page)

(continued from previous page)

```
ppVKB =           [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppTrErVKB =       [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDrift_foc =     [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
#ppFin  =          [ 0,  0, 1.0,  0,  0, 0.05,5.0, 0.05,5.0,  0,  0,   0]
ppFin =           [ 0,  0, 1.0,  0,  1, .01, 20.0, .01, 20.0,  0,  0,   0]

optBL0 = SRWLOptC([opApM1,  DriftM1_KB],
                  [ppM1,ppDriftM1_KB])

scale = 2     #5 mirror profile scaling factor
print('*****HOM1 data for BL1 beamline ')
opTrErM1 = SRWLOptT(1500, 100, horApM1, range_xy)
#defineOPD(opTrErM1, os.path.join(strInputDataFolder,'mirror1.dat'), 2, '\t', 'x',
↪thetaOM, scale)
defineOPD(opTrErM1, os.path.join(strInputDataFolder,'mirror2.dat'), 2, ' ', 'x',
↪thetaOM, scale)
opdTmp=np.array(opTrErM1.arTr)[1::2].reshape(opTrErM1.mesh.ny,opTrErM1.mesh.nx)
plt.figure()
plot_2d(opdTmp, opTrErM1.mesh.xStart*1e3,opTrErM1.mesh.xFin*1e3,opTrErM1.mesh.
↪yStart*1e3,opTrErM1.mesh.yFin*1e3,
        'OPD [m]', 'x (mm)', 'y (mm)')

optBL1 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB],
                  [ppM1,ppTrErM1,ppDriftM1_KB])

dhkb_vkb = dhkb_foc - dvkb_foc          # distance between centers of HFM and VFM
d2vkb = d2hkb +  dhkb_vkb
vkbfoc =  1. /(1./dvkb_foc + 1. / d2vkb) # for thin lens approx
hkbfoc =  1. /(1./dhkb_foc + 1. / d2hkb) # for thin lens approx

z3 = dhkb_vkb
z4 = vkbfoc #distance to focal plane

#HKB = SRWLOptMirEl(_p=d2hkb, _q=dhkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=cos(thetaKB), _nvy=0, _nvz=-sin(thetaKB), _tvx=-sin(thetaKB), _
↪tvy=0, _x=0, _y=0, _treat_in_out=1) #HKB Ellipsoidal Mirror
#VKB = SRWLOptMirEl(_p=d2vkb, _q=dvkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=0, _nvy=cos(thetaKB), _nvz=-sin(thetaKB), _tvx=0, _tvy=-
↪sin(thetaKB), _x=0, _y=0, _treat_in_out=1) #VKB Ellipsoidal Mirror
HKB = SRWLOptL(hkbfoc) #HKB as Thin Lens
VKB = SRWLOptL(1e23,vkbfoc) #VKB as Thin Lens
Drift_KB  = SRWLOptD(z3)
Drift_foc = SRWLOptD(z4)
optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,  Drift_
↪foc],
                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,ppDrift_
↪foc,ppFin])
```

```
*****Defining optical beamline(s) ...
*****HOM1 data for BL1 beamline
```





Propagating through BL0 beamline. Ideal mirror: HOM as an aperture

```
print('*****Ideal mirror: HOM as an aperture')
bPlotted = False
isHlog = False
isVlog = False
```

(continues on next page)

(continued from previous page)

```
bSaved = True
optBL = optBL0
strBL = 'bl0'
pos_title = 'at exp hall wall'
print('*****setting-up optical elements, beamline: {}'.format(strBL))
bl = Beamline(optBL)
print(bl)

if bSaved:
    out_file_name = os.path.join(strOutputDataFolder, fname0+'_'+strBL+'.h5')
    print('save hdf5: {}'.format(out_file_name))
else:
    out_file_name = None

startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted: {} min'.format(round((time.time() - startTime) / 6.) / 10.
↪))
```

```
*****Ideal mirror: HOM as an aperture
*****setting-up optical elements, beamline: bl0
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0109459510409
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 161.3
    treat = 0


save hdf5: Tutorial_case_2/g0_800kev_bl0.h5
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0109459510409
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 161.3
    treat = 0


*****reading wavefront from h5 file...
R-space
nx   400  range_x [-5.5e+00, 5.5e+00] mm
ny   400  range_y [-5.5e+00, 5.5e+00] mm
```

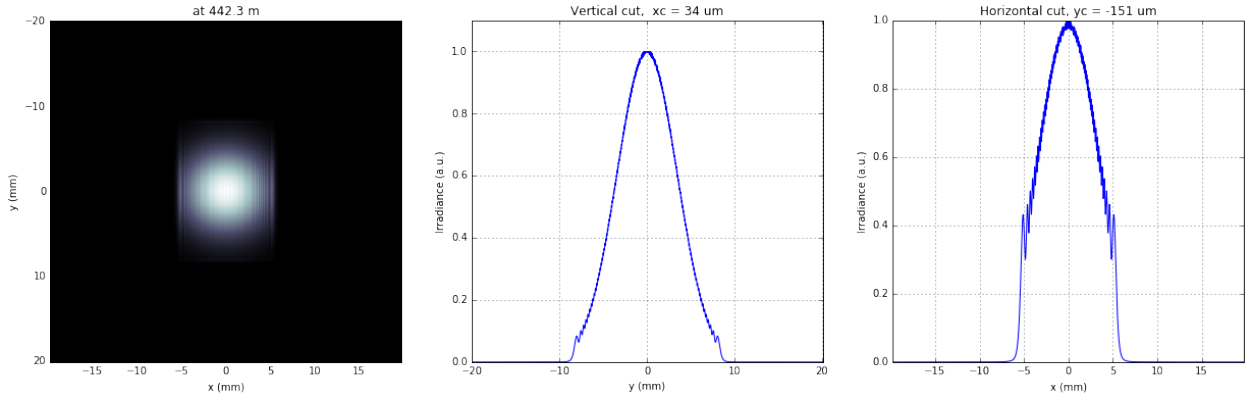
(continues on next page)

(continued from previous page)

```
*****propagating wavefront (with resizing)...
save hdf5: Tutorial_case_2/g0_800kev_bl0.h5
done
propagation lasted: 0.1 min
```

```
print('*****Ideal mirror: HOM as an aperture')
plot_wfront(mwf, 'at '+str(z1+z2)+' m',False, False, 1e-5,1e-5,'x', True)
plt.set_cmap('bone') #set color map, 'bone', 'hot', 'jet', etc
plt.axis('tight')
print('FWHMx [mm], theta_fwhm [urad]: {}, {}'.format(calculate_fwhm_x(mwf)*1e3,
↪calculate_fwhm_x(mwf)/(z1+z2)*1e6))
print('FWHMy [mm], theta_fwhm [urad]:{}, {}'.format(calculate_fwhm_y(mwf)*1e3,
↪calculate_fwhm_y(mwf)/(z1+z2)*1e6))
```

```
*****Ideal mirror: HOM as an aperture
FWHMx [mm]: 8.5730592677
FWHMy [mm]: 8.1457466277
Coordinates of center, [mm]: 0.0342922370708 -0.15171161341
stepX, stepY [um]: 22.86149138052557 23.34024821691403

R-space
FWHMx [mm], theta_fwhm [urad]: 8.57305926769709, 19.38290587315643
FWHMy [mm], theta_fwhm [urad]:8.145746627702996, 18.41679092856205
```



Propagating through BL1 beamline. Imperfect mirror, at KB aperture

```
print ('*****Imperfect mirror, at KB aperture')
bPlotted = False
isHlog = True
isVlog = False
bSaved = False
optBL = optBL1
strBL = 'bl1'
pos_title = 'at exp hall wall'
print('*****setting-up optical elements, beamline:', strBL)
bl = Beamline(optBL)
print(bl)

if bSaved:
```

(continues on next page)

(continued from previous page)

```
    out_file_name = os.path.join(strOutputDataFolder, fname0+'_'+strBL+'.h5')
    print('save hdf5:', out_file_name)
else:
    out_file_name = None

startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Imperfect mirror, at KB aperture
*****setting-up optical elements, beamline: bl1
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0109459510409
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1500
             ny = 100
             xFin = 0.0036
             xStart = -0.0036
             yFin = 0.00547297552044
             yStart = -0.00547297552044
             zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 161.3
    treat = 0


Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0109459510409
```

(continues on next page)

(continued from previous page)

```
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0036
            xStart = -0.0036
            yFin = 0.00547297552044
            yStart = -0.00547297552044
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 161.3
    treat = 0


*****reading wavefront from h5 file...
R-space
nx   400  range_x [-5.5e+00, 5.5e+00] mm
ny   400  range_y [-5.5e+00, 5.5e+00] mm
*****propagating wavefront (with resizing)...
done
propagation lasted: 0.1 min
```

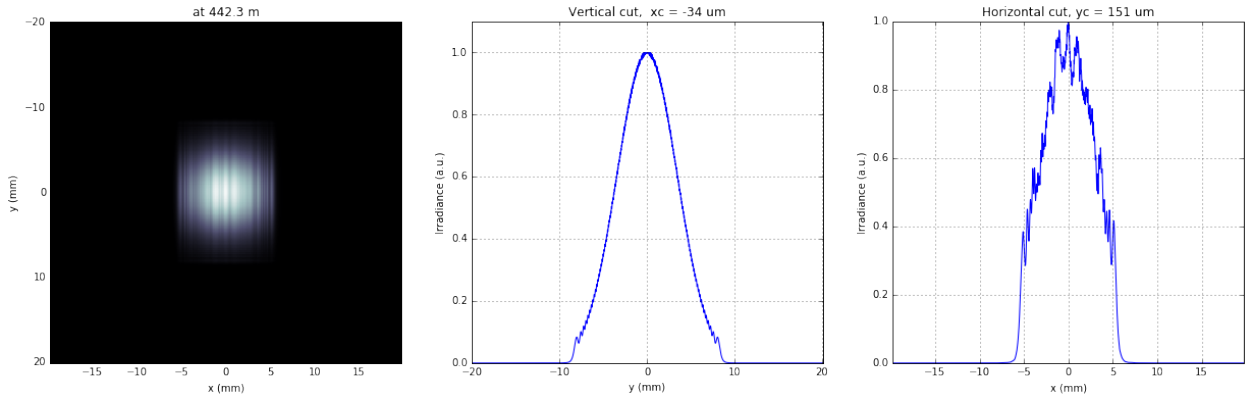
```
print ('*****Imperfect mirror, at KB aperture')
plot_wfront(mwf, 'at '+str(z1+z2)+' m',False, False, 1e-5,1e-5,'x', True)
plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [mm], theta_fwhm [urad]:',calculate_fwhm_x(mwf)*1e3,calculate_fwhm_
↪x(mwf)/(z1+z2)*1e6)
print('FWHMy [mm], theta_fwhm [urad]:',calculate_fwhm_y(mwf)*1e3,calculate_fwhm_
↪y(mwf)/(z1+z2)*1e6)
```

```
*****Imperfect mirror, at KB aperture
FWHMx [mm]: 7.93322911953
```

(continues on next page)

(continued from previous page)

```
FWHMy [mm]: 8.1457466277
Coordinates of center, [mm]: -0.0342934976348 0.15171161341
stepX, stepY [um]: 22.86233175656062 23.34024821691403

R-space
FWHMx [mm], theta_fwhm [urad]: 7.93322911953 17.936308206
FWHMy [mm], theta_fwhm [urad]: 8.1457466277 18.4167909286
```



Propagating through BL2 beamline. Focused beam: perfect KB

```
print('*****Focused beam: perfect KB')
#optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,
↪Drift_foc],
#                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,
↪ppDrift_foc])
z3 = dhkb_vkb
z4 = vkbfoc #distance to focal plane

#HKB = SRWLOptMirEl(_p=d2hkb, _q=dhkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=cos(thetaKB), _nvy=0, _nvz=-sin(thetaKB), _tvx=-sin(thetaKB), _
↪tvy=0, _x=0, _y=0, _treat_in_out=1) #HKB Ellipsoidal Mirror
#VKB = SRWLOptMirEl(_p=d2vkb, _q=dvkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=0, _nvy=cos(thetaKB), _nvz=-sin(thetaKB), _tvx=0, _tvy=-
↪sin(thetaKB), _x=0, _y=0, _treat_in_out=1) #VKB Ellipsoidal Mirror
#HKB = SRWLOptL(hkbfoc) #HKB as Thin Lens
#VKB = SRWLOptL(1e23,vkbfoc) #VKB as Thin Lens
Drift_foc = SRWLOptD(dvkb_foc)
optBL2 = SRWLOptC([opApM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,  Drift_foc],
                  [ppM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,ppDrift_foc])
optBL = optBL2
strBL = 'bl2'
pos_title = 'at sample position'
print('*****setting-up optical elements, beamline:', strBL)

bl = Beamline(optBL)
bl.append(Empty(), Use_PP(zoom=0.02, sampling=5.0))

print(bl)

if bSaved:
```

(continues on next page)

(continued from previous page)

```
    out_file_name = os.path.join(strOutputDataFolder, fname0+'_'+strBL+'.h5')
    print('save hdf5:', out_file_name)
else:
    out_file_name = None

startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Focused beam: perfect KB
*****setting-up optical elements, beamline: bl2
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0109459510409
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 161.3
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0072
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 2.698436007775019
    Fy = 1e+23
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 0.9999999999999998
    treat = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1.7083907284024138
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.715
```

(continues on next page)

(continued from previous page)

```
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 0, 0.02, 5.0, 0.02, 5.0, 0, 0, 0]


Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0109459510409
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 161.3
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0072
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 2.698436007775019
    Fy = 1e+23
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 0.9999999999999998
    treat = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1.7083907284024138
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.715
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront
```

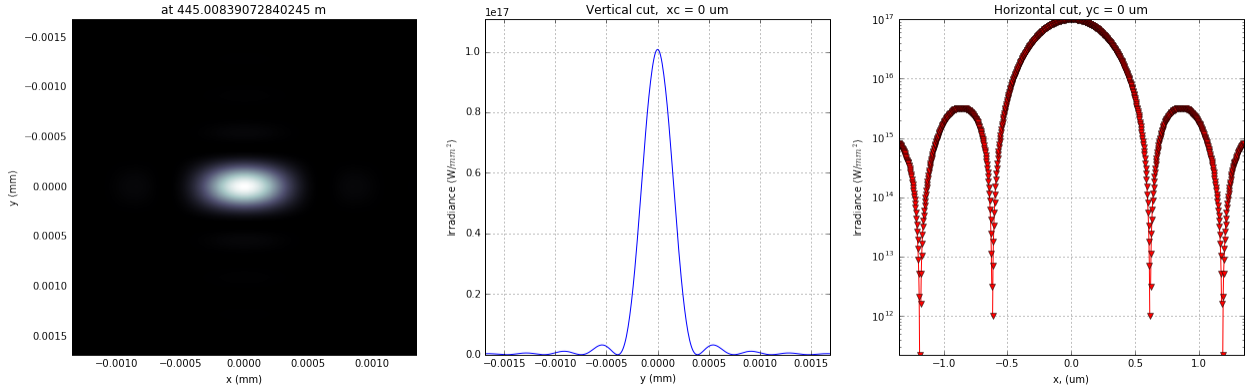
(continues on next page)

(continued from previous page)

```
Prop. parameters = [0, 0, 1.0, 0, 0, 0.02, 5.0, 0.02, 5.0, 0, 0, 0]


*****reading wavefront from h5 file...
R-space
nx    400   range_x [-5.5e+00, 5.5e+00] mm
ny    400   range_y [-5.5e+00, 5.5e+00] mm
*****propagating wavefront (with resizing)...
done
propagation lasted: 1.6 min
```

```
print('*****Focused beam: Focused beam: perfect KB')
bOnePlot = True
isHlog = True
isVlog = False
bSaved = False
try:
    plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m',isHlog, isVlog, 1e-2,1e-3,'x',␣
↪bOnePlot)
except ValueError as e:
    print(e)
plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [um], FWHMy [um]:',calculate_fwhm_x(mwf)*1e6,calculate_fwhm_y(mwf)*1e6)
```

```
*****Focused beam: Focused beam: perfect KB
FWHMx[um]: 0.536754127757
FWHMy [mm]: 0.000332922388264
Coordinates of center, [mm]: 1.62652765991e-06 4.06002912521e-06
stepX, stepY [um]: 0.003253055319740485 0.008120058250345424

Total power (integrated over full range): 20.8782 [GW]
Peak power calculated using FWHM:         20.4748 [GW]
Max irradiance: 1.00706e+08 [GW/mm^2]
R-space
FWHMx [um], FWHMy [um]: 0.536754127757 0.332922388264
```



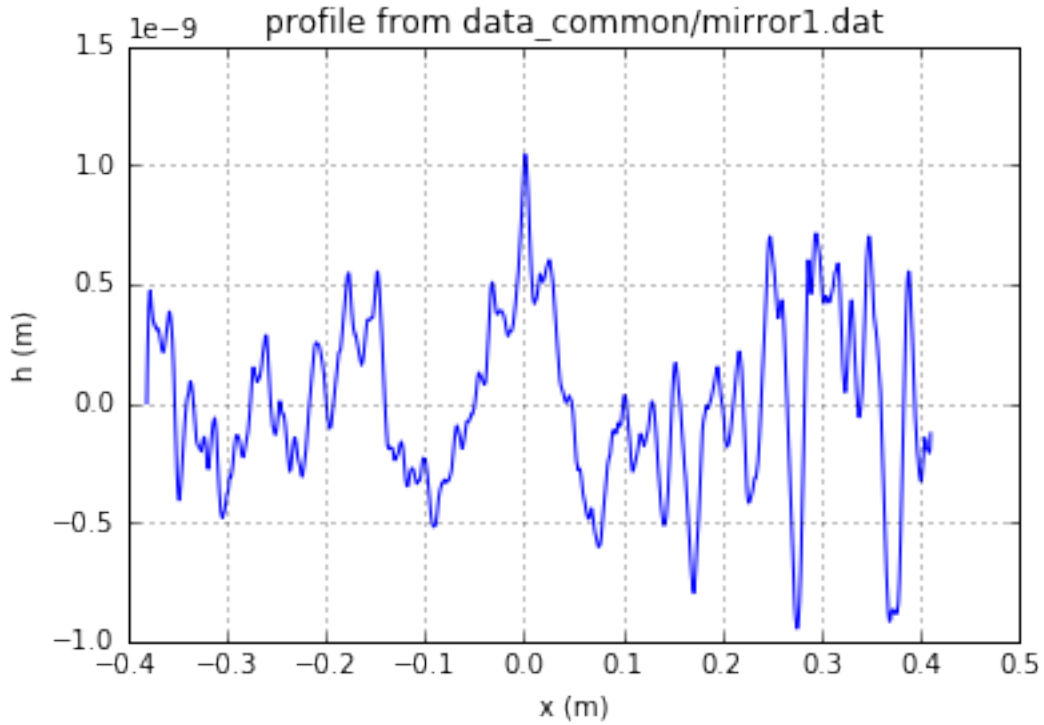

```
opTrErHKB = SRWLOptT(1500, 100, horApKB, horApKB)
defineOPD(opTrErHKB, os.path.join(strInputDataFolder,'mirror1.dat'), 2, '\t', 'x', ␣
↪thetaOM, scale)
```

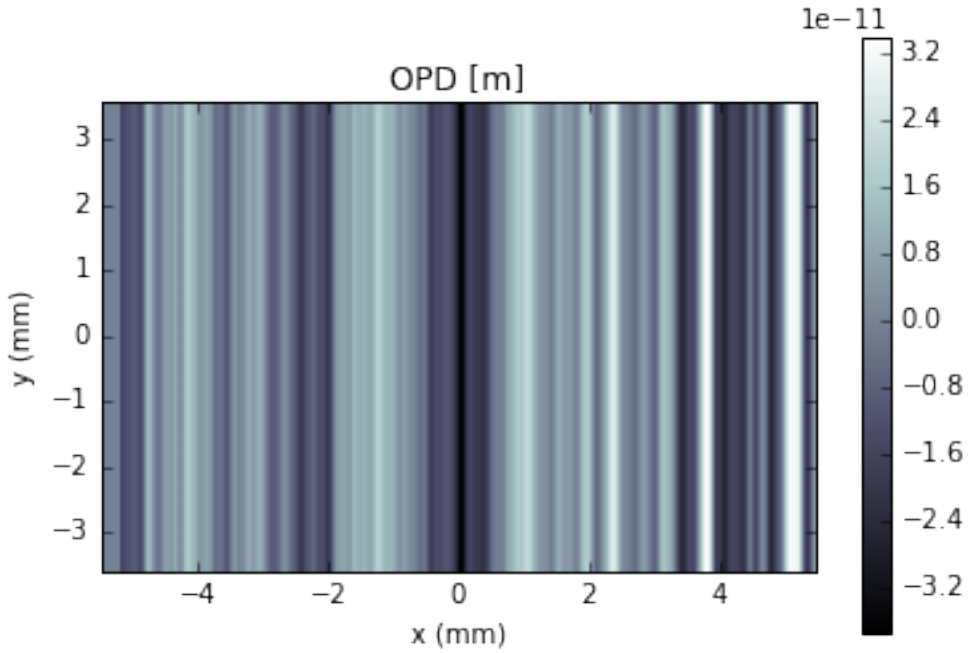
(continues on next page)

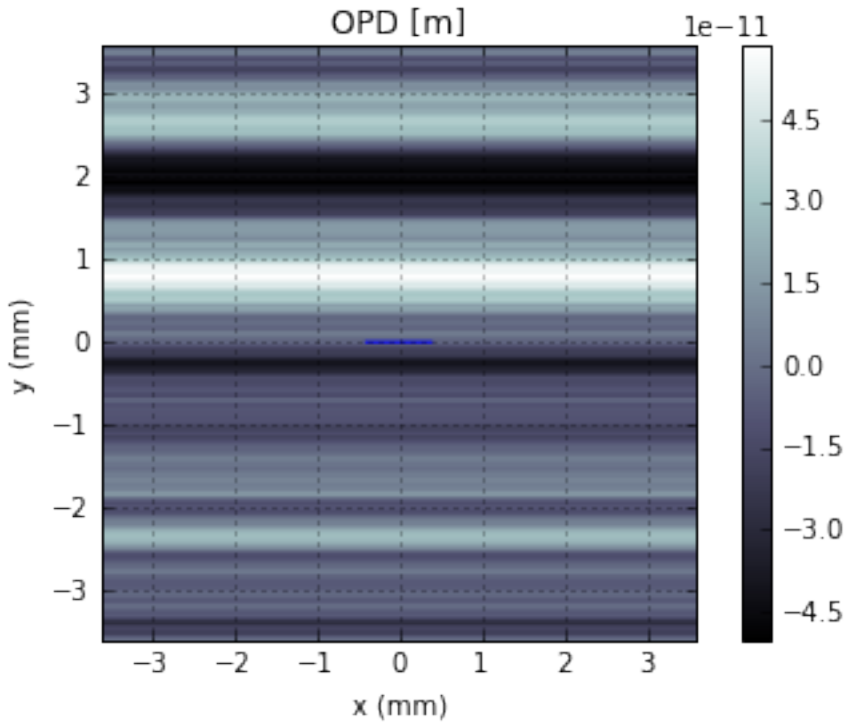
(continued from previous page)

```
opdTmp=np.array(opTrErHKB.arTr)[1::2].reshape(opTrErHKB.mesh.ny,opTrErHKB.mesh.nx)
print('*****HKB data  ')
plt.figure()
#subplot()
plot_2d(opdTmp, opTrErM1.mesh.xStart*1e3,opTrErM1.mesh.xFin*1e3,opTrErM1.mesh.
↪yStart*1e3,opTrErM1.mesh.yFin*1e3,
        'OPD [m]', 'x (mm)', 'y (mm)')
print('*****VKB data  ')
opTrErVKB = SRWLOptT(100, 1500, horApKB, horApKB)
defineOPD(opTrErVKB, os.path.join(strInputDataFolder,'mirror2.dat'), 2, ' ', 'y', ␣
↪thetaOM, scale)
opdTmp=np.array(opTrErVKB.arTr)[1::2].reshape(opTrErVKB.mesh.ny,opTrErVKB.mesh.nx)
#subplot()
plot_2d(opdTmp, opTrErVKB.mesh.xStart*1e3,opTrErVKB.mesh.xFin*1e3,opTrErVKB.mesh.
↪yStart*1e3,opTrErVKB.mesh.yFin*1e3,
        'OPD [m]', 'x (mm)', 'y (mm)')
print(vkbfoc-dvkb_foc)
```

```
*****HKB data
*****VKB data
-0.006609271597586286
```








```
print ('*****Focused beam behind focus: perfect KB')
#optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,
↪Drift_foc],
#                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,
↪ppDrift_foc])
z3 = dhkb_vkb
#z4 = dvkb_foc #distance to focal plane
```

(continues on next page)

(continued from previous page)

```
z4 = vkbfoc

#HKB = SRWLOptMirEl(_p=d2hkb, _q=dhkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=cos(thetaKB), _nvy=0, _nvz=-sin(thetaKB), _tvx=-sin(thetaKB), _
↪tvy=0, _x=0, _y=0, _treat_in_out=1) #HKB Ellipsoidal Mirror
#VKB = SRWLOptMirEl(_p=d2vkb, _q=dvkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=0, _nvy=cos(thetaKB), _nvz=-sin(thetaKB), _tvx=0, _tvy=-
↪sin(thetaKB), _x=0, _y=0, _treat_in_out=1) #VKB Ellipsoidal Mirror
#HKB = SRWLOptL(hkbfoc) #HKB as Thin Lens
#VKB = SRWLOptL(1e23,vkbfoc) #VKB as Thin Lens
Drift_foc = SRWLOptD(z4)
optBL2 = SRWLOptC([opApM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,  Drift_foc],
                  [ppM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,ppDrift_foc])
optBL = optBL2
strBL = 'bl2'
pos_title = 'at sample position'
print('*****setting-up optical elements, beamline:', strBL)
bl = Beamline(optBL)

print(bl)

if bSaved:
    out_file_name = os.path.join(strOutputDataFolder, fname0+'_'+strBL+'.h5')
    print('save hdf5:', out_file_name)
else:
    out_file_name = None

startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Focused beam behind focus: perfect KB
*****setting-up optical elements, beamline: bl2
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0109459510409
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 161.3
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0072
    ap_or_ob = a
    shape = r
    x = 0
    y = 0
```

(continues on next page)

(continued from previous page)

```
Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 2.698436007775019
    Fy = 1e+23
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 0.9999999999999998
    treat = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1.7083907284024138
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.7083907284024138
    treat = 0


Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0109459510409
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 161.3
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0072
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 2.698436007775019
    Fy = 1e+23
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
```

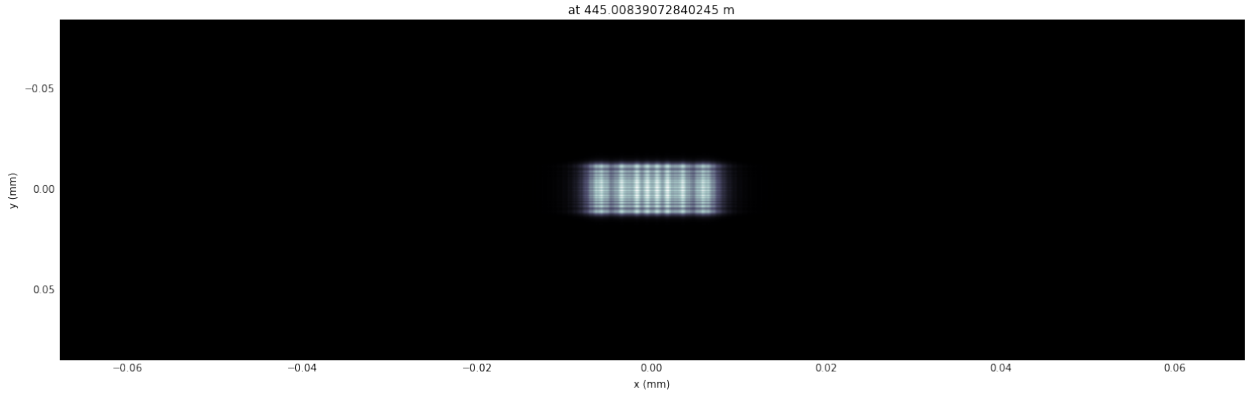
(continues on next page)

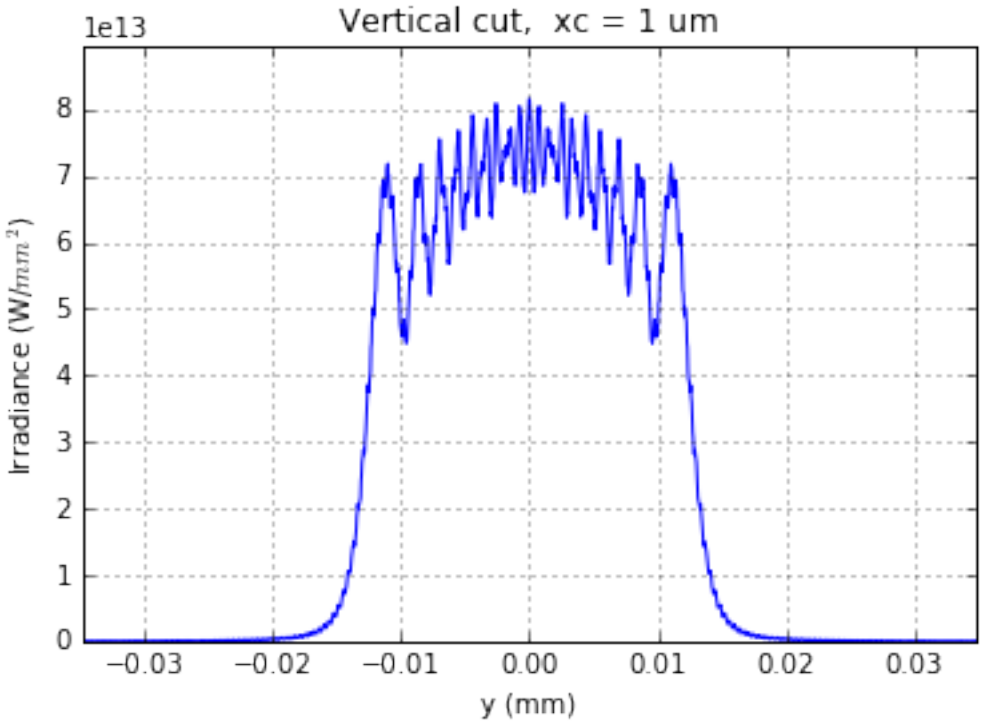
(continued from previous page)

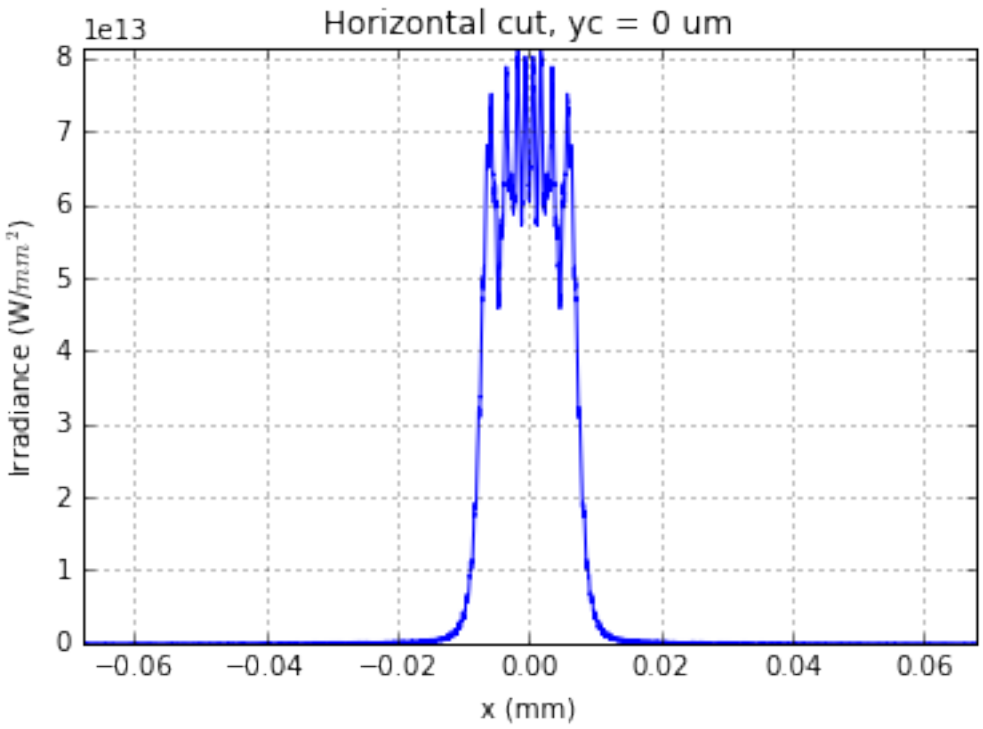
```
    L = 0.9999999999999998
    treat = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1.7083907284024138
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.7083907284024138
    treat = 0


*****reading wavefront from h5 file...
R-space
nx   400  range_x [-5.5e+00, 5.5e+00] mm
ny   400  range_y [-5.5e+00, 5.5e+00] mm
*****propagating wavefront (with resizing)...
done
propagation lasted: 1.5 min
```

```
print ('*****Focused beam: Focused beam: perfect KB')
bOnePlot = False
isHlog = False
isVlog = False
bSaved = False
plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m',isHlog, isVlog, 1e-3,1e-3,'x', bOnePlot)
plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [um], FWHMy [um]:',calculate_fwhm_x(mwf)*1e6,calculate_fwhm_y(mwf)*1e6)
```

```
*****Focused beam: Focused beam: perfect KB
FWHMx [mm]: 0.0145065362204
FWHMy [mm]: 0.0246160199519
Coordinates of center, [mm]: 0.00170521151297 2.03438181421e-05
stepX, stepY [um]: 0.01631781352120671 0.04068763628414966

Total power (integrated over full range): 21.9029 [GW]
Peak power calculated using FWHM:         33.1407 [GW]
Max irradiance: 81571 [GW/mm^2]
R-space
FWHMx [um], FWHMy [um]: 14.5065362204 24.6160199519
```


at 445.00839072840245 m
−0.05
0.00
0.05
y (mm)
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
x (mm)


Vertical cut, xc = 1 um
1e13
Irradiance (W/mm²)
0
1
2
3
4
5
6
7
8
−0.03
−0.02
−0.01
0.00
0.01
0.02
0.03
y (mm)



1.10.4 Wavefront propagation simulation tutorial - Case 2_new

L.Samoylova liubov.samoylova@xfel.eu, A.Buzmakov buzmakov@gmail.com

Tutorial course on Wavefront Propagation Simulations, 28/11/2013, European XFEL, Hamburg.

Wave optics software is based on SRW core library https://github.com/ochubar/SRW, available through WPG interactive framework https://github.com/samoylv/WPG

Propagation Gaussian through HOM and KB optics: soft x-ray beamline

Import modules

```
%matplotlib inline
```

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

#Importing necessary modules:
import os
import sys
sys.path.insert(0,os.path.join('..','..'))

import time
import copy
import numpy as np
```

(continues on next page)

(continued from previous page)

```
import pylab as plt


#import SRW core functions
from wpg.srwlib import srwl,SRWLOptD,SRWLOptA,SRWLOptC,SRWLOptT,SRWLOptL,SRWLOptMirEl

#import SRW helpers functions
from wpg.useful_code.srwutils import AuxTransmAddSurfHeightProfileScaled

#import some helpers functions
from wpg.useful_code.wfrutils import calculate_fwhm_x, plot_wfront, calculate_fwhm_y,␣
↪print_beamline, get_mesh

#Import base wavefront class
from wpg import Wavefront

#Gaussian beam generator
from wpg.generators import build_gauss_wavefront_xy

plt.ion()
```

Define auxiliary functions

```
#Plotting
def plot_1d(profile, title_fig, title_x, title_y):
    plt.plot(profile[0], profile[1])
    plt.xlabel(title_x)
    plt.ylabel(title_y)
    plt.title(title_fig)
    plt.grid(True)


def plot_2d(amap, xmin, xmax, ymin, ymax, title_fig, title_x, title_y):
    plt.imshow(amap, extent=(ymin, ymax, xmin, xmax))
    plt.colorbar()
    plt.xlabel(title_x)
    plt.ylabel(title_y)
    plt.title(title_fig)
```

```
#calculate source size from photon energy and FWHM angular divergence
def calculate_source_fwhm(ekev, theta_fwhm):
    wl = 12.39e-10/ekev
    k = 2 * np.sqrt(2*np.log(2))
    theta_sigma = theta_fwhm /k
    sigma0 = wl /(2*np.pi*theta_sigma)
    return sigma0*k
```

```
#calculate angular divergence using formula from CDR2011
def calculate_theta_fwhm_cdr(ekev,qnC):
    theta_fwhm = (17.2 - 6.4 * np.sqrt(qnC))*1e-6/ekev**0.85
    return theta_fwhm
```

```
#define optical path difference (OPD) from mirror profile, i.e.
```

(continues on next page)

(continued from previous page)

```
#fill the struct opTrErMirr
#input:
#    mdatafile: an ascii file with mirror profile data
#    ncol:      number of columns in the file
#    delim:     delimiter between numbers in an row, can be space (' '), tab '\t', etc
#    Orient:    mirror orientation, 'x' (horizontal) or 'y' (vertical)
#    theta:     incidence angle
#    scale:     scaling factor for the mirror profile
def defineOPD(opTrErMirr, mdatafile, ncol, delim, Orient, theta, scale):
    heightProfData = np.loadtxt(mdatafile).T
    AuxTransmAddSurfHeightProfileScaled(opTrErMirr, heightProfData, Orient, theta,␣
↪scale)
    plt.figure()
    plot_1d(heightProfData,'profile from ' + mdatafile,'x (m)', 'h (m)') #todo add␣
↪the func def in on top of example
```

Defining initial wavefront and writing electric field data to h5-file

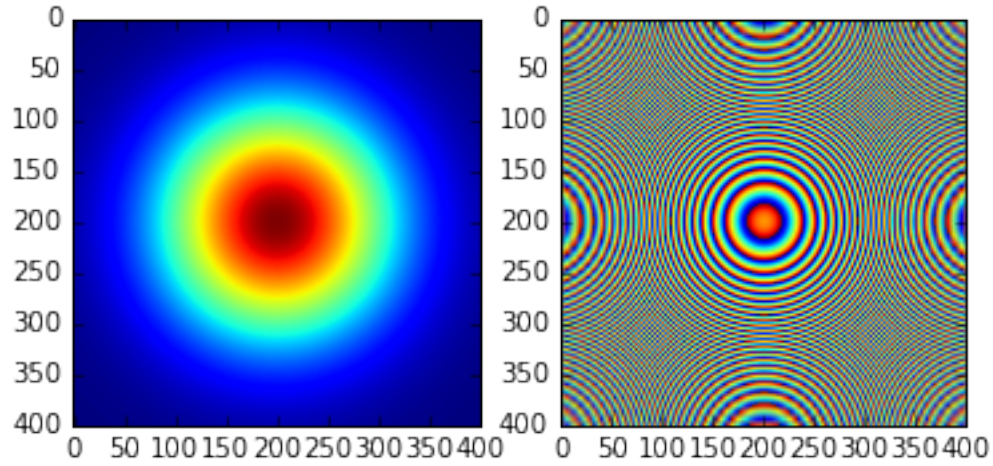
```
# #**********************Input Wavefront Structure and Parameters
print('*****defining initial wavefront and writing electric field data to h5-file...')
strInputDataFolder = 'data_common'  # input data sub-folder name
strOutputDataFolder = 'Tutorial_case_2'  # output data sub-folder name

#init Gauusian beam parameters
d2m1_sase1 = 246.5
d2m1_sase2 = 290.0
d2m1_sase3 = 281.0
d2hkb_sase1 = 904.0
d2hkb_sase3 = 442.3
dHKB_foc_sase3    = 2.715      # nominal focal length for HFM KB
dVKB_foc_sase3    = 1.715      # nominal focal length for VFM KB


qnC = 0.1                     # e-bunch charge, [nC]
pulse_duration = 9.e-15;

ekev_sase3 = 0.8;pulseEnergy_sase3 = 1.e-3; coh_time_sase_3 = 0.82e-15
thetaOM_sase3 = 9.e-3
thetaKB_sase3 = 9.e-3
ekev_sase1 = 8.0
thetaOM_sase1 = 2.5e-3       #
thetaKB_sase1 = 3.5e-3


ekev = ekev_sase3;pulseEnergy=pulseEnergy_sase3;coh_time=coh_time_sase_3
thetaOM = thetaOM_sase3
d2m1 = d2m1_sase3
d2hkb = d2hkb_sase3
thetaKB = thetaKB_sase3
dhkb_foc = dHKB_foc_sase3      # nominal focal length for HFM KB
dvkb_foc = dVKB_foc_sase3      # nominal focal length for VFM KB
dhkb_vkb = dhkb_foc - dvkb_foc          # distance between centers of HFM and VFM

z1 = d2m1
```

(continues on next page)

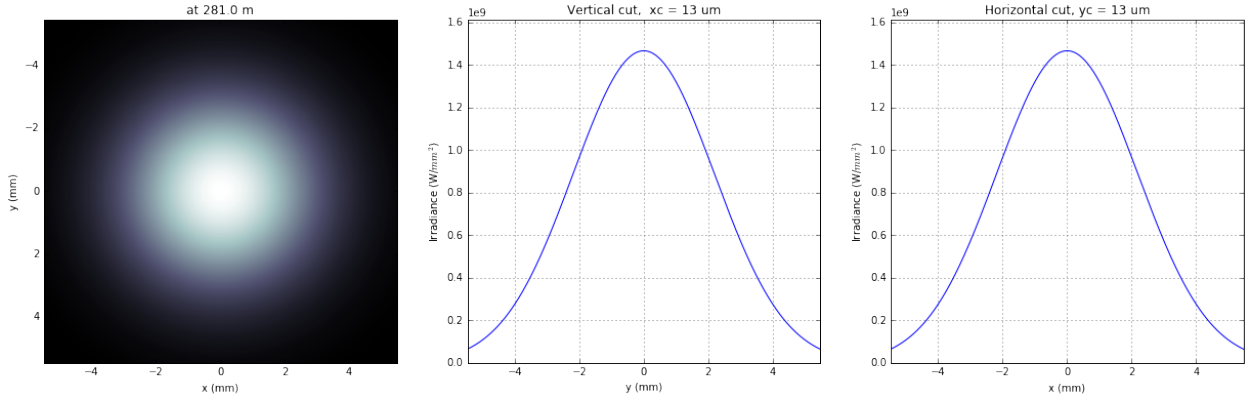
(continued from previous page)

```
theta_fwhm = calculate_theta_fwhm_cdr(ekev,qnC)
k = 2*np.sqrt(2*np.log(2))
sigX = 12.4e-10*k/(ekev*4*np.pi*theta_fwhm)
print('waist_fwhm [um], theta_fwhms [urad]:', sigX*k*1e6, theta_fwhm*1e6)
#define limits
range_xy = theta_fwhm/k*z1*5. # sigma*4 beam size
npoints=400

#define unique filename for storing results
ip = np.floor(ekev)
frac = np.floor((ekev - ip)*1e3)
fname0 = 'g' + str(int(ip))+'_'+str(int(frac))+'kev'
print('save hdf5: '+fname0+'.h5')
ifname = os.path.join(strOutputDataFolder,fname0+'.h5')

#build SRW gauusian wavefront
# wfr0=build_gauss_wavefront_xy(nx=np,ny=np,ekev=ekev,xMin=-range_xy/2,xMax=range_xy/
↪2,
#                               yMin=-range_xy/2,yMax=range_xy/2,sigX=sigX,sigY=sigX,
↪d2waist=z1)

wfr0 = build_gauss_wavefront_xy(npoints,npoints,ekev,-range_xy/2,range_xy/2,
                                -range_xy/2,range_xy/2,sigX,sigX,z1,
                                pulseEn=pulseEnergy,pulseTau=coh_time/np.sqrt(2),
                                repRate=1/(np.sqrt(2)*pulse_duration))


#init WPG Wavefront helper class
mwf = Wavefront(wfr0)

#store wavefront to HDF5 file
mwf.store_hdf5(ifname)

#draw wavefront with common functions
plt.subplot(1,2,1)
plt.imshow(mwf.get_intensity(slice_number=0))
plt.subplot(1,2,2)
plt.imshow(mwf.get_phase(slice_number=0,polarization='horizontal'))
plt.show()

#draw wavefront with cuts
plot_wfront(mwf, title_fig='at '+str(z1)+' m',
            isHlog=False, isVlog=False,
            i_x_min=1e-5, i_y_min=1e-5, orient='x', onePlot=True)

plt.set_cmap('bone') #set color map, 'bone', 'hot', 'jet', etc
fwhm_x = calculate_fwhm_x(mwf)
print('FWHMx [mm], theta_fwhm [urad]:',fwhm_x*1e3,fwhm_x/z1*1e6)
```

```
*****defining initial wavefront and writing electric field data to h5-file...
waist_fwhm [um], theta_fwhms [urad]: 37.2822729018 18.3457259238
save hdf5: g0_800kev.h5
```




```
FWHMx [mm]: 5.13005725474
FWHMy [mm]: 5.13005725474
Coordinates of center, [mm]: 0.0137167306277 0.0137167306277
stepX, stepY [um]: 27.433461255317237 27.433461255317237

Total power (integrated over full range): 43.1073 [GW]
Peak power calculated using FWHM:         43.9358 [GW]
Max irradiance: 1.46734 [GW/mm^2]
R-space
FWHMx [mm], theta_fwhm [urad]: 5.13005725474 18.2564315115
```



Defining optical beamline(s)

```
print('*****Defining optical beamline(s) ...')

z2 = d2hkb - d2m1

DriftM1_KB = SRWLOptD(z2) #Drift from first offset mirror (M1) to exp hall
horApM1 = 0.8*thetaOM
opApM1 = SRWLOptA('r', 'a', horApM1, range_xy)  # clear aperture of the Offset␣
↪Mirror(s)
horApKB = 0.8 * thetaKB # Aperture of the KB system, CA 0.8 m
opApKB = SRWLOptA('r', 'a', horApKB, horApKB)  # clear aperture of the Offset␣
↪Mirror(s)
```

(continues on next page)

(continued from previous page)

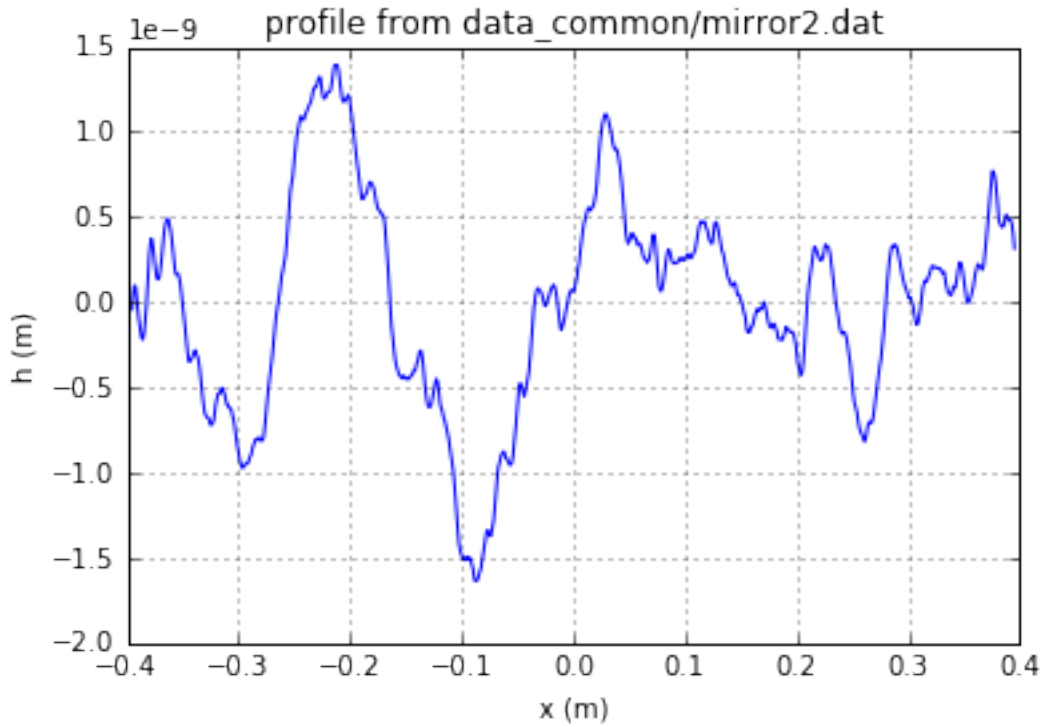
```
#Wavefront Propagation Parameters:
#[0]:  Auto-Resize (1) or not (0) Before propagation
#[1]:  Auto-Resize (1) or not (0) After propagation
#[2]:  Relative Precision for propagation with Auto-Resizing (1. is nominal)
#[3]:  Allow (1) or not (0) for semi-analytical treatment of quadratic phase terms at␣
↪propagation
#[4]:  Do any Resizing on Fourier side, using FFT, (1) or not (0)
#[5]:  Horizontal Range modification factor at Resizing (1. means no modification)
#[6]:  Horizontal Resolution modification factor at Resizing
#[7]:  Vertical Range modification factor at Resizing
#[8]:  Vertical Resolution modification factor at Resizing
#[9]:  Type of wavefront Shift before Resizing (not yet implemented)
#[10]: New Horizontal wavefront Center position after Shift (not yet implemented)
#[11]: New Vertical wavefront Center position after Shift (not yet implemented)
#                  [ 0] [1] [2]  [3] [4] [5]  [6]  [7]  [8]  [9] [10] [11]
ppM1 =             [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppTrErM1 =         [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDriftM1_KB =     [ 0,  0, 1.0,  1,  0, 2.4, 1.8, 2.4, 1.8,  0,  0,   0]
ppApKB =           [ 0,  0, 1.0,  0,  0, 0.6, 8.0, 0.6, 4.0,  0,  0,   0]
ppHKB =            [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppTrErHKB =        [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDrift_HKB_foc =  [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDrift_KB =       [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppVKB =            [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppTrErVKB =        [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDrift_foc =      [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
#ppFin  =           [ 0,  0, 1.0,  0,  0, 0.05,5.0, 0.05,5.0,  0,  0,   0]
#ppFin =            [ 0,  0, 1.0,  0,  1, .01, 20.0, .01, 20.0,  0,  0,   0]
ppFin =            [ 0,  0, 1.0,  0,  1, .02, 10.0, .02, 10.0,  0,  0,   0]

optBL0 = SRWLOptC([opApM1,  DriftM1_KB],
                  [ppM1,ppDriftM1_KB])

scale = 2     #5 mirror profile scaling factor
print('*****HOM1 data for BL1 beamline ')
opTrErM1 = SRWLOptT(1500, 100, horApM1, range_xy)
#defineOPD(opTrErM1, os.path.join(strInputDataFolder,'mirror1.dat'), 2, '\t', 'x',␣
↪thetaOM, scale)
defineOPD(opTrErM1, os.path.join(strInputDataFolder,'mirror2.dat'), 2, ' ', 'x',␣
↪thetaOM, scale)
opdTmp=np.array(opTrErM1.arTr)[1::2].reshape(opTrErM1.mesh.ny,opTrErM1.mesh.nx)
plt.figure()
plot_2d(opdTmp, opTrErM1.mesh.xStart*1e3,opTrErM1.mesh.xFin*1e3,opTrErM1.mesh.
↪yStart*1e3,opTrErM1.mesh.yFin*1e3,
        'OPD [m]', 'x (mm)', 'y (mm)')

optBL1 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB],
                  [ppM1,ppTrErM1,ppDriftM1_KB])

dhkb_vkb = dhkb_foc - dvkb_foc          # distance between centers of HFM and VFM
d2vkb = d2hkb +  dhkb_vkb
vkbfoc =  1. /(1./dvkb_foc + 1. / d2vkb) # for thin lens approx
hkbfoc =  1. /(1./dhkb_foc + 1. / d2hkb) # for thin lens approx

z3 = dhkb_vkb
z4 = vkbfoc #distance to focal plane
```

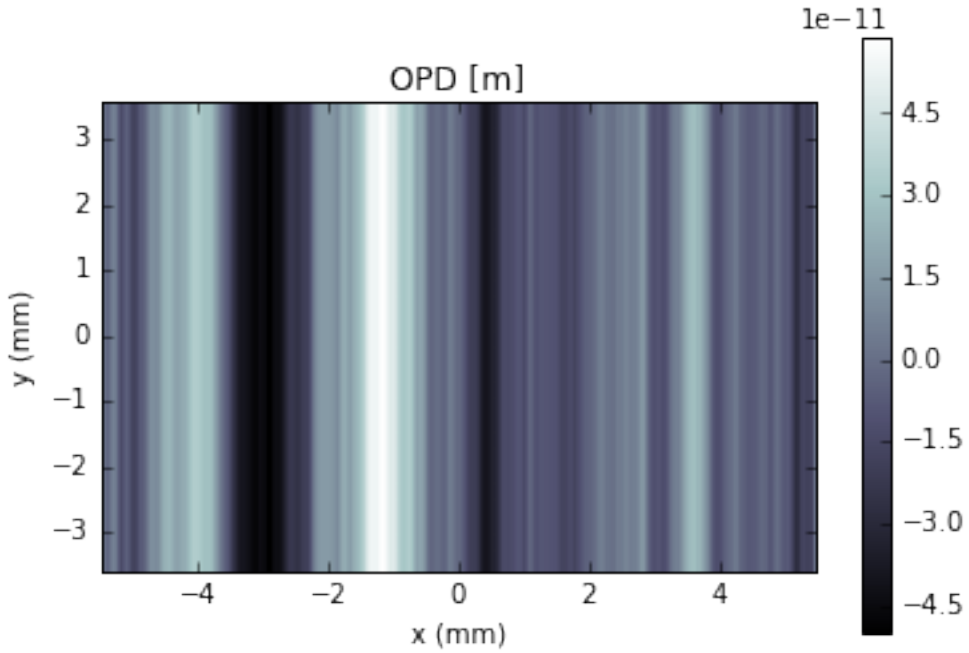
(continues on next page)

(continued from previous page)

```
# HKB = SRWLOptMirEl(_p=d2hkb, _q=dhkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85,
#                    _nvx=cos(thetaKB), _nvy=0, _nvz=-sin(thetaKB), _tvx=-
↪sin(thetaKB), _tvy=0,
#                    _x=0, _y=0, _treat_in_out=1) #HKB Ellipsoidal Mirror
# VKB = SRWLOptMirEl(_p=d2vkb, _q=dvkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85,
#                    _nvx=0, _nvy=cos(thetaKB), _nvz=-sin(thetaKB), _tvx=0, _tvy=-
↪sin(thetaKB),
#                    _x=0, _y=0, _treat_in_out=1) #VKB Ellipsoidal Mirror

HKB = SRWLOptL(hkbfoc) #HKB as Thin Lens
VKB = SRWLOptL(1e23,vkbfoc) #VKB as Thin Lens
Drift_KB  = SRWLOptD(z3)
Drift_foc = SRWLOptD(z4)
optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,  Drift_
↪foc],
                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,ppDrift_
↪foc,ppFin])
```

```
*****Defining optical beamline(s) ...
*****HOM1 data for BL1 beamline
```





Propagating through BL0 beamline. Ideal mirror: HOM as an aperture

```
print('*****Ideal mirror: HOM as an aperture')
bPlotted = False
isHlog = False
isVlog = False
bSaved = True
optBL = optBL0
strBL = 'bl0'
pos_title = 'at exp hall wall'
print('*****setting-up optical elements, beamline:', strBL)
print_beamline(optBL)
startTime = time.time()

print('*****reading wavefront from h5 file...')
w2 = Wavefront()
w2.load_hdf5(ifname)
wfr = w2._srwl_wf

print('*****propagating wavefront (with resizing)...')
srwl.PropagElecField(wfr, optBL)
mwf = Wavefront(wfr)
print('[nx, ny, xmin, xmax, ymin, ymax]', get_mesh(mwf))
if bSaved:
    print('save hdf5:', fname0+'_'+strBL+'.h5')
    mwf.store_hdf5(os.path.join(strOutputDataFolder, fname0+'_'+strBL+'.h5'))
print('done')
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Ideal mirror: HOM as an aperture
*****setting-up optical elements, beamline: bl0
Optical Element: Aperture / Obstacle
```

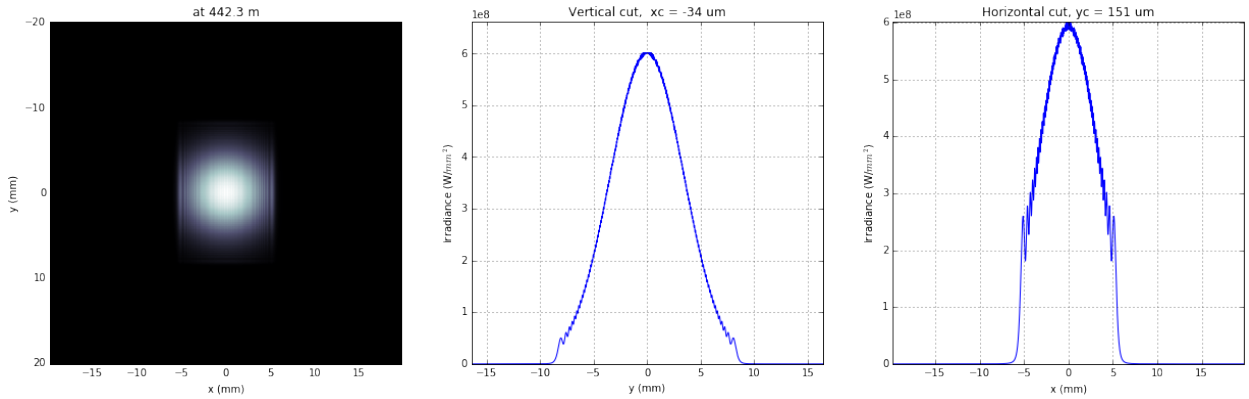
(continues on next page)

(continued from previous page)

```
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0109459510409
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 161.3
    treat = 0


*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
[nx, ny, xmin, xmax, ymin, ymax] [1728, 1728, -0.019740897807083827, 0.
↪019740897807083834, -0.02015430433530526, 0.020154304335305268]
save hdf5: g0_800kev_bl0.h5
done
propagation lasted: 0.1 min
```

```
print('*****Ideal mirror: HOM as an aperture')
plot_wfront(mwf, 'at '+str(z1+z2)+' m',False, False, 1e-5,1e-5,'x', True)
plt.set_cmap('bone') #set color map, 'bone', 'hot', 'jet', etc
plt.axis('tight')
print('FWHMx [mm], theta_fwhm [urad]:',calculate_fwhm_x(mwf)*1e3,calculate_fwhm_
↪x(mwf)/(z1+z2)*1e6)
print('FWHMy [mm], theta_fwhm [urad]:',calculate_fwhm_y(mwf)*1e3,calculate_fwhm_
↪y(mwf)/(z1+z2)*1e6)
```

```
*****Ideal mirror: HOM as an aperture
FWHMx [mm]: 8.5730592677
FWHMy [mm]: 8.1457466277
Coordinates of center, [mm]: -0.0342922370708 0.15171161341
stepX, stepY [um]: 22.86149138052557 23.34024821691403

Total power (integrated over full range): 39.2445 [GW]
Peak power calculated using FWHM:         47.8113 [GW]
Max irradiance: 0.601754 [GW/mm^2]
R-space
FWHMx [mm], theta_fwhm [urad]: 8.5730592677 19.3829058732
FWHMy [mm], theta_fwhm [urad]: 8.1457466277 18.4167909286
```



Propagating through BL1 beamline. Imperfect mirror, at KB aperture

```
print ('*****Imperfect mirror, at KB aperture')
bPlotted = False
isHlog = True
isVlog = False
bSaved = False
optBL = optBL1
strBL = 'bl1'
pos_title = 'at exp hall wall'
print('*****setting-up optical elements, beamline:', strBL)
print_beamline(optBL)
startTime = time.time()
print('*****reading wavefront from h5 file...')
w2 = Wavefront()
w2.load_hdf5(ifname)
wfr = w2._srwl_wf
print('*****propagating wavefront (with resizing)...')
srwl.PropagElecField(wfr, optBL)
mwf = Wavefront(wfr)
print('[nx, ny, xmin, xmax, ymin, ymax]', get_mesh(mwf))
if bSaved:
    print('save hdf5:', fname0+'_'+strBL+'.h5')
    mwf.store_hdf5(os.path.join(strOutputDataFolder,fname0+'_'+strBL+'.h5'))
print('done')
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Imperfect mirror, at KB aperture
*****setting-up optical elements, beamline: bl1
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0109459510409
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
```

(continues on next page)

(continued from previous page)

```
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1500
             ny = 100
             xFin = 0.0036
             xStart = -0.0036
             yFin = 0.00547297552044
             yStart = -0.00547297552044
             zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 161.3
    treat = 0


*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
[nx, ny, xmin, xmax, ymin, ymax] [1728, 1728, -0.01974162347178916, 0.
↪019741623471789167, -0.02015430433530526, 0.020154304335305268]
done
propagation lasted: 0.1 min
```

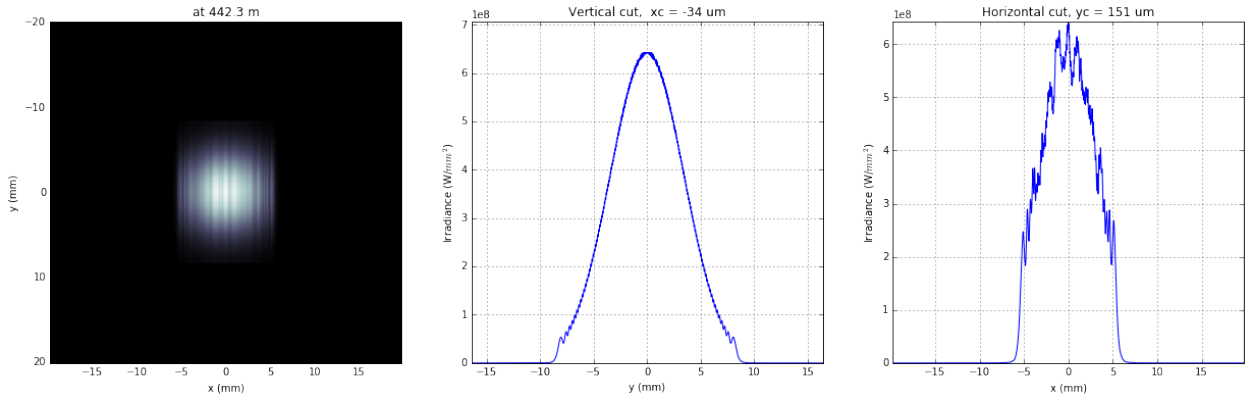
```
print ('*****Imperfect mirror, at KB aperture')
plot_wfront(mwf, 'at '+str(z1+z2)+' m',False, False, 1e-5,1e-5,'x', True)
plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [mm], theta_fwhm [urad]:',calculate_fwhm_x(mwf)*1e3,calculate_fwhm_
↪x(mwf)/(z1+z2)*1e6)
print('FWHMy [mm], theta_fwhm [urad]:',calculate_fwhm_y(mwf)*1e3,calculate_fwhm_
↪y(mwf)/(z1+z2)*1e6)
```

```
*****Imperfect mirror, at KB aperture
FWHMx [mm]: 7.93322911953
FWHMy [mm]: 8.1457466277
Coordinates of center, [mm]: -0.0342934976348 0.15171161341
stepX, stepY [um]: 22.86233175655954 23.34024821691403

Total power (integrated over full range): 39.2445 [GW]
Peak power calculated using FWHM:         47.2669 [GW]
Max irradiance: 0.642883 [GW/mm^2]
R-space
FWHMx [mm], theta_fwhm [urad]: 7.93322911953 17.936308206
```

(continues on next page)

(continued from previous page)

```
FWHMy [mm], theta_fwhm [urad]: 8.1457466277 18.4167909286
```



Propagating through BL2 beamline. Focused beam: perfect KB

```
print('*****Focused beam: perfect KB')
#optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,
↪Drift_foc],
#                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,
↪ppDrift_foc])
z3 = dhkb_vkb
z4 = vkbfoc #distance to focal plane

HKB = SRWLOptMirEl(_p=d2hkb, _q=dhkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85,
                   _nvx=np.cos(thetaKB), _nvy=0, _nvz=-np.sin(thetaKB),
                   _tvx=-np.sin(thetaKB), _tvy=0, _x=0, _y=0, _treat_in_out=1) #HKB
↪Ellipsoidal Mirror
VKB = SRWLOptMirEl(_p=d2vkb, _q=dvkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85,
                   _nvx=0, _nvy=np.cos(thetaKB), _nvz=-np.sin(thetaKB),
                   _tvx=0, _tvy=-np.sin(thetaKB), _x=0, _y=0, _treat_in_out=1) #VKB
↪Ellipsoidal Mirror
#HKB = SRWLOptL(hkbfoc) #HKB as Thin Lens
#VKB = SRWLOptL(1e23,vkbfoc) #VKB as Thin Lens
Drift_foc = SRWLOptD(dvkb_foc)
optBL2 = SRWLOptC([opApM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,  Drift_foc],
                  [ppM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,ppDrift_foc,
↪ppFin])
optBL = optBL2
strBL = 'bl2'
pos_title = 'at sample position'
print('*****setting-up optical elements, beamline:', strBL)
print_beamline(optBL)
startTime = time.time()
print('*****reading wavefront from h5 file...')
w2 = Wavefront()
w2.load_hdf5(ifname)
wfr = w2._srwl_wf
print('*****propagating wavefront (with resizing)...')
srwl.PropagElecField(wfr, optBL)
```

(continues on next page)

(continued from previous page)

```
mwf = Wavefront(wfr)
print('[nx, ny, xmin, xmax, ymin, ymax]', get_mesh(mwf))
if bSaved:
    print('save hdf5:', fname0+'_'+strBL+'.h5')
    mwf.store_hdf5(os.path.join(strOutputDataFolder,fname0+'_'+strBL+'.h5'))
print('done')
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Focused beam: perfect KB
*****setting-up optical elements, beamline: bl2
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0109459510409
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 161.3
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0072
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.009
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0.999959500273
    nvy = 0
    nvz = -0.00899987850049
    p = 442.3
    q = 2.715
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
```

(continues on next page)

(continued from previous page)

```
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = -0.00899987850049
    tvy = 0
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 0.9999999999999998
    treat = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.009
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0
    nvy = 0.999959500273
    nvz = -0.00899987850049
    p = 443.3
    q = 1.715
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = 0
    tvy = -0.00899987850049
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.715
    treat = 0
```

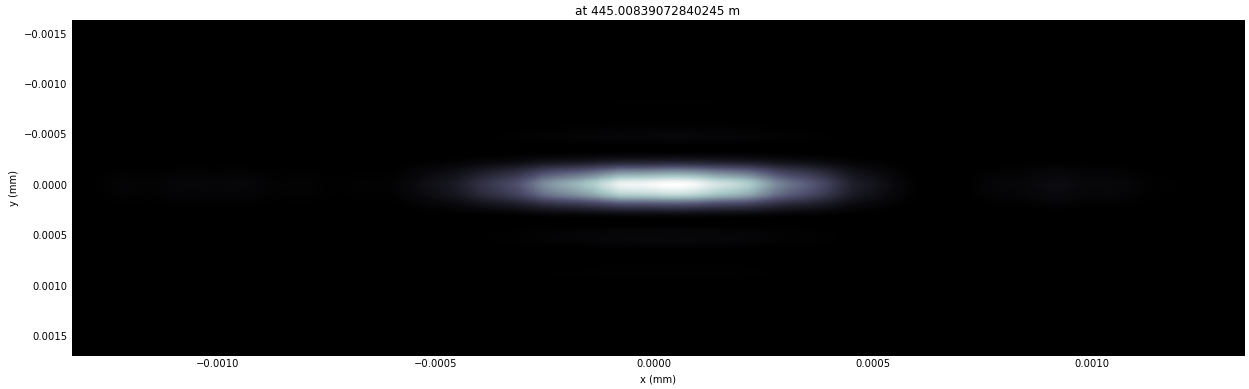
(continues on next page)

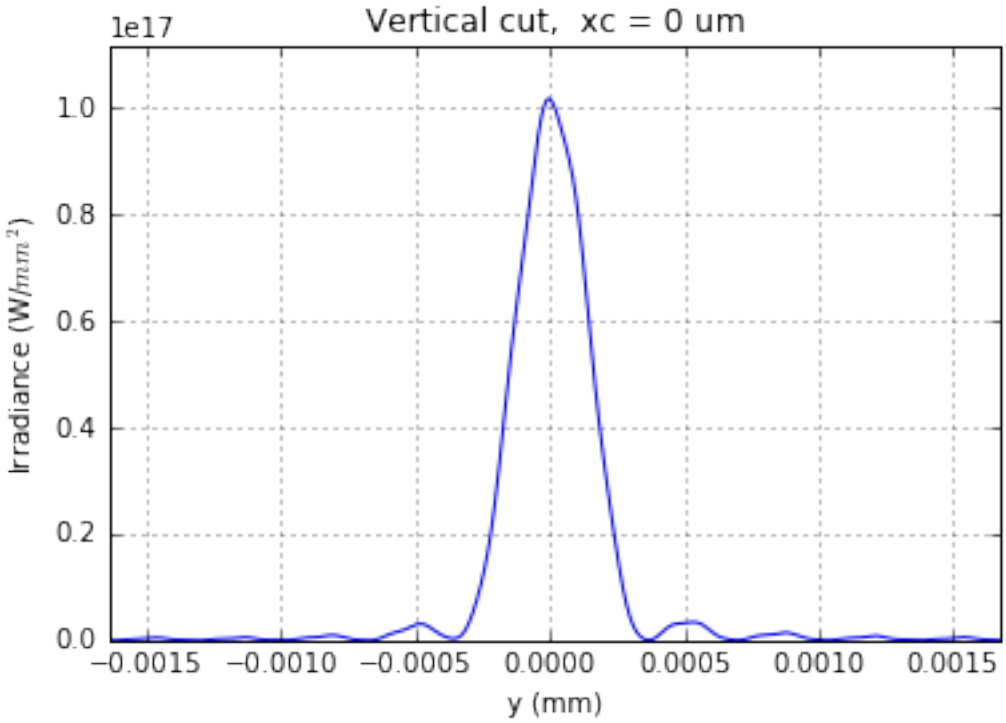
(continued from previous page)

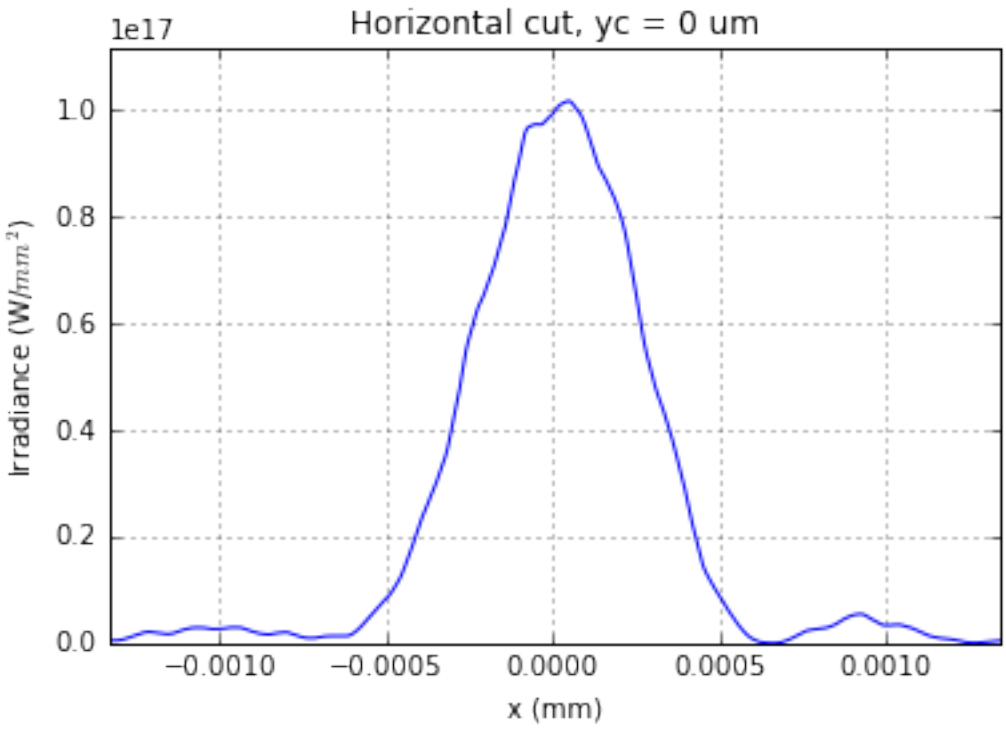
```
Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 1, 0.02, 10.0, 0.02, 10.0, 0, 0, 0]


*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
[nx, ny, xmin, xmax, ymin, ymax] [1664, 832, -1.3331119492844142e-06, 1.
↪347755378255926e-06, -1.6419426896857787e-06, 1.67847891996219e-06]
done
propagation lasted: 1.9 min
```

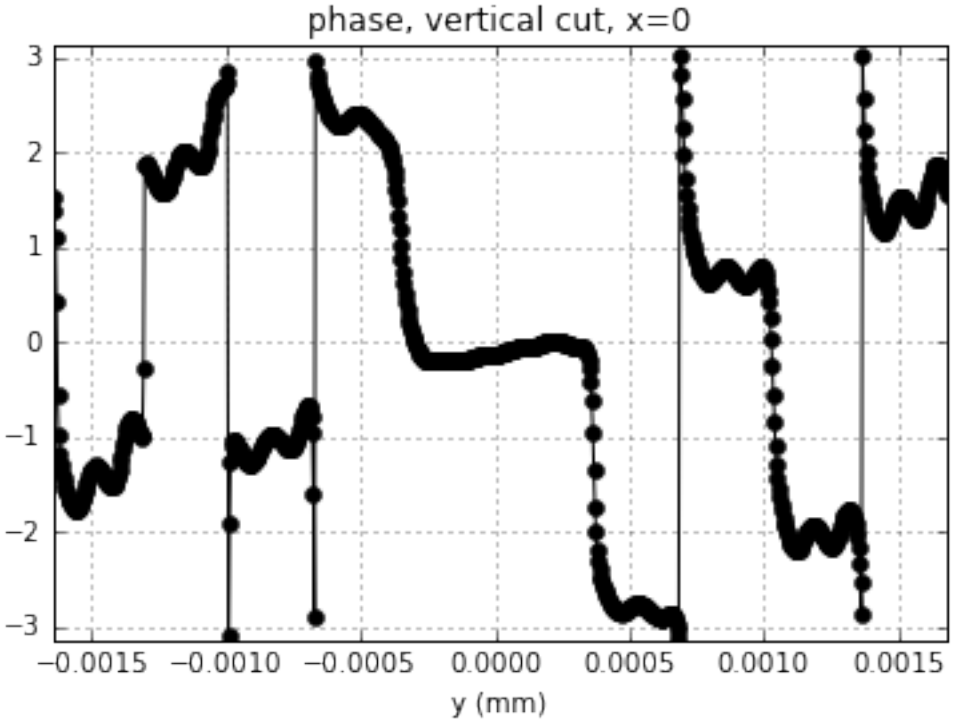
```
print ('*****Focused beam: Focused beam: perfect KB')
bOnePlot = True
isHlog = True
isVlog = True
bSaved = False
#plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m',isHlog, isVlog, 1e-6,1e-6,'x', bOnePlot)
dd0_v = plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m', False,  False,1e-6,1e-6, 'y',␣
↪False,True)
dd0_h = plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m',isHlog, isVlog,1e-6,1e-6, 'x',␣
↪bOnePlot)
plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [um], FWHMy [um]:',calculate_fwhm_x(mwf)*1e6,calculate_fwhm_y(mwf)*1e6)
```

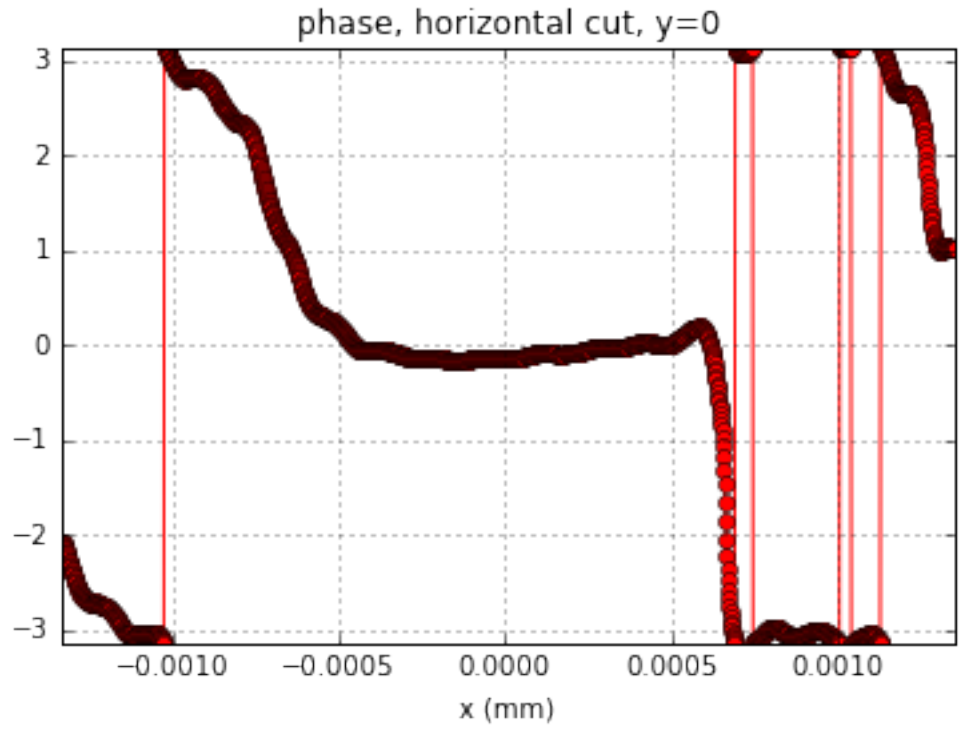
```
*****Focused beam: Focused beam: perfect KB
FWHMx [mm]: 0.000565835497274
FWHMy [mm]: 0.000311664122205
Coordinates of center, [mm]: 4.84294215527e-05 -3.70820117114e-06
stepX, stepY [um]: 0.0016120669438005656 0.003995693874425955

Total power (integrated over full range): 20.9969 [GW]
Peak power calculated using FWHM:         20.3889 [GW]
Max irradiance: 1.01619e+08 [GW/mm^2]
R-space
FWHMx[um]: 0.565835497274
FWHMy [um]: 0.311664122205
Coordinates of center, [mm]: 4.84294215527e-05 -3.70820117114e-06
stepX, stepY [um]: 0.0016120669438005656 0.003995693874425955

Total power (integrated over full range): 20.9969 [GW]
Peak power calculated using FWHM:         20.3889 [GW]
Max irradiance: 1.01619e+08 [GW/mm^2]
R-space
FWHMx [um], FWHMy [um]: 0.565835497274 0.311664122205
```

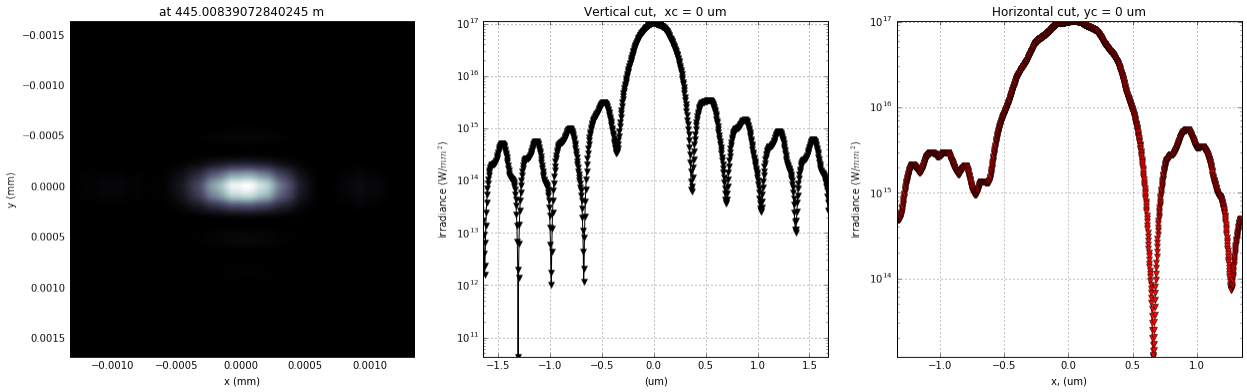

at 445.00839072840245 m
−0.0015
−0.0010
−0.0005
0.0000
0.0005
0.0010
0.0015
y (mm)
−0.0010
−0.0005
0.0000
0.0005
0.0010
x (mm)


Vertical cut, xc = 0 um
1e17
1.0
0.8
0.6
0.4
0.2
0.0
Irradiance (W/mm²)
−0.0015
−0.0010
−0.0005
0.0000
0.0005
0.0010
0.0015
y (mm)


Horizontal cut, yc = 0 um
1e17
Irradiance (W/mm²)
1.0
0.8
0.6
0.4
0.2
0.0
−0.0010
−0.0005
0.0000
0.0005
0.0010
x (mm)


phase, vertical cut, x=0
3
2
1
0
−1
−2
−3
−0.0015
−0.0010
−0.0005
0.0000
0.0005
0.0010
0.0015
y (mm)





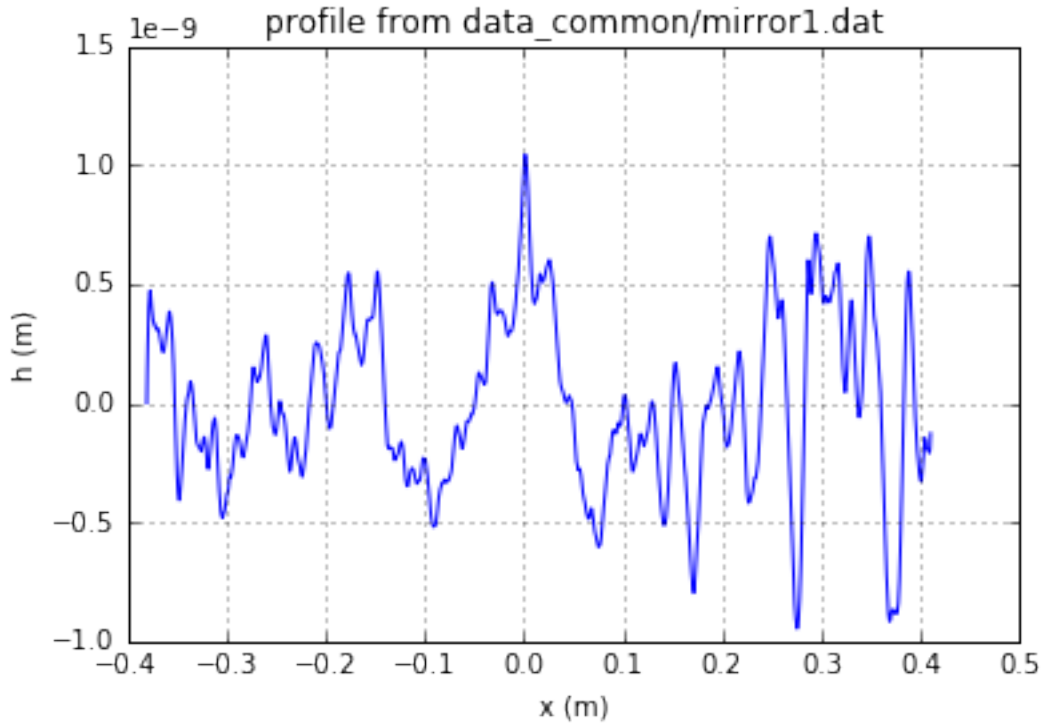

```
scaleKB = 8
opTrErHKB = SRWLOptT(1500, 100, horApKB, horApKB)
defineOPD(opTrErHKB, os.path.join(strInputDataFolder,'mirror1.dat'), 2, '\t', 'x', ␣
↪thetaKB, scaleKB)
opdTmp=np.array(opTrErHKB.arTr)[1::2].reshape(opTrErHKB.mesh.ny,opTrErHKB.mesh.nx)
print('*****HKB data  ')
plt.figure()
#subplot()
plot_2d(opdTmp, opTrErHKB.mesh.xStart*1e3,opTrErHKB.mesh.xFin*1e3,opTrErHKB.mesh.
↪yStart*1e3,opTrErHKB.mesh.yFin*1e3,
        'OPD [m]', 'x (mm)', 'y (mm)')
print('*****VKB data  ')
opTrErVKB = SRWLOptT(100, 1500, horApKB, horApKB)
defineOPD(opTrErVKB, os.path.join(strInputDataFolder,'mirror2.dat'), 2, ' ', 'y', ␣
↪thetaKB, scaleKB)
opdTmp=np.array(opTrErVKB.arTr)[1::2].reshape(opTrErVKB.mesh.ny,opTrErVKB.mesh.nx)
#subplot()
```

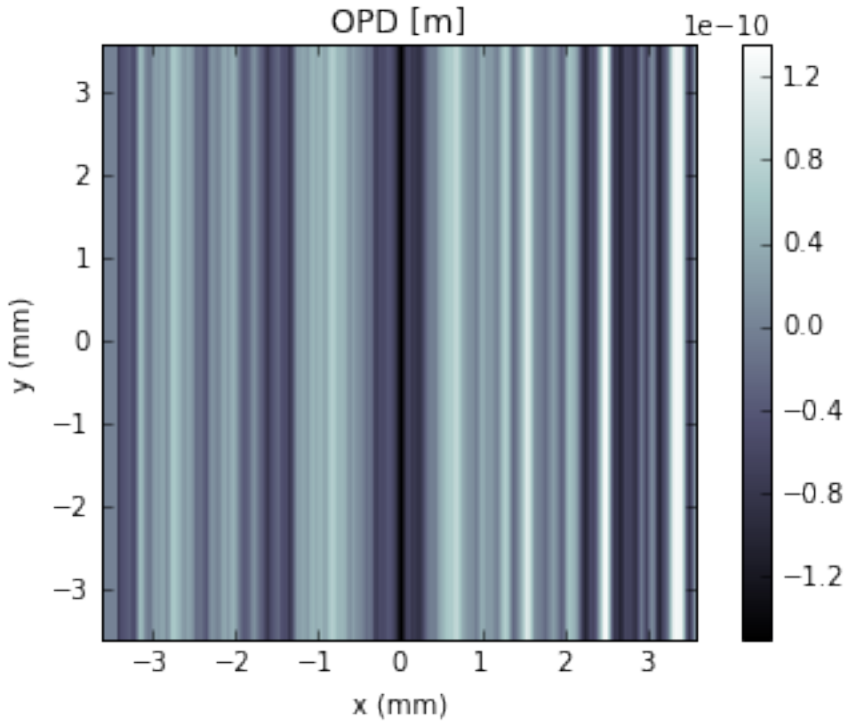
(continues on next page)

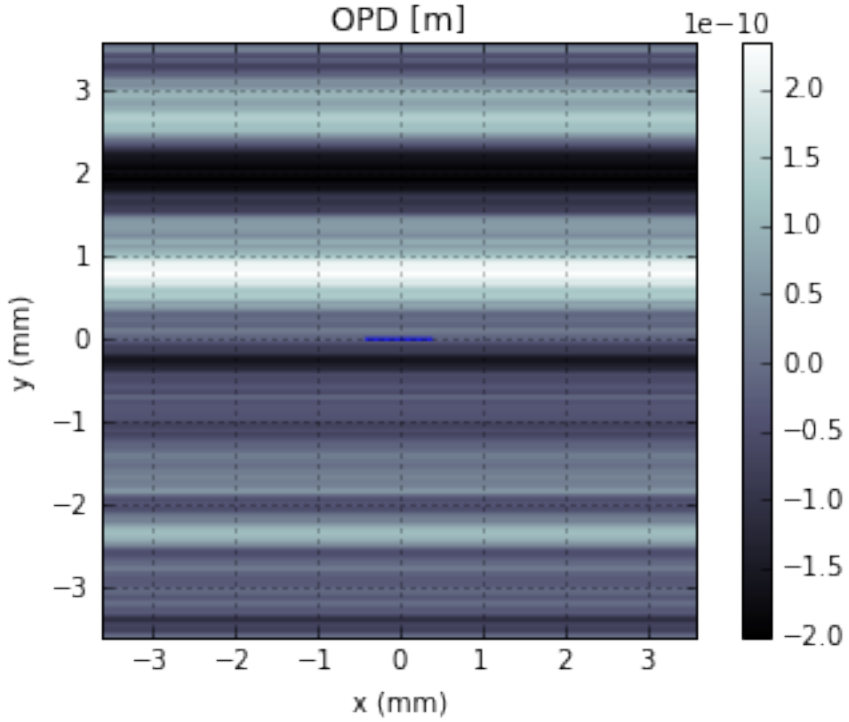
(continued from previous page)

```
plot_2d(opdTmp, opTrErVKB.mesh.xStart*1e3,opTrErVKB.mesh.xFin*1e3,opTrErVKB.mesh.
↪yStart*1e3,opTrErVKB.mesh.yFin*1e3,
        'OPD [m]', 'x (mm)', 'y (mm)')
print(vkbfoc-dvkb_foc)
```

```
*****HKB data
*****VKB data
-0.006609271597586286
```








```
print('*****Focused beam: non-perfect KB')
#optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,
↪Drift_foc],
#                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,
↪ppDrift_foc])
```

(continues on next page)

(continued from previous page)

```
z3 = dhkb_vkb
z4 = dvkb_foc #distance to focal plane
#z4 = vkbfoc

HKB = SRWLOptMirEl(_p=d2hkb, _q=dhkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85,
                   _nvx=np.cos(thetaKB), _nvy=0, _nvz=-np.sin(thetaKB),
                   _tvx=-np.sin(thetaKB), _tvy=0, _x=0, _y=0, _treat_in_out=1) #HKB␣
↪Ellipsoidal Mirror
VKB = SRWLOptMirEl(_p=d2vkb, _q=dvkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85,
                   _nvx=0, _nvy=np.cos(thetaKB), _nvz=-np.sin(thetaKB), _tvx=0, _tvy=-
↪np.sin(thetaKB),
                   _x=0, _y=0, _treat_in_out=1) #VKB Ellipsoidal Mirror
#HKB = SRWLOptL(hkbfoc) #HKB as Thin Lens
#VKB = SRWLOptL(1e23,vkbfoc) #VKB as Thin Lens
Drift_foc = SRWLOptD(z4)
optBL2 = SRWLOptC([opApM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,  Drift_foc],
                  [ppM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,ppDrift_foc,
↪ppFin])
optBL3 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,opTrErHKB,  Drift_KB, ␣
↪VKB,opTrErVKB,  Drift_foc],
                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppTrErM1,ppDrift_KB,
↪ppVKB,ppTrErM1, ppDrift_foc,ppFin])
optBL = optBL3
strBL = 'bl3'
pos_title = 'at sample position'
print('*****setting-up optical elements, beamline:', strBL)
print_beamline(optBL)
startTime = time.time()
print('*****reading wavefront from h5 file...')
w2 = Wavefront()
w2.load_hdf5(ifname)
wfr = w2._srwl_wf
print('*****propagating wavefront (with resizing)...')
srwl.PropagElecField(wfr, optBL)
mwf = Wavefront(wfr)
print('[nx, ny, xmin, xmax, ymin, ymax]', get_mesh(mwf))
if bSaved:
    print('save hdf5:', fname0+'_'+strBL+'.h5')
    mwf.store_hdf5(os.path.join(strOutputDataFolder,fname0+'_'+strBL+'.h5'))
print('done')
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Focused beam: non-perfect KB
*****setting-up optical elements, beamline: bl3
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0109459510409
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
```

(continues on next page)

(continued from previous page)

```
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0036
            xStart = -0.0036
            yFin = 0.00547297552044
            yStart = -0.00547297552044
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 161.3
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0072
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.009
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0.999959500273
    nvy = 0
    nvz = -0.00899987850049
    p = 442.3
```

(continues on next page)

(continued from previous page)

```
    q = 2.715
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = -0.00899987850049
    tvy = 0
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0036
            xStart = -0.0036
            yFin = 0.0036
            yStart = -0.0036
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 0.9999999999999998
    treat = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.009
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
```

(continues on next page)

(continued from previous page)

```
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0
    nvy = 0.999959500273
    nvz = -0.00899987850049
    p = 443.3
    q = 1.715
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = 0
    tvy = -0.00899987850049
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 100
            ny = 1500
            xFin = 0.0036
            xStart = -0.0036
            yFin = 0.0036
            yStart = -0.0036
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.715
    treat = 0
```

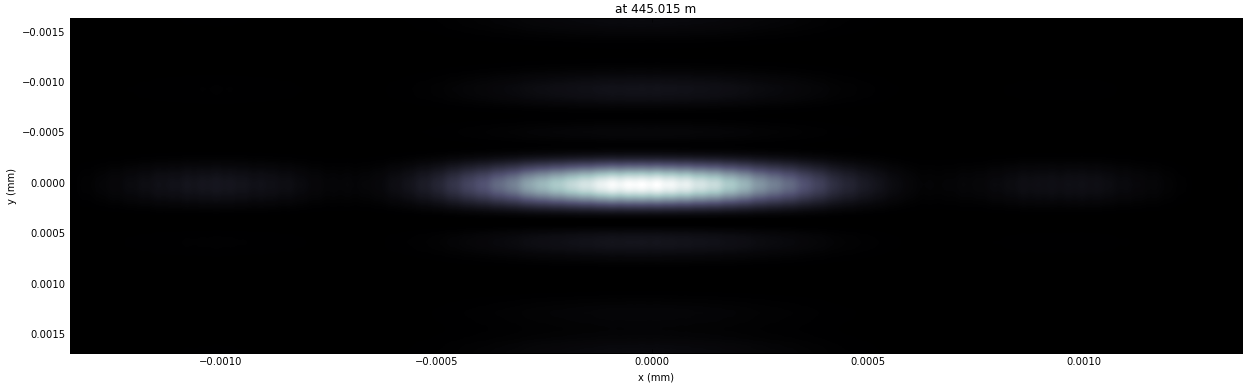
(continues on next page)

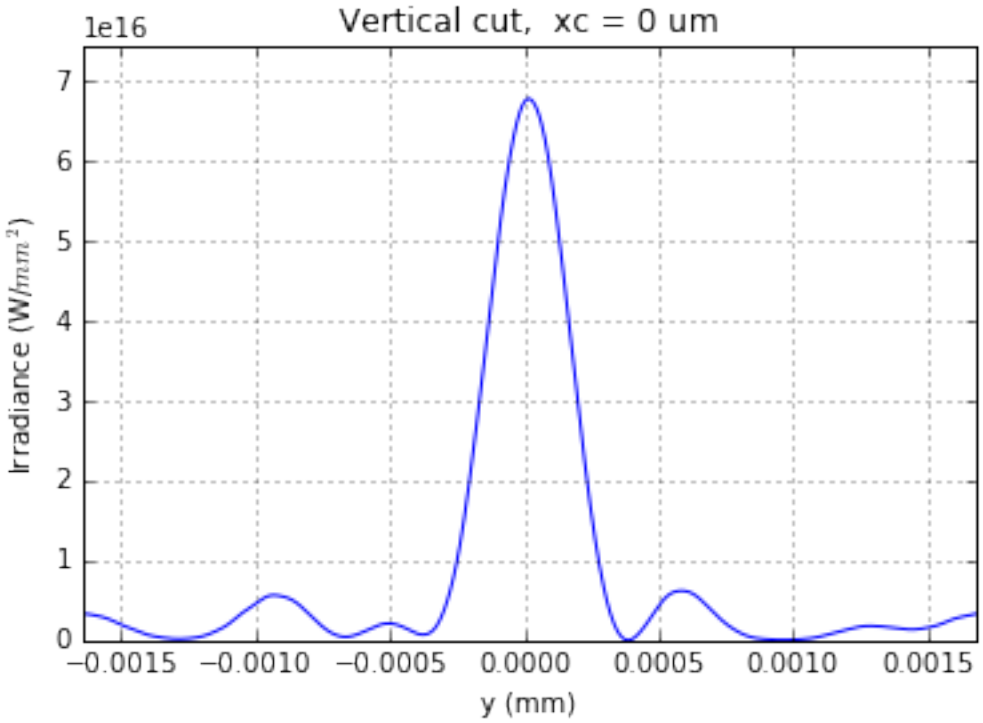
(continued from previous page)

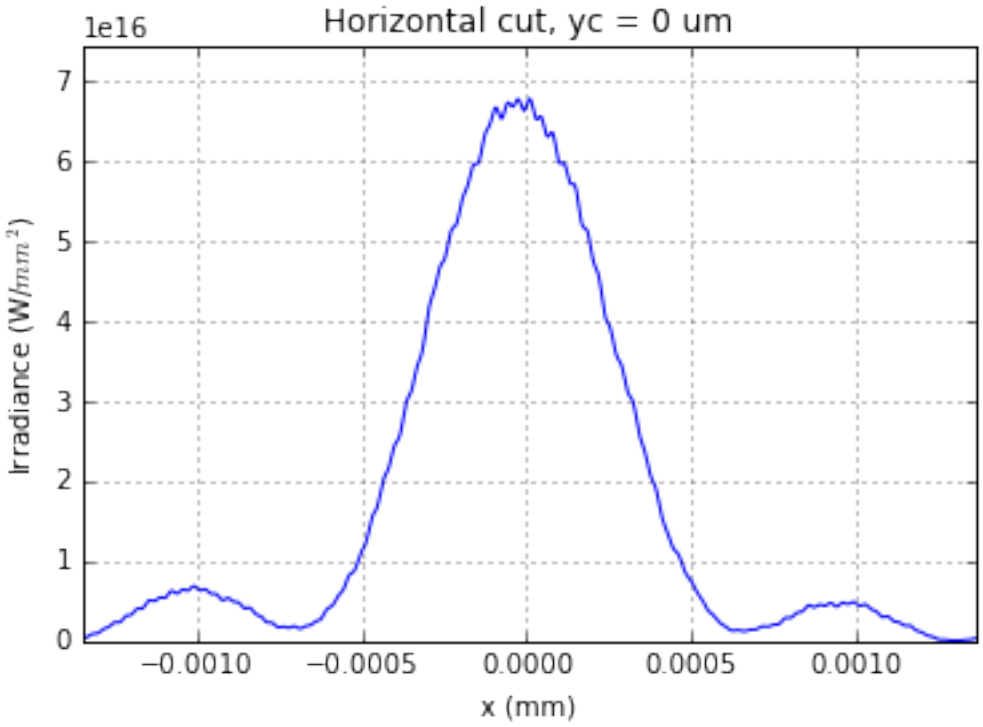
```
Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 1, 0.02, 10.0, 0.02, 10.0, 0, 0, 0]


*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
[nx, ny, xmin, xmax, ymin, ymax] [1664, 832, -1.3491024746288286e-06, 1.
↪3639215498558462e-06, -1.6419426740561654e-06, 1.678478903984771e-06]
done
propagation lasted: 1.8 min
```

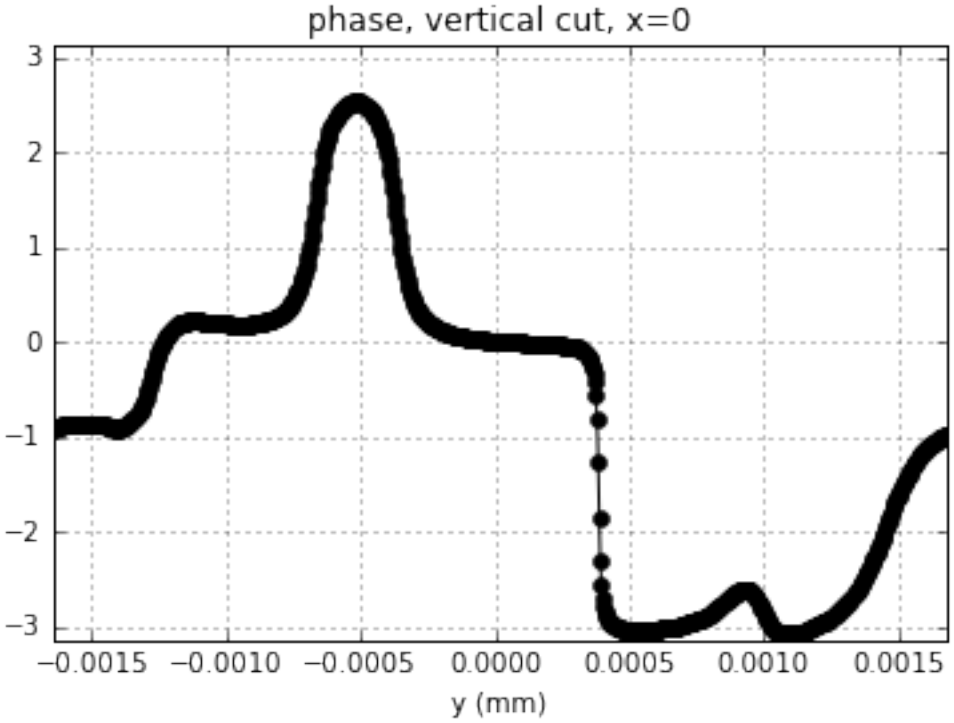
```
print ('*****Focused beam: Focused beam: non-perfect KB')
bOnePlot = True
isHlog = True
isVlog = True
bSaved = False
#plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m',isHlog, isVlog, 1e-6,1e-6,'x', bOnePlot)
dd1_v = plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m', False,  False,1e-6,1e-6, 'y',␣
↪False,True)
dd1_h = plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m',isHlog, isVlog,1e-6,1e-6, 'x',␣
↪bOnePlot)
plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [um], FWHMy [um]:',calculate_fwhm_x(mwf)*1e6,calculate_fwhm_y(mwf)*1e6)
```

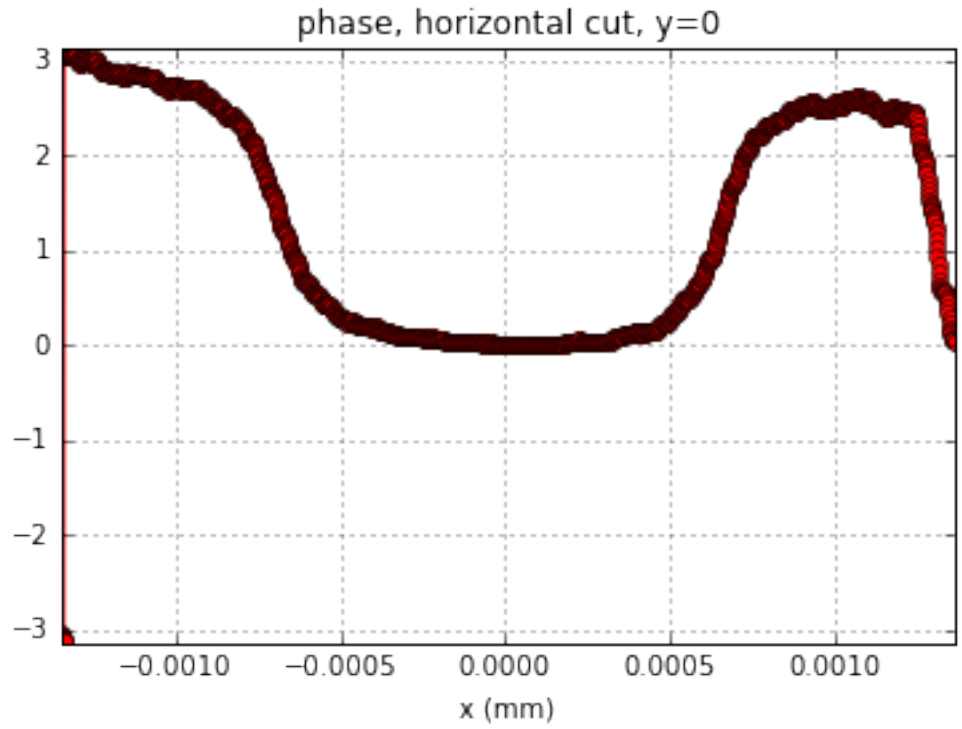
```
*****Focused beam: Focused beam: non-perfect KB
FWHMx [mm]: 0.00062645894492
FWHMy [mm]: 0.000335638282257
Coordinates of center, [mm]: 8.22523936471e-06 1.62702680461e-05
stepX, stepY [um]: 0.0016314035023960764 0.003995693836391019

Total power (integrated over full range): 20.2694 [GW]
Peak power calculated using FWHM:         16.2036 [GW]
Max irradiance: 6.77334e+07 [GW/mm^2]
R-space
FWHMx[um]: 0.62645894492
FWHMy [um]: 0.335638282257
Coordinates of center, [mm]: 8.22523936471e-06 1.62702680461e-05
stepX, stepY [um]: 0.0016314035023960764 0.003995693836391019

Total power (integrated over full range): 20.2694 [GW]
Peak power calculated using FWHM:         16.2036 [GW]
Max irradiance: 6.77334e+07 [GW/mm^2]
R-space
FWHMx [um], FWHMy [um]: 0.62645894492 0.335638282257
```

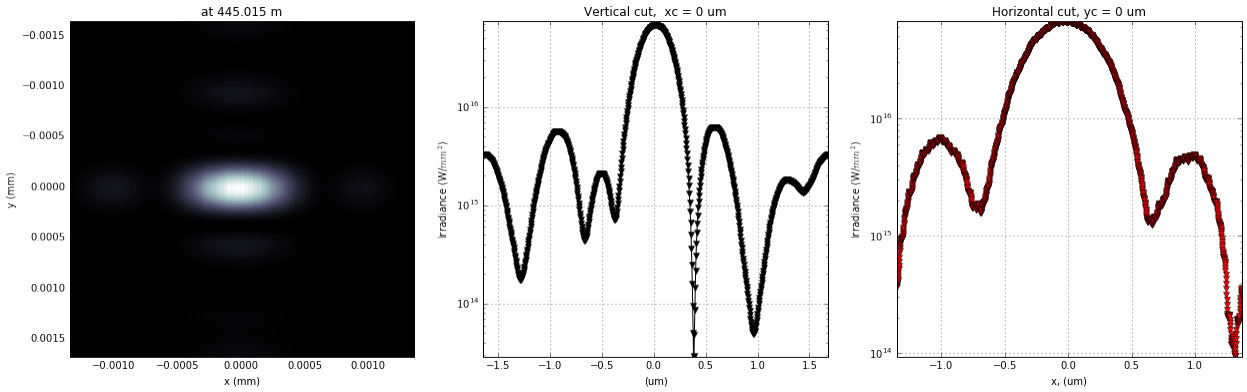

at 445.015 m
−0.0015
−0.0010
−0.0005
0.0000
0.0005
0.0010
0.0015
y (mm)
−0.0010
−0.0005
0.0000
0.0005
0.0010
x (mm)


Vertical cut, xc = 0 um
1e16
Irradiance (W/mm²)
0
1
2
3
4
5
6
7
−0.0015
−0.0010
−0.0005
0.0000
0.0005
0.0010
0.0015
y (mm)


Horizontal cut, yc = 0 um
1e16
Irradiance (W/mm²)
x (mm)
0
1
2
3
4
5
6
7
−0.0010
−0.0005
0.0000
0.0005
0.0010


phase, vertical cut, x=0
y (mm)
−3
−2
−1
0
1
2
3
−0.0015
−0.0010
−0.0005
0.0000
0.0005
0.0010
0.0015



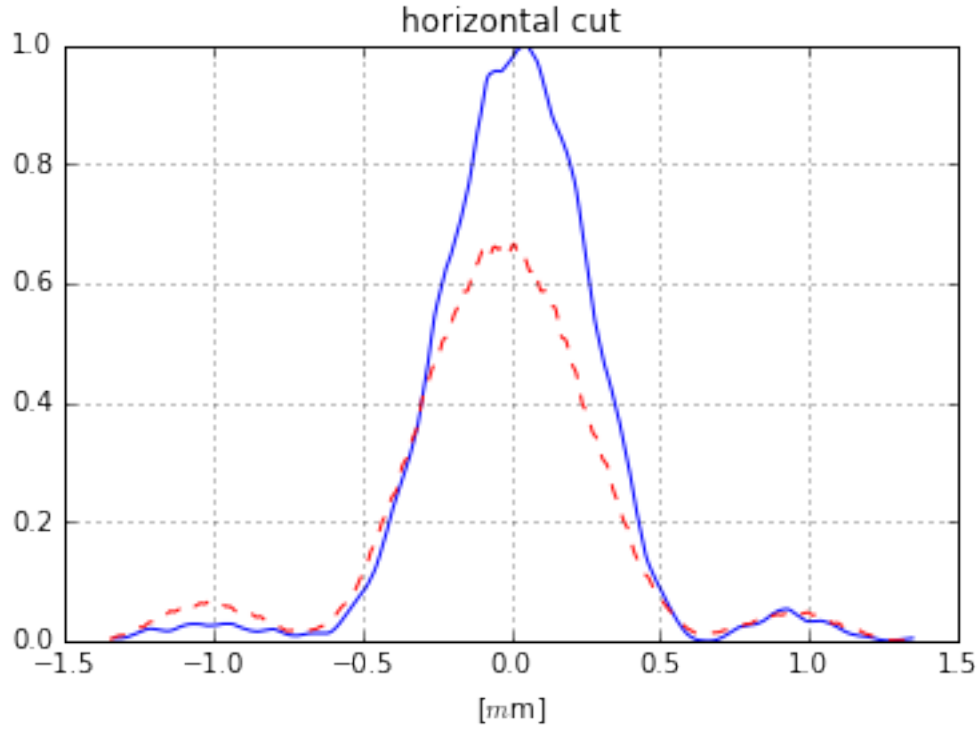


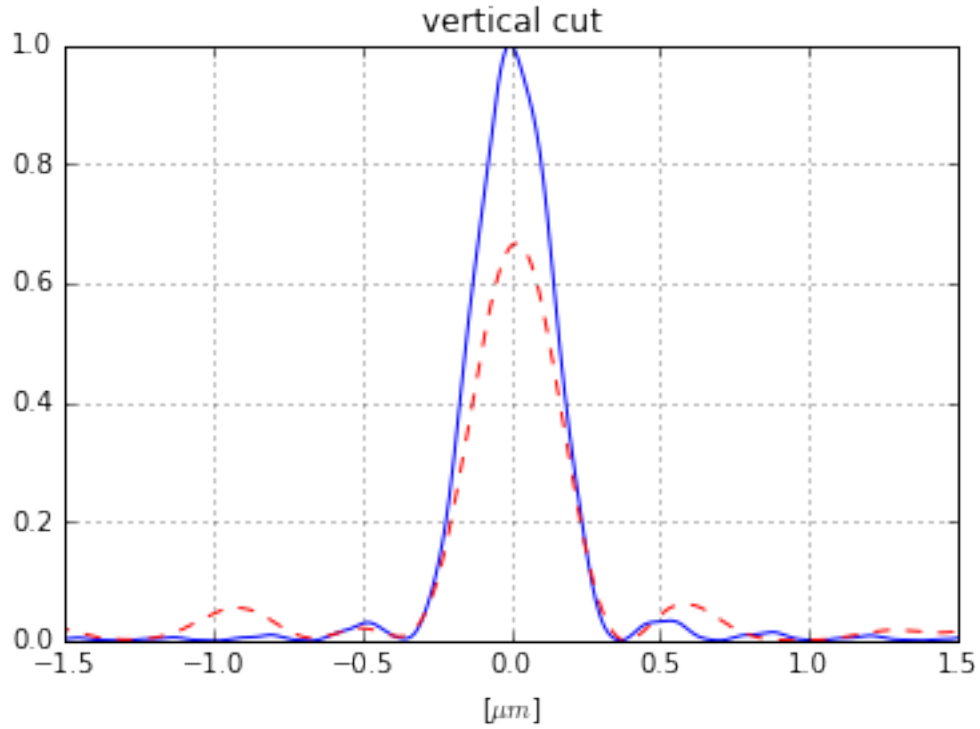

```
plt.figure()
plt.plot(dd0_h[:,0]*1e6, dd0_h[:,1]/max(dd0_h[:,1]),#/ max(dd21_950_h[:,1]),
     dd1_h[:,0]*1e6, dd1_h[:,1]/max(dd0_h[:,1]),'--r')#/max(dd120_950_h[:,1]),'--r')
plt.xlim([-1.5,1.5])
#ylim([0,1.5])
plt.title('horizontal cut')
#legend(["4 nm PV height errors","ideal KB"])
plt.xlabel('[$m$m]')
plt.grid(True)
plt.show()
plt.figure()
plt.plot(dd0_v[:,0]*1e6, dd0_v[:,1]/max(dd0_v[:,1]),#/ max(dd21_950_h[:,1]),
     dd1_v[:,0]*1e6, dd1_v[:,1]/max(dd0_v[:,1]),'--r')#/max(dd120_950_h[:,1]),'--r')
plt.xlim([-1.5,1.5])
#ylim([0,1.5])
plt.title('vertical cut')
#legend(["4 nm PV height errors","ideal KB"])
```

(continues on next page)

(continued from previous page)

```
plt.xlabel('[$\mu m$]')
plt.grid(True)
plt.show()
```





```
print('*****Focused beam behind focus: perfect KB')
#optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,
↪Drift_foc],
#                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,
↪ppDrift_foc])
z3 = dhkb_vkb
#z4 = dvkb_foc #distance to focal plane
z4 = vkbfoc

HKB = SRWLOptMirEl(_p=d2hkb, _q=dhkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85,
                   _nvx=np.cos(thetaKB), _nvy=0, _nvz=-np.sin(thetaKB), _tvx=-np.
↪sin(thetaKB),
                   _tvy=0, _x=0, _y=0, _treat_in_out=1) #HKB Ellipsoidal Mirror
VKB = SRWLOptMirEl(_p=d2vkb, _q=dvkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85,
                   _nvx=0, _nvy=np.cos(thetaKB), _nvz=-np.sin(thetaKB), _tvx=0, _tvy=-
↪np.sin(thetaKB),
                   _x=0, _y=0, _treat_in_out=1) #VKB Ellipsoidal Mirror
#HKB = SRWLOptL(hkbfoc) #HKB as Thin Lens
#VKB = SRWLOptL(1e23,vkbfoc) #VKB as Thin Lens
Drift_foc = SRWLOptD(z4)
optBL2 = SRWLOptC([opApM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,  Drift_foc],
                  [ppM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,ppDrift_foc])
#optBL3 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,opTrErHKB,  Drift_KB,
↪VKB,opTrErVKB,  Drift_foc],
#                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppTrErM1,ppDrift_KB,
↪ppVKB,ppTrErM1, ppDrift_foc])
optBL = optBL2
strBL = 'bl2'
pos_title = 'behind the focus'
print('*****setting-up optical elements, beamline:', strBL)
print_beamline(optBL)
startTime = time.time()
print('*****reading wavefront from h5 file...')
w2 = Wavefront()
w2.load_hdf5(ifname)
wfr = w2._srwl_wf
print('*****propagating wavefront (with resizing)...')
srwl.PropagElecField(wfr, optBL)
mwf = Wavefront(wfr)
print('[nx, ny, xmin, xmax, ymin, ymax]', get_mesh(mwf))
if bSaved:
    print('save hdf5:', fname0+'_'+strBL+'.h5')
    mwf.store_hdf5(os.path.join(strOutputDataFolder,fname0+'_'+strBL+'.h5'))
print('done')
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Focused beam behind focus: perfect KB
*****setting-up optical elements, beamline: bl2
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0109459510409
    ap_or_ob = a
    shape = r
    x = 0
```

(continues on next page)

(continued from previous page)

```
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 161.3
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0072
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.009
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0.999959500273
    nvy = 0
    nvz = -0.00899987850049
    p = 442.3
    q = 2.715
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = -0.00899987850049
    tvy = 0
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 0.9999999999999998
    treat = 0

Optical Element: Mirror: Ellipsoid
```

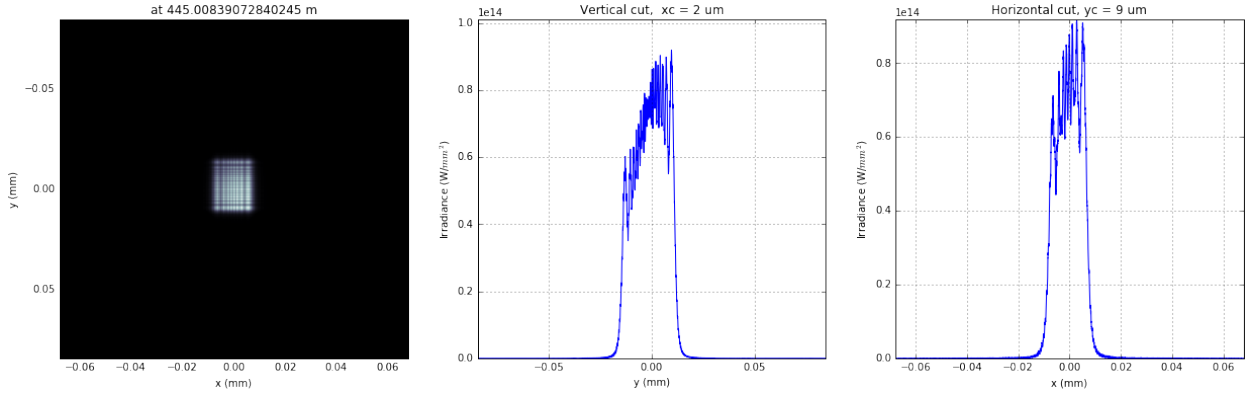
(continues on next page)

(continued from previous page)

```
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.009
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0
    nvy = 0.999959500273
    nvz = -0.00899987850049
    p = 443.3
    q = 1.715
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = 0
    tvy = -0.00899987850049
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.7083907284024138
    treat = 0


*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
[nx, ny, xmin, xmax, ymin, ymax] [8316, 4158, -6.784130971441318e-05, 6.
↪784130971441323e-05, -8.456925201660192e-05, 8.4569252016602e-05]
done
propagation lasted: 1.6 min
```

```
print('*****Focused beam behind focus: perfect KB')
bOnePlot = True
isHlog = False
isVlog = False
bSaved = False
plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m',isHlog, isVlog, 1e-6,1e-6,'x', bOnePlot)
plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [um], FWHMy [um]:',calculate_fwhm_x(mwf)*1e6,calculate_fwhm_y(mwf)*1e6)
```

```
*****Focused beam behind focus: perfect KB
FWHMx [mm]: 0.0144086293392
FWHMy [mm]: 0.024778770497
Coordinates of center, [mm]: 0.00279850501889 0.0096226259812
stepX, stepY [um]: 0.016317813521205822 0.04068763628414816

Total power (integrated over full range): 21.8569 [GW]
Peak power calculated using FWHM:         37.3409 [GW]
Max irradiance: 91925.9 [GW/mm^2]
R-space
FWHMx [um], FWHMy [um]: 14.4086293392 24.778770497
```




```
print('*****Focused beam behind focus: perfect KB')
#optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,
↪Drift_foc],
#                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,
↪ppDrift_foc])
z3 = dhkb_vkb
#z4 = dvkb_foc #distance to focal plane
z4 = vkbfoc

HKB = SRWLOptMirEl(_p=d2hkb, _q=dhkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85,
                   _nvx=np.cos(thetaKB), _nvy=0, _nvz=-np.sin(thetaKB), _tvx=-np.
↪sin(thetaKB),
                   _tvy=0, _x=0, _y=0, _treat_in_out=1) #HKB Ellipsoidal Mirror
VKB = SRWLOptMirEl(_p=d2vkb, _q=dvkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85,
                   _nvx=0, _nvy=np.cos(thetaKB), _nvz=-np.sin(thetaKB), _tvx=0, _tvy=-
↪np.sin(thetaKB),
                   _x=0, _y=0, _treat_in_out=1) #VKB Ellipsoidal Mirror
#HKB = SRWLOptL(hkbfoc) #HKB as Thin Lens
#VKB = SRWLOptL(1e23,vkbfoc) #VKB as Thin Lens
Drift_foc = SRWLOptD(z4)
optBL2 = SRWLOptC([opApM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,  Drift_foc],
                  [ppM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,ppDrift_foc])
#optBL3 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,opTrErHKB,  Drift_KB,
↪VKB,opTrErVKB,  Drift_foc],
#                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppTrErM1,ppDrift_KB,
↪ppVKB,ppTrErM1, ppDrift_foc])
optBL = optBL2
strBL = 'bl2'
pos_title = 'behind the focus'
```

(continues on next page)

(continued from previous page)

```
print('*****setting-up optical elements, beamline:', strBL)
print_beamline(optBL)
startTime = time.time()
print('*****reading wavefront from h5 file...')
w2 = Wavefront()
w2.load_hdf5(ifname)
wfr = w2._srwl_wf
print('*****propagating wavefront (with resizing)...')
srwl.PropagElecField(wfr, optBL)
mwf = Wavefront(wfr)
print('[nx, ny, xmin, xmax, ymin, ymax]', get_mesh(mwf))
if bSaved:
    print('save hdf5:', fname0+'_'+strBL+'.h5')
    mwf.store_hdf5(os.path.join(strOutputDataFolder,fname0+'_'+strBL+'.h5'))
print('done')
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Focused beam behind focus: perfect KB
*****setting-up optical elements, beamline: bl2
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0109459510409
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 161.3
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0072
    Dy = 0.0072
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.009
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
```

(continues on next page)

(continued from previous page)

```
    nvx = 0.999959500273
    nvy = 0
    nvz = -0.00899987850049
    p = 442.3
    q = 2.715
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = -0.00899987850049
    tvy = 0
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 0.9999999999999998
    treat = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.009
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0
    nvy = 0.999959500273
    nvz = -0.00899987850049
    p = 443.3
    q = 1.715
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = 0
```

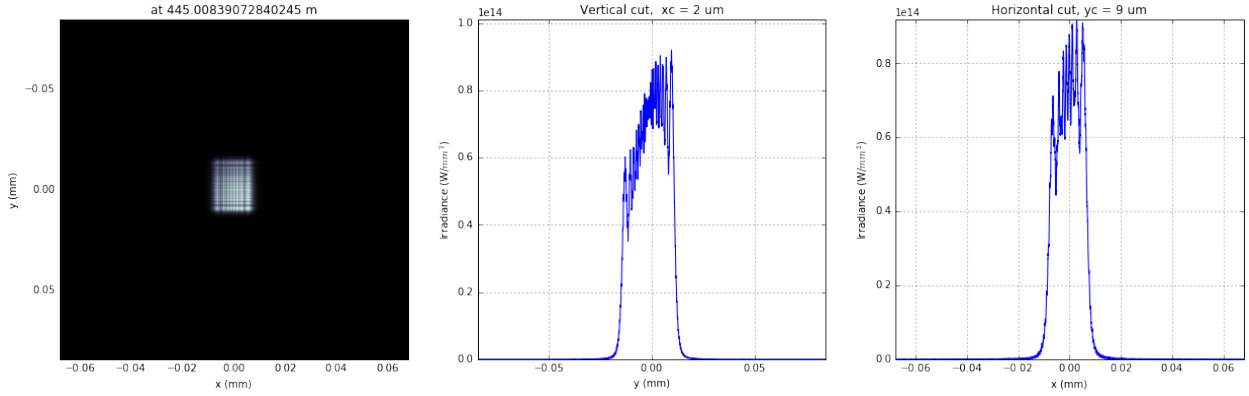
(continues on next page)

(continued from previous page)

```
    tvy = -0.00899987850049
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.7083907284024138
    treat = 0


*****reading wavefront from h5 file...
*****propagating wavefront (with resizing)...
[nx, ny, xmin, xmax, ymin, ymax] [8316, 4158, -6.784130971441318e-05, 6.
↪784130971441323e-05, -8.456925201660192e-05, 8.4569252016602e-05]
done
propagation lasted: 1.5 min
```

```
print('*****Focused beam behind focus: perfect KB')
bOnePlot = True
isHlog = False
isVlog = False
bSaved = False
plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m',isHlog, isVlog, 1e-6,1e-6,'x', bOnePlot)
plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [um], FWHMy [um]:',calculate_fwhm_x(mwf)*1e6,calculate_fwhm_y(mwf)*1e6)
```

```
*****Focused beam behind focus: perfect KB
FWHMx [mm]: 0.0144086293392
FWHMy [mm]: 0.024778770497
Coordinates of center, [mm]: 0.00279850501889 0.0096226259812
stepX, stepY [um]: 0.016317813521205822 0.04068763628414816

Total power (integrated over full range): 21.8569 [GW]
Peak power calculated using FWHM:         37.3409 [GW]
Max irradiance: 91925.9 [GW/mm^2]
R-space
FWHMx [um], FWHMy [um]: 14.4086293392 24.778770497
```



1.10.5 Wavefront propagation simulation tutorial - Case 3

L.Samoylova liubov.samoylova@xfel.eu, A.Buzmakov buzmakov@gmail.com

Tutorial course on Wavefront Propagation Simulations, 28/11/2013, European XFEL, Hamburg.

Wave optics software is based on SRW core library https://github.com/ochubar/SRW, available through WPG interactive framework https://github.com/samoylv/WPG

Propagation Gaussian through HOM and KB optics: extended analysis

Import modules

```
%matplotlib inline
```

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

#Importing necessary modules:

import os
import sys
sys.path.insert(0,os.path.join('..','..'))

import time
import copy
import numpy as np
import pylab as plt


#import SRW core functions
from wpg.srwlib import srwl,SRWLOptD,SRWLOptA,SRWLOptC,SRWLOptT,SRWLOptL,SRWLOptMirEl

#import SRW helpers functions
from wpg.useful_code.srwutils import AuxTransmAddSurfHeightProfileScaled

#import some helpers functions
from wpg.useful_code.wfrutils import calculate_fwhm_x, plot_wfront, calculate_fwhm_y,␣
↪print_beamline, get_mesh, plot_1d, plot_2d
from wpg.useful_code.wfrutils import  propagate_wavefront
#Import base wavefront class
from wpg import Wavefront

#Gaussian beam generator
from wpg.generators import build_gauss_wavefront_xy

from wpg import Beamline
from wpg.optical_elements import Empty, Use_PP

plt.ion()
```

Define auxiliary functions

```
def calculate_source_fwhm(ekev, theta_fwhm):
    """
    Calculate source size from photon energy and FWHM angular divergence

    :param evev: Energy in keV
    :param theta_fwhm: theta_fwhm [units?]
    """
    wl = 12.39e-10/ekev
    k = 2 * np.sqrt(2*np.log(2))
    theta_sigma = theta_fwhm /k
    sigma0 = wl /(2*np.pi*theta_sigma)
    return sigma0*k

def calculate_theta_fwhm_cdr(ekev,qnC):
    """
    Calculate angular divergence using formula from XFEL CDR2011

    :param ekev: Energy in keV
    :param qnC: e-bunch charge, [nC]
    :return: theta_fwhm [units?]
    """
    theta_fwhm = (17.2 - 6.4 * np.sqrt(qnC))*1e-6/ekev**0.85
    return theta_fwhm

def defineOPD(opTrErMirr, mdatafile, ncol, delim, Orient, theta, scale):
    """
    Define optical path difference (OPD) from mirror profile, i.e. ill the struct␣
↪opTrErMirr

    :params mdatafile: an ascii file with mirror profile data
    :params ncol: number of columns in the file
    :params delim: delimiter between numbers in an row, can be space (' '), tab '\t',␣
↪etc
    :params orient: mirror orientation, 'x' (horizontal) or 'y' (vertical)
    :params theta: incidence angle
    :params scale: scaling factor for the mirror profile
    """
    heightProfData = np.loadtxt(mdatafile).T
    AuxTransmAddSurfHeightProfileScaled(opTrErMirr, heightProfData, Orient, theta,␣
↪scale)
    plt.figure()
    plot_1d(heightProfData,'profile from ' + mdatafile,'x (m)', 'h (m)')
```

Defining initial wavefront and writing electric field data to h5-file

```
# #**********************Input Wavefront Structure and Parameters
print('******defining initial wavefront and writing electric field data to h5-file...')
strInputDataFolder = 'data_common'  # input data sub-folder name
strOutputDataFolder = 'Tutorial_case_3'  # output data sub-folder name

#init Gauusian beam parameters
d2m1_sase1 = 246.5
d2m1_sase2 = 290.0
```

(continues on next page)

(continued from previous page)

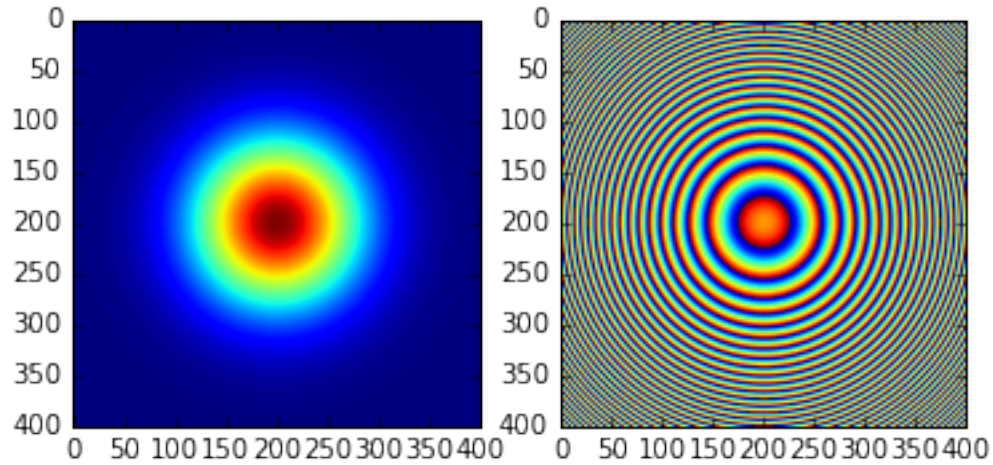
```
d2m1_sase3 = 281.0

d2hkb_sase1 = 929.6        # distance to nmKB's HFM
dHKB_foc_sase1 = 3.0       # nominal focal length for HFM KB
dVKB_foc_sase1 = 1.9       # nominal focal length for VFM KB
d2hkb_sase3 = 442.3
dHKB_foc_sase3 = 2.715     # nominal focal length for HFM KB
dVKB_foc_sase3 = 1.715     # nominal focal length for VFM KB


qnC = 0.1                      # e-bunch charge, [nC]
ekev_sase3 = 0.8
thetaOM_sase3 = 9.e-3
thetaKB_sase3 = 9.e-3
ekev_sase1 = 5.0
thetaOM_sase1 = 3.5e-3         #
thetaKB_sase1 = 3.5e-3

ekev = ekev_sase1
thetaOM = thetaOM_sase1
d2m1 = d2m1_sase1
d2hkb = d2hkb_sase1
thetaKB = thetaKB_sase1
dhkb_foc = dHKB_foc_sase1      # nominal focal length for HFM KB
dvkb_foc = dVKB_foc_sase1       # nominal focal length for VFM KB
dhkb_vkb = dhkb_foc - dvkb_foc           # distance between centers of HFM and VFM

z1 = d2m1
theta_fwhm = calculate_theta_fwhm_cdr(ekev,qnC)
k = 2*np.sqrt(2*np.log(2))
sigX = 12.4e-10*k/(ekev*4*np.pi*theta_fwhm)
print('waist_fwhm [um], theta_fwhms [urad]:', sigX*k*1e6, theta_fwhm*1e6)
#define limits
range_xy = theta_fwhm/k*z1*7. # sigma*7 beam size
npoints=400

#define unique filename for storing results
ip = np.floor(ekev)
frac = np.floor((ekev - ip)*1e3)
fname0 = 'g' + str(int(ip))+'_'+str(int(frac))+'kev'
print('save hdf5: '+fname0+'.h5')
ifname = os.path.join(strOutputDataFolder,fname0+'.h5')

#tiltX = theta_rms
#build SRW gauusian wavefront
wfr0=build_gauss_wavefront_xy(nx=npoints, ny=npoints, ekev=ekev,
                              xMin=-range_xy/2, xMax=range_xy/2,
                              yMin=-range_xy/2, yMax=range_xy/2,
                              sigX=sigX, sigY=sigX, d2waist=z1,
                              xoff=0, yoff=0, tiltX=0, tiltY=0)

#init WPG Wavefront helper class
mwf = Wavefront(wfr0)

#store wavefront to HDF5 file
mwf.store_hdf5(ifname)
```

(continues on next page)

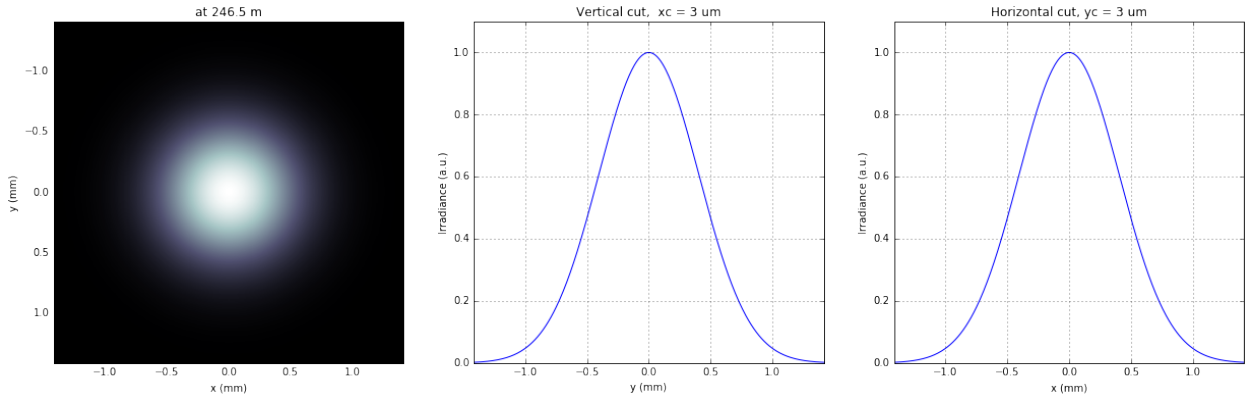
(continued from previous page)

```
#draw wavefront with common functions
plt.subplot(1,2,1)
plt.imshow(mwf.get_intensity(slice_number=0))
plt.subplot(1,2,2)
plt.imshow(mwf.get_phase(slice_number=0,polarization='horizontal'))
plt.show()

#draw wavefront with cuts
plot_wfront(mwf, title_fig='at '+str(z1)+' m',
            isHlog=False, isVlog=False,
            i_x_min=1e-5, i_y_min=1e-5, orient='x', onePlot=True)

plt.set_cmap('bone') #set color map, 'bone', 'hot', 'jet', etc
fwhm_x = calculate_fwhm_x(mwf)
print('FWHMx [mm], theta_fwhm [urad]:',fwhm_x*1e3,fwhm_x/z1*1e6)
```

```
*****defining initial wavefront and writing electric field data to h5-file...
waist_fwhm [um], theta_fwhms [urad]: 28.3217691481 3.86399794107
save hdf5: g5_0kev.h5
```




```
FWHMx [mm]: 0.943784566665
FWHMy [mm]: 0.943784566665
Coordinates of center, [mm]: 0.0035480622807 0.0035480622807
stepX, stepY [um]: 7.096124561394084 7.096124561394084

R-space
FWHMx [mm], theta_fwhm [urad]: 0.943784566665 3.82874063556
```



Defining optical beamline(s)

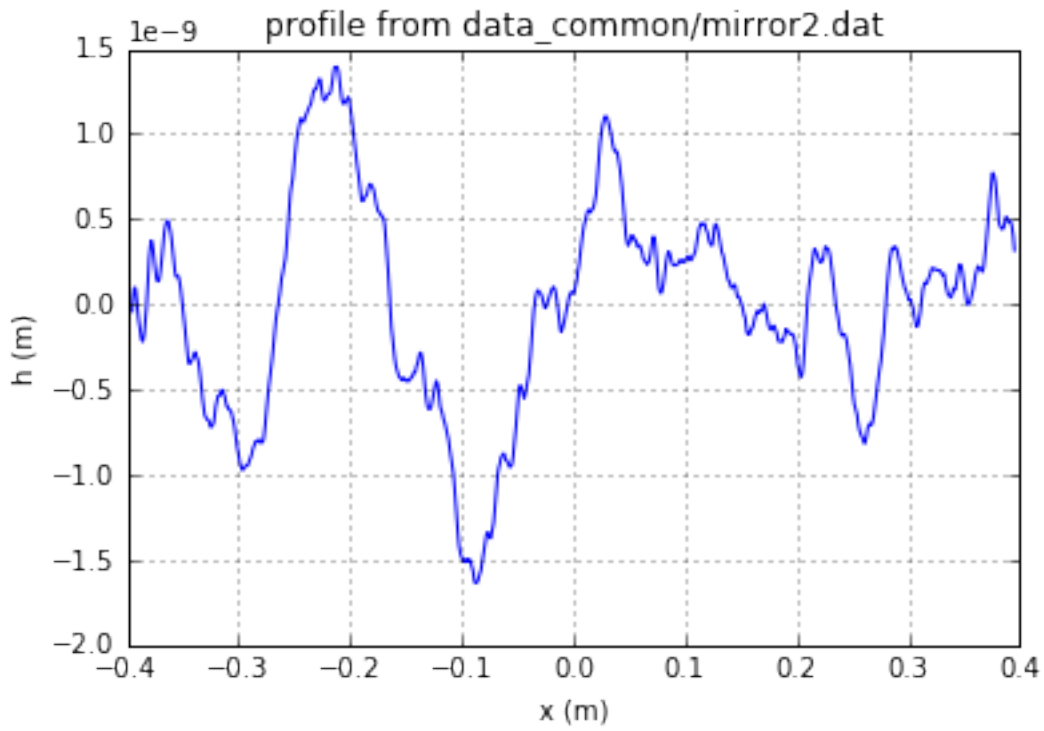
```
print('*****Defining optical beamline(s) ...')

z2 = d2hkb - d2m1

DriftM1_KB = SRWLOptD(z2) #Drift from first offset mirror (M1) to exp hall
horApM1 = 0.8*thetaOM
opApM1 = SRWLOptA('r', 'a', horApM1, range_xy)  # clear aperture of the Offset␣
↪Mirror(s)
horApKB = 0.8 * thetaKB # Aperture of the KB system, CA 0.8 m
opApKB = SRWLOptA('r', 'a', horApKB, horApKB)  # clear aperture of the Offset␣
↪Mirror(s)

#Wavefront Propagation Parameters:
#[0]:  Auto-Resize (1) or not (0) Before propagation
#[1]:  Auto-Resize (1) or not (0) After propagation
#[2]:  Relative Precision for propagation with Auto-Resizing (1. is nominal)
#[3]:  Allow (1) or not (0) for semi-analytical treatment of quadratic phase terms at␣
↪propagation
#[4]:  Do any Resizing on Fourier side, using FFT, (1) or not (0)
#[5]:  Horizontal Range modification factor at Resizing (1. means no modification)
#[6]:  Horizontal Resolution modification factor at Resizing
#[7]:  Vertical Range modification factor at Resizing
#[8]:  Vertical Resolution modification factor at Resizing
#[9]:  Type of wavefront Shift before Resizing (not yet implemented)
#[10]: New Horizontal wavefront Center position after Shift (not yet implemented)
#[11]: New Vertical wavefront Center position after Shift (not yet implemented)
#                  [ 0] [1] [2]  [3] [4] [5]  [6]  [7]  [8]  [9] [10] [11]
ppM1 =             [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppTrErM1 =         [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDriftM1_KB =     [ 0,  0, 1.0,  1,  0, 2.4, 1.8, 2.4, 1.8,  0,  0,   0]
ppApKB =           [ 0,  0, 1.0,  0,  0, 0.6, 8.0, 0.6, 4.0,  0,  0,   0]
ppHKB =            [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppTrErHKB =        [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDrift_HKB_foc =  [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDrift_KB =       [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppVKB =            [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppTrErVKB =        [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDrift_foc =      [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
#ppFin  =           [ 0,  0, 1.0,  0,  0, 0.05,5.0, 0.05,5.0,  0,  0,   0]
ppFin =            [ 0,  0, 1.0,  0,  1, .01, 20.0, .01, 20.0,  0,  0,   0]
```

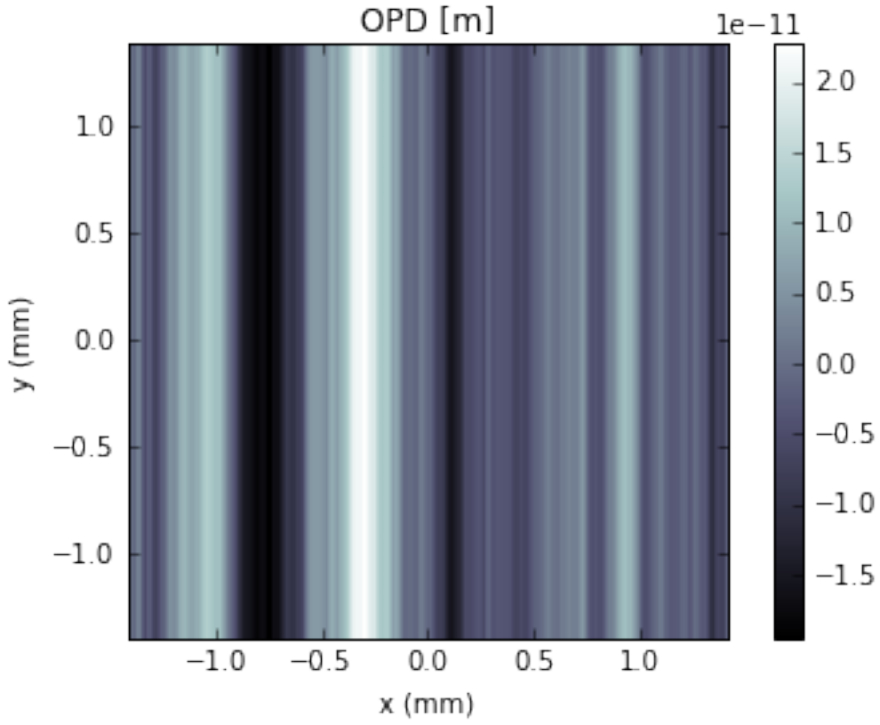
(continues on next page)

(continued from previous page)

```
optBL0 = SRWLOptC([opApM1,  DriftM1_KB],
                  [ppM1,ppDriftM1_KB])

scale = 2     #5 mirror profile scaling factor
print('*****HOM1 data for BL1 beamline ')
opTrErM1 = SRWLOptT(1500, 100, horApM1, range_xy)
#defineOPD(opTrErM1, os.path.join(strInputDataFolder,'mirror1.dat'), 2, '\t', 'x',␣
↪thetaOM, scale)
defineOPD(opTrErM1, os.path.join(strInputDataFolder,'mirror2.dat'), 2, ' ', 'x',␣
↪thetaOM, scale)
opdTmp=np.array(opTrErM1.arTr)[1::2].reshape(opTrErM1.mesh.ny,opTrErM1.mesh.nx)
plt.figure()
plot_2d(opdTmp, opTrErM1.mesh.xStart*1e3,opTrErM1.mesh.xFin*1e3,opTrErM1.mesh.
↪yStart*1e3,opTrErM1.mesh.yFin*1e3,
        'OPD [m]', 'x (mm)', 'y (mm)')

optBL1 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB],
                  [ppM1,ppTrErM1,ppDriftM1_KB])

dhkb_vkb = dhkb_foc - dvkb_foc          # distance between centers of HFM and VFM
d2vkb = d2hkb +  dhkb_vkb
vkbfoc =  1. /(1./dvkb_foc + 1. / d2vkb) # for thin lens approx
hkbfoc =  1. /(1./dhkb_foc + 1. / d2hkb) # for thin lens approx

z3 = dhkb_vkb
z4 = vkbfoc #distance to focal plane

#HKB = SRWLOptMirEl(_p=d2hkb, _q=dhkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=cos(thetaKB), _nvy=0, _nvz=-sin(thetaKB), _tvx=-sin(thetaKB), _
↪tvy=0, _x=0, _y=0, _treat_in_out=1) #HKB Ellipsoidal Mirror
#VKB = SRWLOptMirEl(_p=d2vkb, _q=dvkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=0, _nvy=cos(thetaKB), _nvz=-sin(thetaKB), _tvx=0, _tvy=-
↪sin(thetaKB), _x=0, _y=0, _treat_in_out=1) #VKB Ellipsoidal Mirror
HKB = SRWLOptL(hkbfoc) #HKB as Thin Lens
VKB = SRWLOptL(1e23,vkbfoc) #VKB as Thin Lens
Drift_KB  = SRWLOptD(z3)
Drift_foc = SRWLOptD(z4)
optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,  Drift_
↪foc],
                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,ppDrift_
↪foc,ppFin])
```

```
*****Defining optical beamline(s) ...
*****HOM1 data for BL1 beamline
```





Propagating through BL0 beamline. Ideal mirror: HOM as an aperture

```
print('*****Ideal mirror: HOM as an aperture')
```

(continues on next page)

(continued from previous page)

```
bPlotted = False
isHlog = False
isVlog = False
bSaved = True
optBL = optBL0
strBL = 'bl0'
pos_title = 'at exp hall wall'
print('*****setting-up optical elements, beamline:', strBL)
bl = Beamline(optBL)
print(bl)

if bSaved:
    out_file_name = os.path.join(strOutputDataFolder, fname0+'_'+strBL+'.h5')
    print('save hdf5:', out_file_name)
else:
    out_file_name = None

startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Ideal mirror: HOM as an aperture
*****setting-up optical elements, beamline: bl0
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0


save hdf5: Tutorial_case_3/g5_0kev_bl0.h5
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0


*****reading wavefront from h5 file...
R-space
```

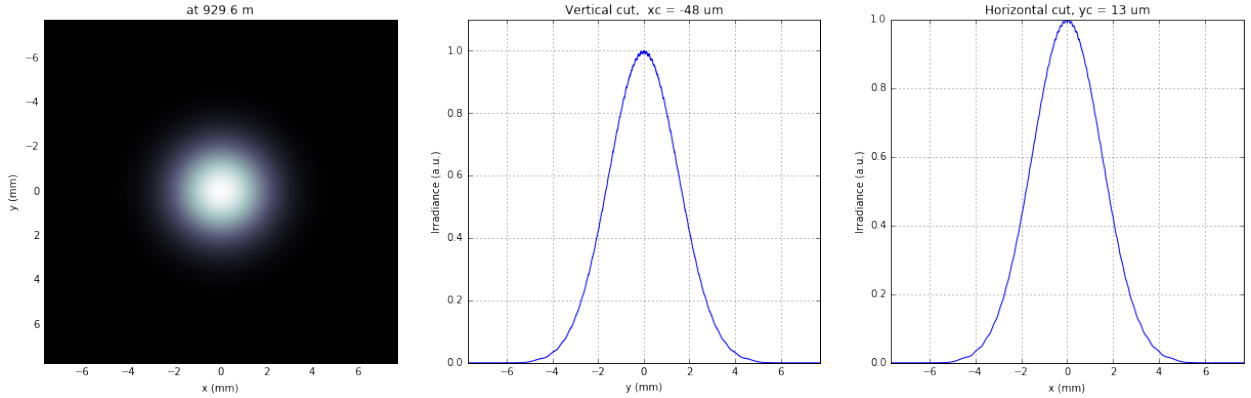
(continues on next page)

(continued from previous page)

```
nx    400  range_x [-1.4e+00, 1.4e+00] mm
ny    400  range_y [-1.4e+00, 1.4e+00] mm
*****propagating wavefront (with resizing)...
save hdf5: Tutorial_case_3/g5_0kev_bl0.h5
done
propagation lasted: 0.1 min
```

```
print('*****Ideal mirror: HOM as an aperture')
plot_wfront(mwf, 'at '+str(z1+z2)+' m',False, False, 1e-5,1e-5,'x', True)
#plt.set_cmap('bone') #set color map, 'bone', 'hot', 'jet', etc
plt.axis('tight')
print('FWHMx [mm], theta_fwhm [urad]:',calculate_fwhm_x(mwf)*1e3,calculate_fwhm_
↪x(mwf)/(z1+z2)*1e6)
print('FWHMy [mm], theta_fwhm [urad]:',calculate_fwhm_y(mwf)*1e3,calculate_fwhm_
↪y(mwf)/(z1+z2)*1e6)
```

```
*****Ideal mirror: HOM as an aperture
FWHMx [mm]: 3.58585062243
FWHMy [mm]: 3.55209077523
Coordinates of center, [mm]: -0.0489384079984 0.0134209978913
stepX, stepY [um]: 8.897892363337482 8.94733192752122

R-space
FWHMx [mm], theta_fwhm [urad]: 3.58585062243 3.85741245958
FWHMy [mm], theta_fwhm [urad]: 3.55209077523 3.8210959286
```



Propagating through BL1 beamline. Imperfect mirror, at KB aperture

```
print('*****Imperfect HOM mirror, at KB aperture')
bPlotted = False
isHlog = True
isVlog = False
bSaved = False
optBL = optBL1
strBL = 'bl1'
pos_title = 'at exp hall wall'
print('*****setting-up optical elements, beamline:', strBL)
bl = Beamline(optBL)
print(bl)
```

(continues on next page)

(continued from previous page)

```
if bSaved:
    out_file_name = os.path.join(strOutputDataFolder, fname0+'_'+strBL+'.h5')
    print('save hdf5:', out_file_name)
else:
    out_file_name = None

startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Imperfect HOM mirror, at KB aperture
*****setting-up optical elements, beamline: bl1
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1500
             ny = 100
             xFin = 0.0014000000000000002
             xStart = -0.0014000000000000002
             yFin = 0.00141567685
             yStart = -0.00141567685
             zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0


Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
```

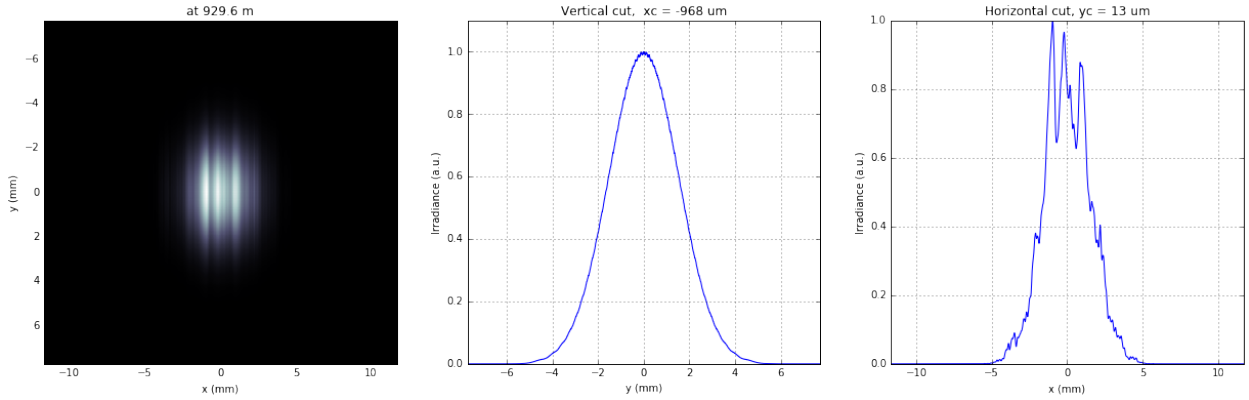
(continues on next page)

(continued from previous page)

```
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1500
             ny = 100
             xFin = 0.0014000000000000002
             xStart = -0.0014000000000000002
             yFin = 0.00141567685
             yStart = -0.00141567685
             zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0


*****reading wavefront from h5 file...
R-space
nx   400  range_x [-1.4e+00, 1.4e+00] mm
ny   400  range_y [-1.4e+00, 1.4e+00] mm
*****propagating wavefront (with resizing)...
done
propagation lasted: 0.1 min
```

```
print('*****Imperfect HOM mirror, at KB aperture')
plot_wfront(mwf, 'at '+str(z1+z2)+' m',False, False, 1e-5,1e-5,'x', True)
#plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [mm], theta_fwhm [urad]:',calculate_fwhm_x(mwf)*1e3,calculate_fwhm_
↪x(mwf)/(z1+z2)*1e6)
print('FWHMy [mm], theta_fwhm [urad]:',calculate_fwhm_y(mwf)*1e3,calculate_fwhm_
↪y(mwf)/(z1+z2)*1e6)
```

```
*****Imperfect HOM mirror, at KB aperture
FWHMx [mm]: 2.89758221191
FWHMy [mm]: 3.55209077523
Coordinates of center, [mm]: -0.968117421268 0.0134209978913
stepX, stepY [um]: 13.540103793954389 8.94733192752122

R-space
FWHMx [mm], theta_fwhm [urad]: 2.89758221191 3.11702045171
FWHMy [mm], theta_fwhm [urad]: 3.55209077523 3.8210959286
```



Propagating through BL2 beamline. Focused beam: perfect KB

```
print('*****Focused beam: perfect KB')
bSaved = False
z3 = dhkb_vkb
z4 = dvkb_foc
z4 = vkbfoc #distance to focal plane

#HKB = SRWLOptMirEl(_p=d2hkb, _q=dhkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=cos(thetaKB), _nvy=0, _nvz=-sin(thetaKB), _tvx=-sin(thetaKB), _
↪tvy=0, _x=0, _y=0, _treat_in_out=1) #HKB Ellipsoidal Mirror
#VKB = SRWLOptMirEl(_p=d2vkb, _q=dvkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=0, _nvy=cos(thetaKB), _nvz=-sin(thetaKB), _tvx=0, _tvy=-
↪sin(thetaKB), _x=0, _y=0, _treat_in_out=1) #VKB Ellipsoidal Mirror
#HKB = SRWLOptL(hkbfoc) #HKB as Thin Lens
#VKB = SRWLOptL(1e23,vkbfoc) #VKB as Thin Lens
Drift_foc = SRWLOptD(dvkb_foc)
#optBL2 = SRWLOptC([opApM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,  Drift_foc],
#                  [ppM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,ppDrift_foc,
↪ppFin])
optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,  Drift_
↪foc],
                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,ppDrift_
↪foc])
optBL = optBL2
strBL = 'bl2'
pos_title = 'at sample position'
print('*****setting-up optical elements, beamline:', strBL)
bl = Beamline(optBL)
bl.append(Empty(), Use_PP(zoom=0.02, sampling=5.0))
print(bl)
```

(continues on next page)

(continued from previous page)

```
if bSaved:
    out_file_name = os.path.join(strOutputDataFolder, fname0+'_'+strBL+'.h5')
    print('save hdf5:', out_file_name)
else:
    out_file_name = None

startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Focused beam: perfect KB
*****setting-up optical elements, beamline: bl2
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1500
             ny = 100
             xFin = 0.0014000000000000002
             xStart = -0.0014000000000000002
             yFin = 0.00141567685
             yStart = -0.00141567685
             zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0028000000000000004
```

(continues on next page)

(continued from previous page)

```
    Dy = 0.0028000000000000004
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 2.9903495603688612
    Fy = 1e+23
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.1
    treat = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1.8961291014368435
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.9
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 0, 0.02, 5.0, 0.02, 5.0, 0, 0, 0]


Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
```

(continues on next page)

(continued from previous page)

```
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.00141567685
            yStart = -0.00141567685
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028000000000000004
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 2.9903495603688612
    Fy = 1e+23
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.1
    treat = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1.8961291014368435
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.9
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 0, 0.02, 5.0, 0.02, 5.0, 0, 0, 0]
```

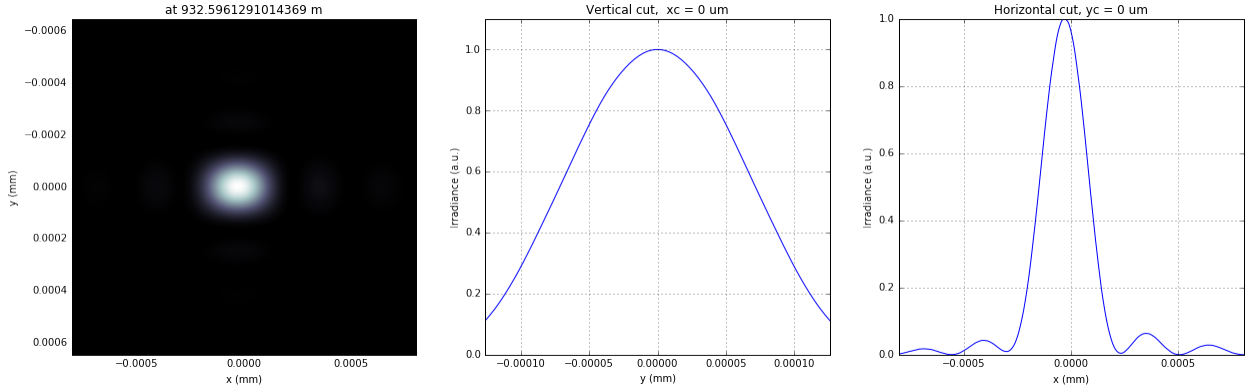
(continues on next page)

(continued from previous page)

```
*****reading wavefront from h5 file...
R-space
nx    400   range_x [-1.4e+00, 1.4e+00] mm
ny    400   range_y [-1.4e+00, 1.4e+00] mm
*****propagating wavefront (with resizing)...
done
propagation lasted: 1.6 min
```

```
print('*****Focused beam: Focused beam: perfect KB')
bOnePlot = True
isHlog = False
isVlog = False
plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m',isHlog, isVlog, 1e-5,1e-5,'x', bOnePlot)
#plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [um], FWHMy [um]:',calculate_fwhm_x(mwf)*1e6,calculate_fwhm_y(mwf)*1e6)
```

```
*****Focused beam: Focused beam: perfect KB
FWHMx [mm]: 0.000234193434075
FWHMy [mm]: 0.000152352450272
Coordinates of center, [mm]: -3.19354682829e-05 1.55461683951e-06
stepX, stepY [um]: 0.001935482926236052 0.0031092336790285112

R-space
FWHMx [um], FWHMy [um]: 0.234193434075 0.152352450272
```



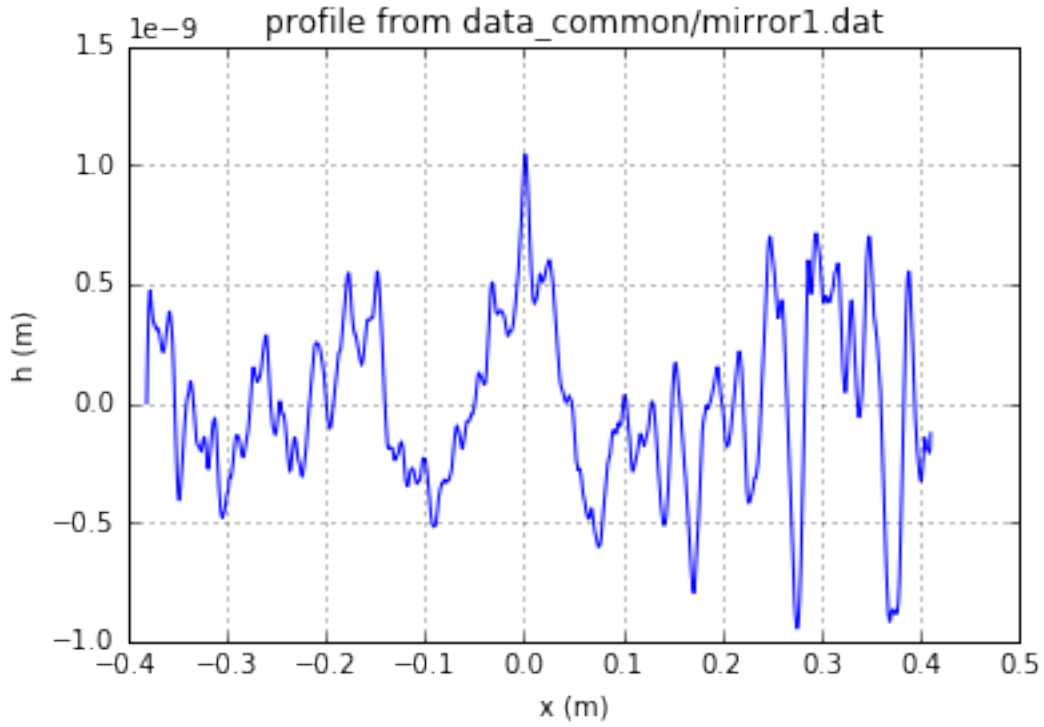

```
print('*****HKB data  ')
scale = 2 #scaling factor of mirror
opTrErHKB = SRWLOptT(1500, 100, horApKB, horApKB)
defineOPD(opTrErHKB, os.path.join(strInputDataFolder,'mirror1.dat'), 2, '\t', 'x',␣
↪thetaOM, scale)
opdTmp=np.array(opTrErHKB.arTr)[1::2].reshape(opTrErHKB.mesh.ny,opTrErHKB.mesh.nx)
plt.figure()
plot_2d(opdTmp, opTrErM1.mesh.xStart*1e3,opTrErM1.mesh.xFin*1e3,opTrErM1.mesh.
↪yStart*1e3,opTrErM1.mesh.yFin*1e3,
        'OPD [m]', 'x (mm)', 'y (mm)')
print('*****VKB data  ')
opTrErVKB = SRWLOptT(100, 1500, horApKB, horApKB)
defineOPD(opTrErVKB, os.path.join(strInputDataFolder,'mirror2.dat'), 2, ' ', 'y',␣
↪thetaOM, scale)
```

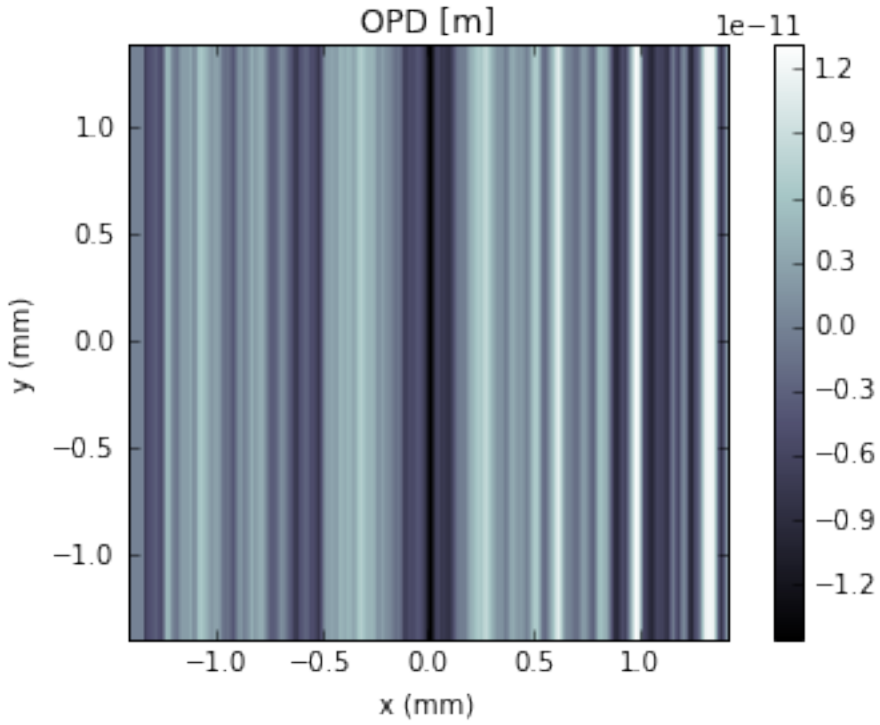
(continues on next page)

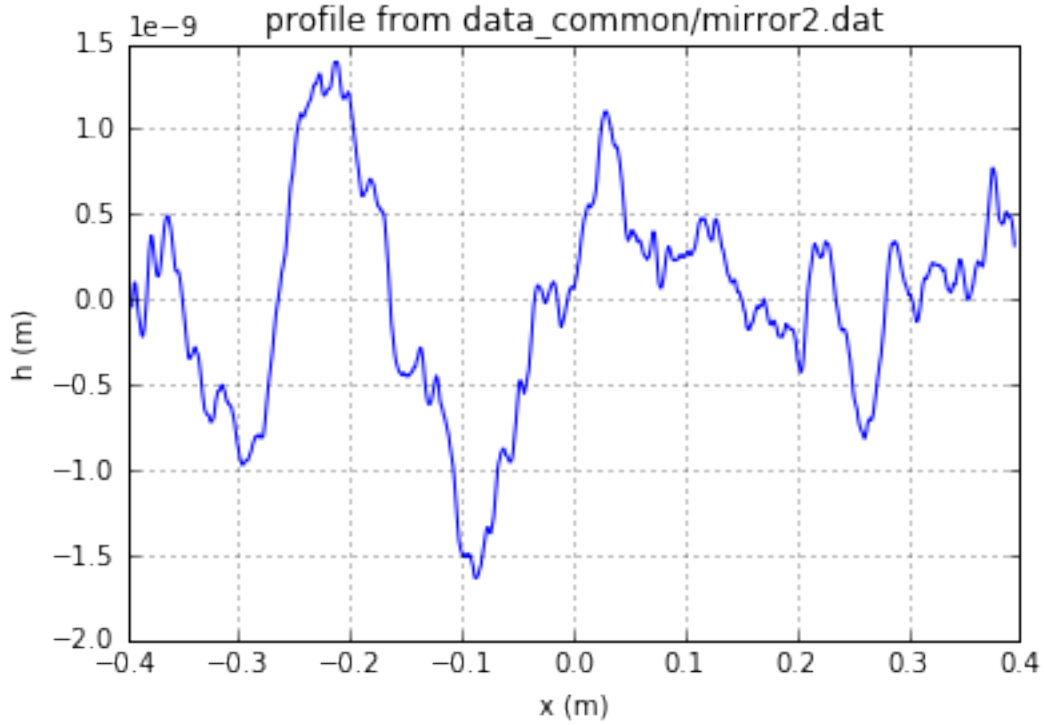
(continued from previous page)

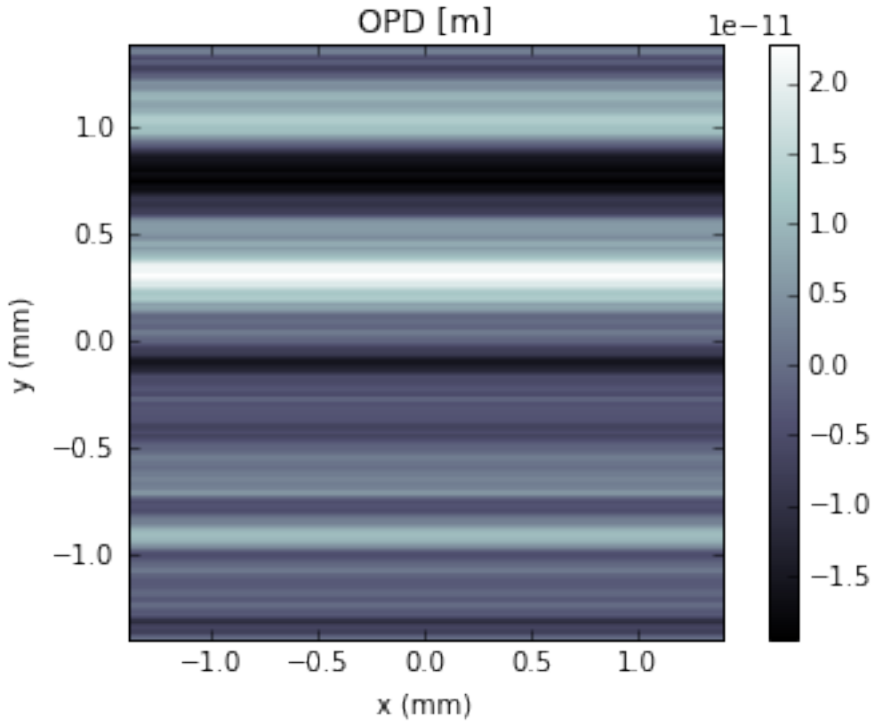
```
opdTmp=np.array(opTrErVKB.arTr)[1::2].reshape(opTrErVKB.mesh.ny,opTrErVKB.mesh.nx)
plt.figure()
plot_2d(opdTmp, opTrErVKB.mesh.xStart*1e3,opTrErVKB.mesh.xFin*1e3,opTrErVKB.mesh.
↪yStart*1e3,opTrErVKB.mesh.yFin*1e3,
        'OPD [m]', 'x (mm)', 'y (mm)')
```

```
*****HKB data
*****VKB data
```




OPD [m]
1e−11
1.2
0.9
0.6
0.3
0.0
−0.3
−0.6
−0.9
−1.2
1.0
0.5
0.0
−0.5
−1.0
y (mm)
−1.0
−0.5
0.0
0.5
1.0
x (mm)


profile from data_common/mirror2.dat
1e−9
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
h (m)
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
x (m)




```
print('*****Focused beam on focus: imperfect KB')
z3 = dhkb_vkb
z4 = dvkb_foc #distance to focal plane
#z4 = vkbfoc  #focus distance of lens

#HKB = SRWLOptMirEl(_p=d2hkb, _q=dhkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=cos(thetaKB), _nvy=0, _nvz=-sin(thetaKB), _tvx=-sin(thetaKB), _
↪tvy=0, _x=0, _y=0, _treat_in_out=1) #HKB Ellipsoidal Mirror
#VKB = SRWLOptMirEl(_p=d2vkb, _q=dvkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=0, _nvy=cos(thetaKB), _nvz=-sin(thetaKB), _tvx=0, _tvy=-
↪sin(thetaKB), _x=0, _y=0, _treat_in_out=1) #VKB Ellipsoidal Mirror
#HKB = SRWLOptL(hkbfoc) #HKB as Thin Lens
#VKB = SRWLOptL(1e23,vkbfoc) #VKB as Thin Lens
Drift_foc = SRWLOptD(z4)
optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,opTrErHKB,  Drift_KB, ␣
↪VKB,opTrErVKB,  Drift_foc],
                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppTrErM1,ppDrift_KB,
↪ppVKB,ppTrErM1, ppDrift_foc])
optBL = optBL2


bl = Beamline(optBL)
bl.append(Empty(), Use_PP(zoom=0.02, sampling=5.0))

print(bl)

if bSaved:
    out_file_name = os.path.join(strOutputDataFolder, fname0+'_'+strBL+'.h5')
    print('save hdf5:', out_file_name)
else:
    out_file_name = None
```

(continues on next page)

(continued from previous page)

```
startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Focused beam on focus: imperfect KB
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.00141567685
            yStart = -0.00141567685
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028000000000000004
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Thin Lens
```

(continues on next page)

(continued from previous page)

```
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 2.9903495603688612
    Fy = 1e+23
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.0014000000000000002
            yStart = -0.0014000000000000002
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.1
    treat = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1.8961291014368435
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
```

(continues on next page)

(continued from previous page)

```
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 100
            ny = 1500
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.0014000000000000002
            yStart = -0.0014000000000000002
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.9
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 0, 0.02, 5.0, 0.02, 5.0, 0, 0, 0]


Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.00141567685
            yStart = -0.00141567685
            zStart = 0
```

(continues on next page)

(continued from previous page)

```
Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028000000000000004
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 2.9903495603688612
    Fy = 1e+23
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.0014000000000000002
            yStart = -0.0014000000000000002
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.1
    treat = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
```

(continues on next page)

(continued from previous page)

```
    Fy = 1.8961291014368435
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 100
            ny = 1500
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.0014000000000000002
            yStart = -0.0014000000000000002
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.9
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 0, 0.02, 5.0, 0.02, 5.0, 0, 0, 0]


*****reading wavefront from h5 file...
R-space
nx   400  range_x [-1.4e+00, 1.4e+00] mm
ny   400  range_y [-1.4e+00, 1.4e+00] mm
*****propagating wavefront (with resizing)...
done
propagation lasted: 1.5 min
```

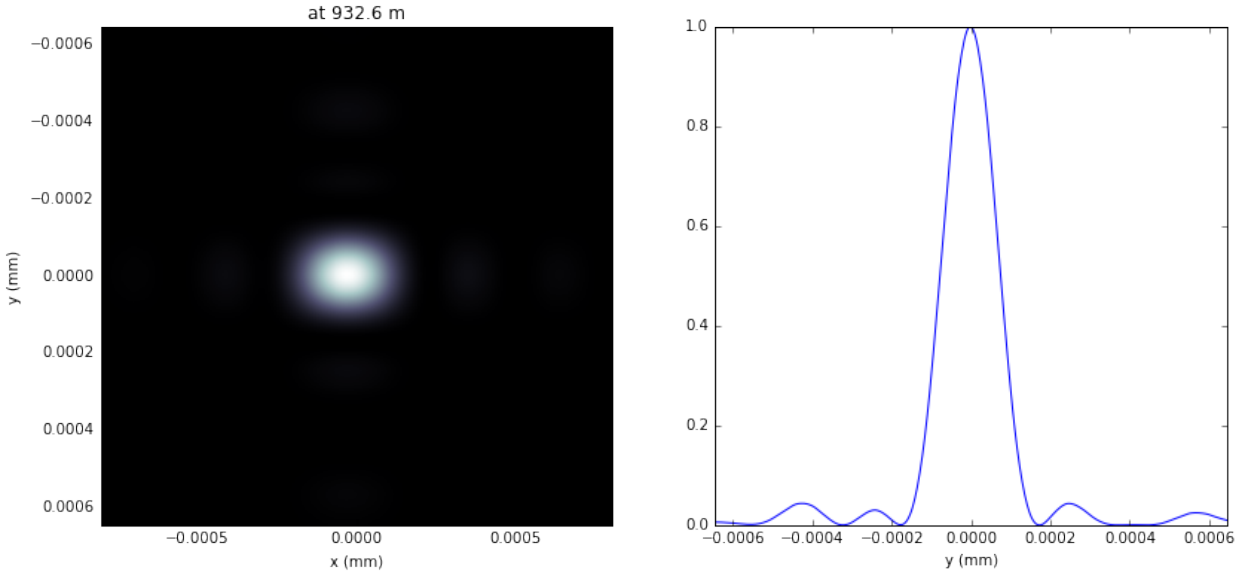
```
print('*****Focused beam: Focused beam: imperfect KB')
bOnePlot= True
isHlog = False
isVlog = False
bSaved = False
try:
    plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m',isHlog, isVlog, 1e-3,1e-3,'x',␣
↪bOnePlot)
```

(continues on next page)

(continued from previous page)

```
except ValueError as e:
    print(e)
#plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [um], FWHMy [um]:',calculate_fwhm_x(mwf)*1e6,calculate_fwhm_y(mwf)*1e6)
```

```
*****Focused beam: Focused beam: imperfect KB
FWHMx [mm]: 0.000236128917001
FWHMy [mm]: 0.000149243216593
Coordinates of center, [mm]: -3.38709512091e-05 -1.55461683952e-06
stepX, stepY [um]: 0.001935482926236052 0.0031092336790285112

R-space
zero-size array to reduction operation minimum which has no identity
FWHMx [um], FWHMy [um]: 0.236128917001 0.149243216593
```




```
print('*****Focused beam behind focus: imperfect KB')
#optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,
↪Drift_foc],
#                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,
↪ppDrift_foc])
z3 = dhkb_vkb
#z4 = dvkb_foc #distance to focal plane
z4 = vkbfoc

#HKB = SRWLOptMirEl(_p=d2hkb, _q=dhkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=cos(thetaKB), _nvy=0, _nvz=-sin(thetaKB), _tvx=-sin(thetaKB), _
↪tvy=0, _x=0, _y=0, _treat_in_out=1) #HKB Ellipsoidal Mirror
#VKB = SRWLOptMirEl(_p=d2vkb, _q=dvkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=0, _nvy=cos(thetaKB), _nvz=-sin(thetaKB), _tvx=0, _tvy=-
↪sin(thetaKB), _x=0, _y=0, _treat_in_out=1) #VKB Ellipsoidal Mirror
#HKB = SRWLOptL(hkbfoc) #HKB as Thin Lens
#VKB = SRWLOptL(1e23,vkbfoc) #VKB as Thin Lens
Drift_foc = SRWLOptD(z4)
optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,opTrErHKB,  Drift_KB,
↪VKB,opTrErVKB,  Drift_foc],
```

(continues on next page)

(continued from previous page)

```
                    [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppTrErM1,ppDrift_KB,
↪ppVKB,ppTrErM1, ppDrift_foc])
optBL = optBL2
strBL = 'bl2'
pos_title = 'at sample position'
print('*****setting-up optical elements, beamline:', strBL)
bl = Beamline(optBL)
print(bl)

if bSaved:
    out_file_name = os.path.join(strOutputDataFolder, fname0+'_'+strBL+'.h5')
    print('save hdf5:', out_file_name)
else:
    out_file_name = None

startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Focused beam behind focus: imperfect KB
*****setting-up optical elements, beamline: bl2
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.00141567685
            yStart = -0.00141567685
            zStart = 0
```

(continues on next page)

(continued from previous page)

```
Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028000000000000004
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 2.9903495603688612
    Fy = 1e+23
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.0014000000000000002
            yStart = -0.0014000000000000002
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.1
    treat = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1.8961291014368435
    x = 0
```

(continues on next page)

(continued from previous page)

```
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 100
             ny = 1500
             xFin = 0.0014000000000000002
             xStart = -0.0014000000000000002
             yFin = 0.0014000000000000002
             yStart = -0.0014000000000000002
             zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.8961291014368435
    treat = 0


Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
```

(continues on next page)

(continued from previous page)

```
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.00141567685
            yStart = -0.00141567685
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028000000000000004
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 2.9903495603688612
    Fy = 1e+23
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.0014000000000000002
            yStart = -0.0014000000000000002
            zStart = 0
```

(continues on next page)

(continued from previous page)

```
Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.1
    treat = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1.8961291014368435
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 100
             ny = 1500
             xFin = 0.0014000000000000002
             xStart = -0.0014000000000000002
             yFin = 0.0014000000000000002
             yStart = -0.0014000000000000002
             zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.8961291014368435
    treat = 0


*****reading wavefront from h5 file...
R-space
nx   400  range_x [-1.4e+00, 1.4e+00] mm
ny   400  range_y [-1.4e+00, 1.4e+00] mm
*****propagating wavefront (with resizing)...
done
propagation lasted: 1.5 min
```

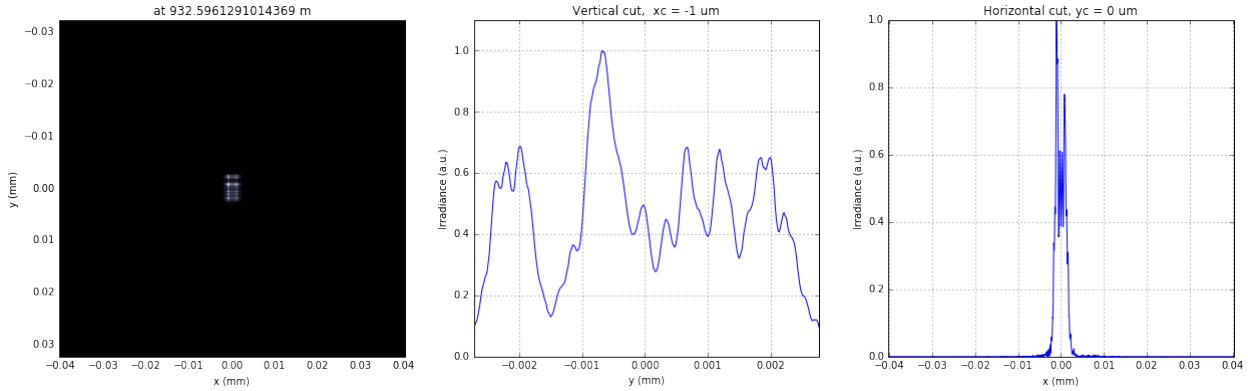
```
print('*****Focused beam behind focus: imperfect KB')
bOnePlot= True
```

(continues on next page)

(continued from previous page)

```
isHlog = False
isVlog = False
bSaved = False
plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m',isHlog, isVlog, 1e-3,1e-3,'x', bOnePlot)
#plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [um], FWHMy [um]:',calculate_fwhm_x(mwf)*1e6,calculate_fwhm_y(mwf)*1e6)
```

```
*****Focused beam behind focus: imperfect KB
FWHMx [mm]: 0.0023646431961
FWHMy [mm]: 0.00447881848302
Coordinates of center, [mm]: -0.00106118209005 -0.000676488208373
stepX, stepY [um]: 0.00969116063974761 0.015551453066055028

R-space
FWHMx [um], FWHMy [um]: 2.3646431961 4.47881848302
```



1.10.6 Wavefront propagation simulation tutorial - Case 3_new

L.Samoylova liubov.samoylova@xfel.eu, A.Buzmakov buzmakov@gmail.com

Tutorial course on Wavefront Propagation Simulations, 28/11/2013, European XFEL, Hamburg.

Wave optics software is based on SRW core library https://github.com/ochubar/SRW, available through WPG interactive framework https://github.com/samoylv/WPG

Propagation Gaussian through HOM and KB optics: extended analysis

Import modules

```
%matplotlib inline
```

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

#Importing necessary modules:
```

(continues on next page)

(continued from previous page)

```
import os
import sys
sys.path.insert(0,os.path.join('..','..'))

import time
import copy
import numpy as np
import pylab as plt


#import SRW core functions
from wpg.srwlib import srwl,SRWLOptD,SRWLOptA,SRWLOptC,SRWLOptT,SRWLOptL,SRWLOptMirEl

#import SRW helpers functions
from wpg.useful_code.srwutils import AuxTransmAddSurfHeightProfileScaled

#import some helpers functions
from wpg.useful_code.wfrutils import calculate_fwhm_x, plot_wfront, calculate_fwhm_y,␣
↪print_beamline, get_mesh, plot_1d, plot_2d
from wpg.useful_code.wfrutils import propagate_wavefront

#Import base wavefront class
from wpg import Wavefront

#Gaussian beam generator
from wpg.generators import build_gauss_wavefront_xy
from wpg.beamline import Beamline
from wpg.optical_elements import Empty, Use_PP

plt.ion()
```

Define auxiliary functions

```
def _resample(wf, axis, data, x0, x1):
    if axis.lower()=='x':
        y = data[data.shape[0]/2,:]
        x = np.linspace(wf.params.Mesh.xMin, wf.params.Mesh.xMax, y.shape[0])
    elif axis.lower()=='y':
        y = data[:,data.shape[1]/2]
        x = np.linspace(wf.params.Mesh.yMin, wf.params.Mesh.yMax, y.shape[0])
    else:
        raise ValueError(
            'Wrong axis {}, should be "x" or "y"'.format(axis))

    if not x0 is None:
        xmin = x0
    else:
        xmin = x[0]

    if not x1 is None:
        xmax = x1
    else:
        xmax = x[-1]
```

(continues on next page)

(continued from previous page)

```
    x1 = np.linspace(xmin,xmax,len(y))
    y1 = np.interp(x1, x,y)
    return x1, y1

def intensity_cut(wf, axis, polarization, x0=None, x1=None):

    if polarization.lower()  == 'v' or polarization.lower() == 'vertical':
        pol = 'vertical'
    elif polarization.lower() == 'h' or polarization.lower() == 'horizontal':
        pol = 'horizontal'
    elif polarization.lower() == 't' or polarization.lower() == 'total':
        pol = 'total'
    else:
        raise ValueError(
            'Wrong polarization {}, should be "v" or "vertical"'+
            ' or "h" or "horizontal" or "t" or "total"'.format(polarization))

    data = wf.get_intensity(slice_number=0, polarization=pol)
    return _resample(wf, axis, data, x0, x1)

def phase_cut(wf, axis, polarization, x0=None, x1=None):

    if polarization.lower()  == 'v' or polarization.lower() == 'vertical':
        pol = 'vertical'
    elif polarization.lower() == 'h' or polarization.lower() == 'horizontal':
        pol = 'horizontal'
    else:
        raise ValueError(
            'Wrong polarization {}, should be "v" or "vertical" or "h" or "horizontal"
↪'.format(polarization))

    data = wf.get_phase(slice_number=0, polarization=pol)
    return _resample(wf, axis, data, x0, x1)
```

```
def calculate_source_fwhm(ekev, theta_fwhm):
    """
    Calculate source size from photon energy and FWHM angular divergence

    :param evev: Energy in keV
    :param theta_fwhm: theta_fwhm [units?]
    """
    wl = 12.39e-10/ekev
    k = 2 * np.sqrt(2*np.log(2))
    theta_sigma = theta_fwhm /k
    sigma0 = wl /(2*np.pi*theta_sigma)
    return sigma0*k

def calculate_theta_fwhm_cdr(ekev,qnC):
    """
    Calculate angular divergence using formula from XFEL CDR2011

    :param ekev: Energy in keV
    :param qnC: e-bunch charge, [nC]
    :return: theta_fwhm [units?]
    """
    theta_fwhm = (17.2 - 6.4 * np.sqrt(qnC))*1e-6/ekev**0.85
```

(continues on next page)

(continued from previous page)

```
    return theta_fwhm

def defineOPD(opTrErMirr, mdatafile, ncol, delim, Orient, theta, scale):
    """
    Define optical path difference (OPD) from mirror profile, i.e. ill the struct␣
↪opTrErMirr

    :params mdatafile: an ascii file with mirror profile data
    :params ncol: number of columns in the file
    :params delim: delimiter between numbers in an row, can be space (' '), tab '\t',␣
↪etc
    :params orient: mirror orientation, 'x' (horizontal) or 'y' (vertical)
    :params theta: incidence angle
    :params scale: scaling factor for the mirror profile
    """
    heightProfData = np.loadtxt(mdatafile).T
    AuxTransmAddSurfHeightProfileScaled(opTrErMirr, heightProfData, Orient, theta,␣
↪scale)
    plt.figure()
    plot_1d(heightProfData,'profile from ' + mdatafile,'x (m)', 'h (m)')
```

```
def defineEFM(orient,p,q,thetaEFM,theta0,lengthEFM):
    """
    A wrapper to a SRWL function SRWLOptMirEl() for defining a plane elliptical␣
↪focusing mirror propagator

    :param Orient:    mirror orientation, 'x' (horizontal) or 'y' (vertical)
    :param p:  the distance to two ellipsis centers
    :param q:  the distance to two ellipsis centers
    :param thetaEFM:  the design incidence angle in the center of the mirror
    :param theta0:    the "real" incidence angle in the center of the mirror
    :param lengthEFM: mirror length, [m]
    :return: the struct opEFM
    """
    if orient == 'x':     #horizontal plane ellipsoidal mirror
        opEFM = SRWLOptMirEl(_p=p, _q=q, _ang_graz=thetaEFM, _r_sag=1.e+40, _size_
↪tang=lengthEFM,
                             _nvx=np.cos(theta0), _nvy=0, _nvz=-np.sin(theta0), _tvx=-
↪np.sin(theta0), _tvy=0,
                              _x=0, _y=0, _treat_in_out=1)
    elif orient == 'y': #vertical plane ellipsoidal mirror
        opEFM = SRWLOptMirEl(_p=p, _q=q, _ang_graz=thetaEFM, _r_sag=1.e+40, _size_
↪tang=lengthEFM,
                             _nvx=0, _nvy=np.cos(theta0), _nvz=-np.sin(theta0), _tvx=0,
↪ _tvy=-np.sin(theta0),
                              _x=0, _y=0, _treat_in_out=1)
    else:
        raise TypeError('orient should be "x" or "y"')
    return opEFM
```

Defining initial wavefront and writing electric field data to h5-file

```
# #**********************Input Wavefront Structure and Parameters
print('*****defining initial wavefront and writing electric field data to h5-file...')
```

(continues on next page)

(continued from previous page)

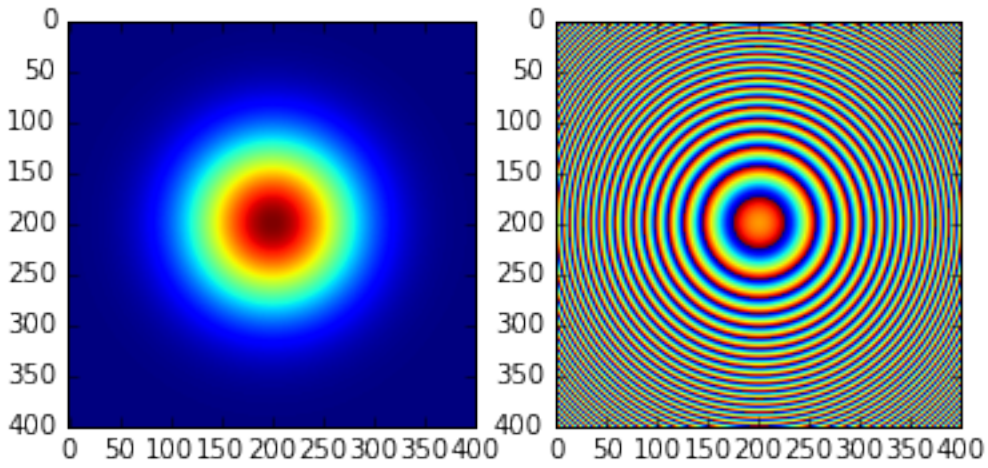
```
strInputDataFolder = 'data_common'  # input data sub-folder name
strOutputDataFolder = 'Tutorial_case_3'  # output data sub-folder name

#init Gauusian beam parameters
d2m1_sase1 = 246.5
d2m1_sase2 = 290.0
d2m1_sase3 = 281.0

d2hkb_sase1 = 929.6        # distance to nmKB's HFM
dHKB_foc_sase1 = 3.0       # nominal focal length for HFM KB
dVKB_foc_sase1 = 1.9       # nominal focal length for VFM KB
d2hkb_sase3 = 442.3
dHKB_foc_sase3 = 2.715     # nominal focal length for HFM KB
dVKB_foc_sase3 = 1.715     # nominal focal length for VFM KB


qnC = 0.1                      # e-bunch charge, [nC]
ekev_sase3 = 0.8
thetaOM_sase3 = 9.e-3
thetaKB_sase3 = 9.e-3
ekev_sase1 = 5.0
thetaOM_sase1 = 3.5e-3         #
thetaKB_sase1 = 3.5e-3

ekev = ekev_sase1
thetaOM = thetaOM_sase1
d2m1 = d2m1_sase1
d2hkb = d2hkb_sase1
thetaKB = thetaKB_sase1
dhkb_foc = dHKB_foc_sase1      # nominal focal length for HFM KB
dvkb_foc = dVKB_foc_sase1       # nominal focal length for VFM KB
dhkb_vkb = dhkb_foc - dvkb_foc          # distance between centers of HFM and VFM

z1 = d2m1
theta_fwhm = calculate_theta_fwhm_cdr(ekev,qnC)
k = 2*np.sqrt(2*np.log(2))
sigX = 12.4e-10*k/(ekev*4*np.pi*theta_fwhm)
print('waist_fwhm [um], theta_fwhms [urad]:', sigX*k*1e6, theta_fwhm*1e6)
#define limits
range_xy = theta_fwhm/k*z1*7. # sigma*7 beam size
npoints=400

#define unique filename for storing results
ip = np.floor(ekev)
frac = np.floor((ekev - ip)*1e3)
fname0 = 'g' + str(int(ip))+'_'+str(int(frac))+'kev'
print('save hdf5: '+fname0+'.h5')
ifname = os.path.join(strOutputDataFolder,fname0+'.h5')

#tiltX = theta_rms
#build SRW gauusian wavefront
wfr0=build_gauss_wavefront_xy(nx=npoints, ny=npoints, ekev=ekev,
                              xMin=-range_xy/2, xMax=range_xy/2,
                              yMin=-range_xy/2, yMax=range_xy/2,
                              sigX=sigX, sigY=sigX, d2waist=z1,
                              xoff=0, yoff=0, tiltX=0, tiltY=0)
```

(continues on next page)

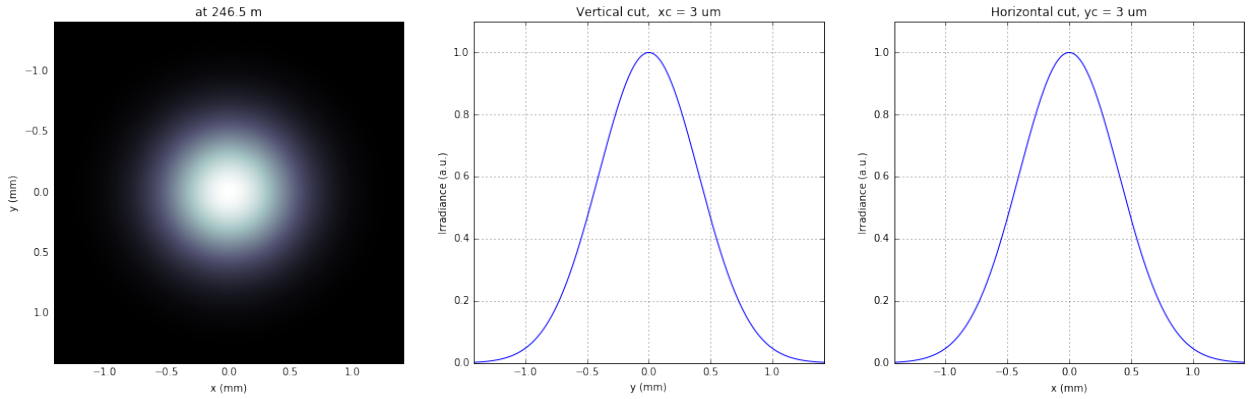
(continued from previous page)

```
#init WPG Wavefront helper class
mwf = Wavefront(wfr0)

#store wavefront to HDF5 file
mwf.store_hdf5(ifname)

#draw wavefront with common functions
plt.subplot(1,2,1)
plt.imshow(mwf.get_intensity(slice_number=0))
plt.subplot(1,2,2)
plt.imshow(mwf.get_phase(slice_number=0,polarization='horizontal'))
plt.show()

#draw wavefront with cuts
plot_wfront(mwf, title_fig='at '+str(z1)+' m',
            isHlog=False, isVlog=False,
            i_x_min=1e-5, i_y_min=1e-5, orient='x', onePlot=True)

plt.set_cmap('bone') #set color map, 'bone', 'hot', 'jet', etc
fwhm_x = calculate_fwhm_x(mwf)
print('FWHMx [mm], theta_fwhm [urad]:',fwhm_x*1e3,fwhm_x/z1*1e6)
```

```
*****defining initial wavefront and writing electric field data to h5-file...
waist_fwhm [um], theta_fwhms [urad]: 28.3217691481 3.86399794107
save hdf5: g5_0kev.h5
```




```
FWHMx [mm]: 0.943784566665
FWHMy [mm]: 0.943784566665
Coordinates of center, [mm]: 0.0035480622807 0.0035480622807
stepX, stepY [um]: 7.096124561394084 7.096124561394084

R-space
FWHMx [mm], theta_fwhm [urad]: 0.943784566665 3.82874063556
```



Defining optical beamline(s)

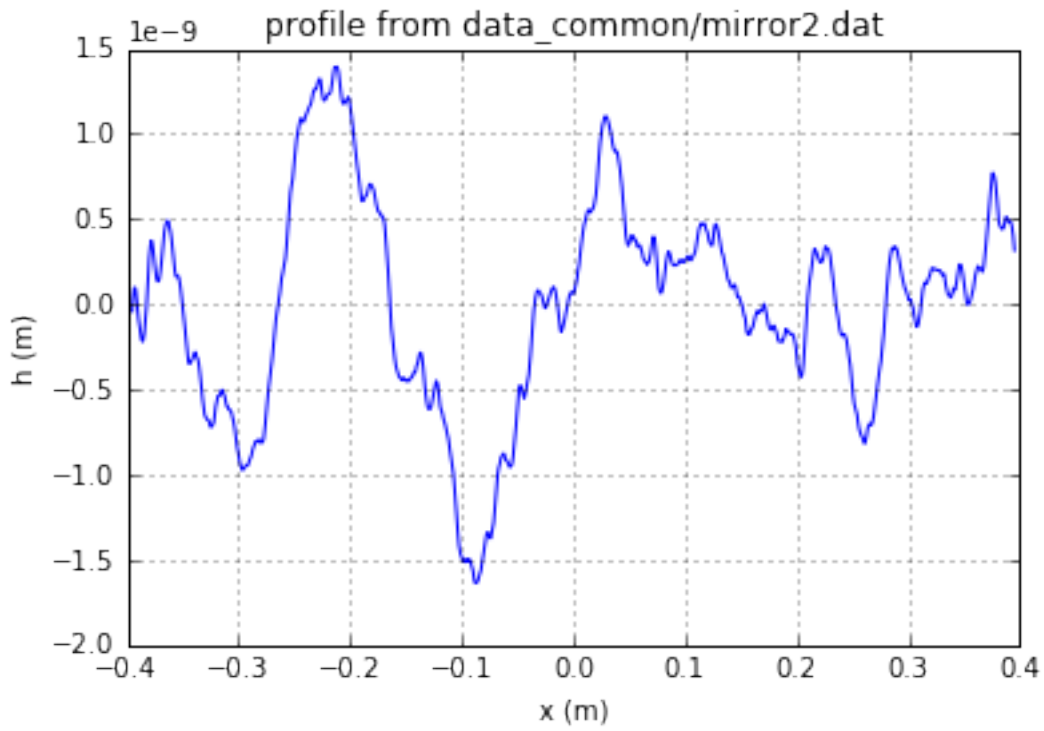
```
print('*****Defining optical beamline(s) ...')

z2 = d2hkb - d2m1

DriftM1_KB = SRWLOptD(z2) #Drift from first offset mirror (M1) to exp hall
horApM1 = 0.8*thetaOM
opApM1 = SRWLOptA('r', 'a', horApM1, range_xy)  # clear aperture of the Offset␣
↪Mirror(s)
horApKB = 0.8 * thetaKB # Aperture of the KB system, CA 0.8 m
opApKB = SRWLOptA('r', 'a', horApKB, horApKB)  # clear aperture of the Offset␣
↪Mirror(s)

#Wavefront Propagation Parameters:
#[0]:  Auto-Resize (1) or not (0) Before propagation
#[1]:  Auto-Resize (1) or not (0) After propagation
#[2]:  Relative Precision for propagation with Auto-Resizing (1. is nominal)
#[3]:  Allow (1) or not (0) for semi-analytical treatment of quadratic phase terms at␣
↪propagation
#[4]:  Do any Resizing on Fourier side, using FFT, (1) or not (0)
#[5]:  Horizontal Range modification factor at Resizing (1. means no modification)
#[6]:  Horizontal Resolution modification factor at Resizing
#[7]:  Vertical Range modification factor at Resizing
#[8]:  Vertical Resolution modification factor at Resizing
#[9]:  Type of wavefront Shift before Resizing (not yet implemented)
#[10]: New Horizontal wavefront Center position after Shift (not yet implemented)
#[11]: New Vertical wavefront Center position after Shift (not yet implemented)
#                  [ 0] [1] [2]  [3] [4] [5]  [6]  [7]  [8]  [9] [10] [11]
ppM1 =             [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppTrErM1 =         [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDriftM1_KB =     [ 0,  0, 1.0,  1,  0, 2.4, 1.8, 2.4, 1.8,  0,  0,   0]
ppApKB =           [ 0,  0, 1.0,  0,  0, 0.6, 8.0, 0.6, 4.0,  0,  0,   0]
ppHKB =            [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppTrErHKB =        [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDrift_HKB_foc =  [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDrift_KB =       [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppVKB =            [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppTrErVKB =        [ 0,  0, 1.0,  0,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
ppDrift_foc =      [ 0,  0, 1.0,  1,  0, 1.0, 1.0, 1.0, 1.0,  0,  0,   0]
#ppFin  =           [ 0,  0, 1.0,  0,  0, 0.05,5.0, 0.05,5.0,  0,  0,   0]
ppFin =            [ 0,  0, 1.0,  0,  1, .01, 20.0, .01, 20.0,  0,  0,   0]
```

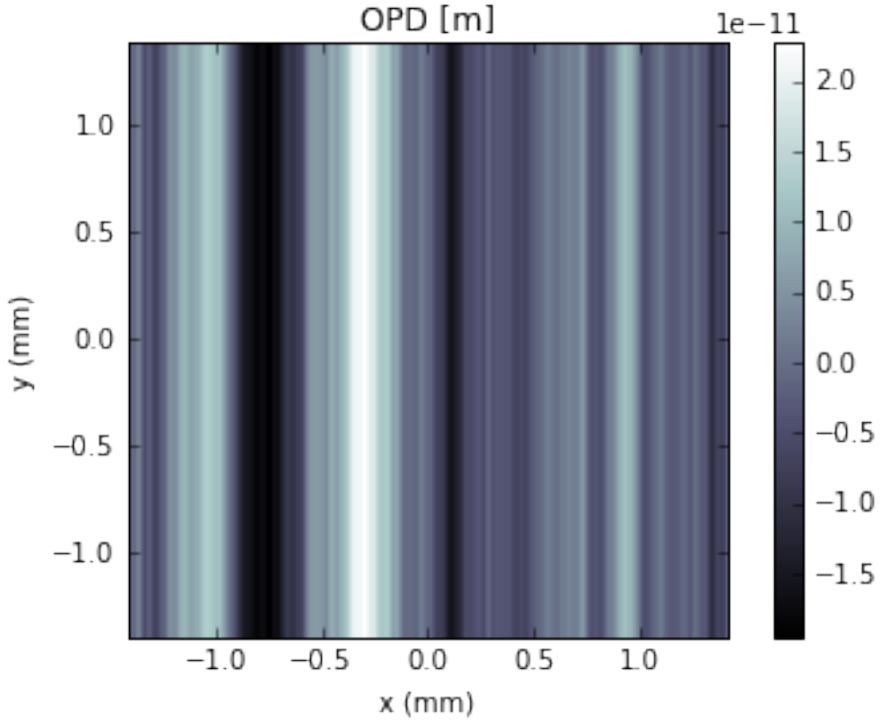
(continues on next page)

(continued from previous page)

```
optBL0 = SRWLOptC([opApM1,  DriftM1_KB],
                  [ppM1,ppDriftM1_KB])

scale = 2     #5 mirror profile scaling factor
print('*****HOM1 data for BL1 beamline ')
opTrErM1 = SRWLOptT(1500, 100, horApM1, range_xy)
#defineOPD(opTrErM1, os.path.join(strInputDataFolder,'mirror1.dat'), 2, '\t', 'x',␣
↪thetaOM, scale)
defineOPD(opTrErM1, os.path.join(strInputDataFolder,'mirror2.dat'), 2, ' ', 'x',␣
↪thetaOM, scale)
opdTmp=np.array(opTrErM1.arTr)[1::2].reshape(opTrErM1.mesh.ny,opTrErM1.mesh.nx)
plt.figure()
plot_2d(opdTmp, opTrErM1.mesh.xStart*1e3,opTrErM1.mesh.xFin*1e3,opTrErM1.mesh.
↪yStart*1e3,opTrErM1.mesh.yFin*1e3,
        'OPD [m]', 'x (mm)', 'y (mm)')

optBL1 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB],
                  [ppM1,ppTrErM1,ppDriftM1_KB])

dhkb_vkb = dhkb_foc - dvkb_foc          # distance between centers of HFM and VFM
d2vkb = d2hkb +  dhkb_vkb
vkbfoc =  1. /(1./dvkb_foc + 1. / d2vkb) # for thin lens approx
hkbfoc =  1. /(1./dhkb_foc + 1. / d2hkb) # for thin lens approx

z3 = dhkb_vkb
z4 = vkbfoc #distance to focal plane

#HKB = SRWLOptMirEl(_p=d2hkb, _q=dhkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=cos(thetaKB), _nvy=0, _nvz=-sin(thetaKB), _tvx=-sin(thetaKB), _
↪tvy=0, _x=0, _y=0, _treat_in_out=1) #HKB Ellipsoidal Mirror
#VKB = SRWLOptMirEl(_p=d2vkb, _q=dvkb_foc, _ang_graz=thetaKB, _r_sag=1.e+40, _size_
↪tang=0.85, _nvx=0, _nvy=cos(thetaKB), _nvz=-sin(thetaKB), _tvx=0, _tvy=-
↪sin(thetaKB), _x=0, _y=0, _treat_in_out=1) #VKB Ellipsoidal Mirror
#HKB = SRWLOptL(hkbfoc) #HKB as Thin Lens
#VKB = SRWLOptL(1e23,vkbfoc) #VKB as Thin Lens
HKB = defineEFM('x', d2hkb, dhkb_foc, thetaKB, thetaKB, 0.85) #HKB Ellipsoidal Mirror
VKB = defineEFM('y', d2vkb, dvkb_foc, thetaKB, thetaKB, 0.85) #VKB Ellipsoidal Mirror
Drift_KB  = SRWLOptD(z3)
Drift_foc = SRWLOptD(z4)
optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,  Drift_
↪foc],
                  [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,ppDrift_
↪foc,ppFin])
```

```
*****Defining optical beamline(s) ...
*****HOM1 data for BL1 beamline
```





Propagating through BL1 beamline. Imperfect mirror, at KB aperture

```
print('*****Imperfect HOM mirror, at KB aperture')
```

(continues on next page)

(continued from previous page)

```
bPlotted = False
isHlog = True
isVlog = False
bSaved = True
optBL = optBL1
strBL = 'bl1'
pos_title = 'at exp hall wall'
print('*****setting-up optical elements, beamline:', strBL)
bl = Beamline(optBL)
print(bl)

if bSaved:
    out_file_name = os.path.join(strOutputDataFolder, fname0+'_'+strBL+'.h5')
    print('save hdf5:', out_file_name)
else:
    out_file_name = None

startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Imperfect HOM mirror, at KB aperture
*****setting-up optical elements, beamline: bl1
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1500
             ny = 100
             xFin = 0.0014000000000000002
             xStart = -0.0014000000000000002
             yFin = 0.00141567685
             yStart = -0.00141567685
             zStart = 0
```

(continues on next page)

(continued from previous page)

```
Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0


save hdf5: Tutorial_case_3/g5_0kev_bl1.h5
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1500
             ny = 100
             xFin = 0.0014000000000000002
             xStart = -0.0014000000000000002
             yFin = 0.00141567685
             yStart = -0.00141567685
             zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0


*****reading wavefront from h5 file...
R-space
nx   400  range_x [-1.4e+00, 1.4e+00] mm
ny   400  range_y [-1.4e+00, 1.4e+00] mm
*****propagating wavefront (with resizing)...
save hdf5: Tutorial_case_3/g5_0kev_bl1.h5
done
```

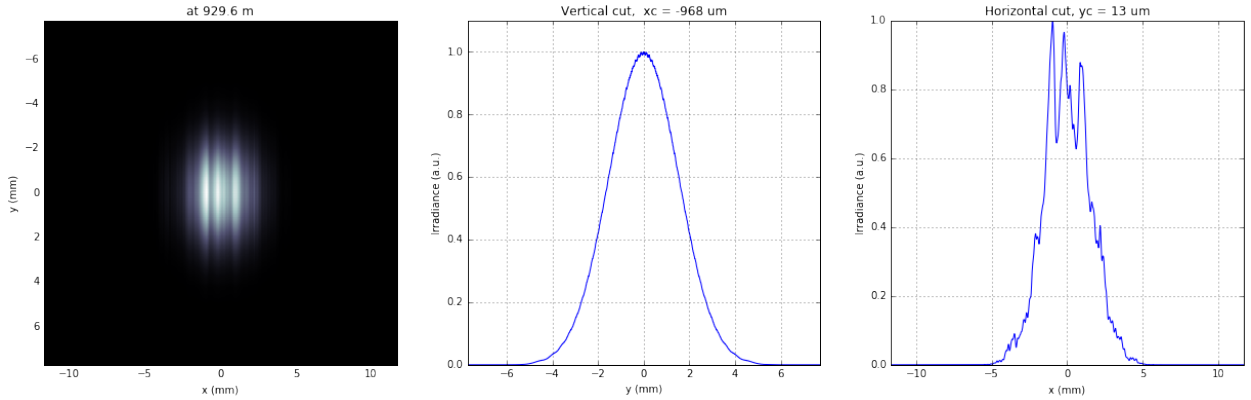
(continues on next page)

(continued from previous page)

```
propagation lasted: 0.1 min
```

```
print('*****Imperfect HOM mirror, at KB aperture')
plot_wfront(mwf, 'at '+str(z1+z2)+' m',False, False, 1e-5,1e-5,'x', True)
#plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [mm], theta_fwhm [urad]:',calculate_fwhm_x(mwf)*1e3,calculate_fwhm_
↪x(mwf)/(z1+z2)*1e6)
print('FWHMy [mm], theta_fwhm [urad]:',calculate_fwhm_y(mwf)*1e3,calculate_fwhm_
↪y(mwf)/(z1+z2)*1e6)
```

```
*****Imperfect HOM mirror, at KB aperture
FWHMx [mm]: 2.89758221191
FWHMy [mm]: 3.55209077523
Coordinates of center, [mm]: -0.968117421268 0.0134209978913
stepX, stepY [um]: 13.540103793954389 8.94733192752122

R-space
FWHMx [mm], theta_fwhm [urad]: 2.89758221191 3.11702045171
FWHMy [mm], theta_fwhm [urad]: 3.55209077523 3.8210959286
```



Propagating through BL2 beamline. Focused beam: perfect KB

```
print('*****Focused beam: perfect KB')
bSaved = False
z3 = dhkb_vkb
z4 = dvkb_foc
z4 = vkbfoc #distance to focal plane

#HKB = SRWLOptL(hkbfoc) #HKB as Thin Lens
#VKB = SRWLOptL(1e23,vkbfoc) #VKB as Thin Lens
#HKB = defineEFM('x', d2hkb, dhkb_foc, thetaKB, thetaKB, 0.85) #HKB Ellipsoidal Mirror
#VKB = defineEFM('y', d2vkb, dvkb_foc, thetaKB, thetaKB, 0.85) #VKB Ellipsoidal Mirror
Drift_foc = SRWLOptD(dvkb_foc)
#optBL2 = SRWLOptC([opApM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,  Drift_foc],
#                  [ppM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,ppDrift_foc,
↪ppFin])
optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,  Drift_
↪foc],
```

(continues on next page)

(continued from previous page)

```
                    [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,ppDrift_
↪foc])
optBL = optBL2
strBL = 'bl2'
pos_title = 'at sample position'
print('*****setting-up optical elements, beamline:', strBL)
bl = Beamline(optBL)
bl.append(Empty(), Use_PP(zoom=0.02, sampling=5.0))
print(bl)

if bSaved:
    out_file_name = os.path.join(strOutputDataFolder, fname0+'_'+strBL+'.h5')
    print('save hdf5:', out_file_name)
else:
    out_file_name = None

startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Focused beam: perfect KB
*****setting-up optical elements, beamline: bl2
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1500
             ny = 100
             xFin = 0.0014000000000000002
             xStart = -0.0014000000000000002
             yFin = 0.00141567685
             yStart = -0.00141567685
             zStart = 0
```

(continues on next page)

(continued from previous page)

```
Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028000000000000004
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.0035
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0.999993875006
    nvy = 0
    nvz = -0.00349999285417
    p = 929.6
    q = 3.0
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = -0.00349999285417
    tvy = 0
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.1
    treat = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
```

(continues on next page)

(continued from previous page)

```
    Fx = 0
    Fy = 0
    angGraz = 0.0035
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0
    nvy = 0.999993875006
    nvz = -0.00349999285417
    p = 930.7
    q = 1.9
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = 0
    tvy = -0.00349999285417
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.9
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 0, 0.02, 5.0, 0.02, 5.0, 0, 0, 0]


Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
```

(continues on next page)

(continued from previous page)

```
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.00141567685
            yStart = -0.00141567685
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028000000000000004
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.0035
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0.999993875006
    nvy = 0
    nvz = -0.00349999285417
    p = 929.6
    q = 3.0
    radSag = 1e+40
    reflAngFin = 0
```

(continues on next page)

(continued from previous page)

```
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = -0.00349999285417
    tvy = 0
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.1
    treat = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.0035
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0
    nvy = 0.999993875006
    nvz = -0.00349999285417
    p = 930.7
    q = 1.9
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = 0
    tvy = -0.00349999285417
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.9
```

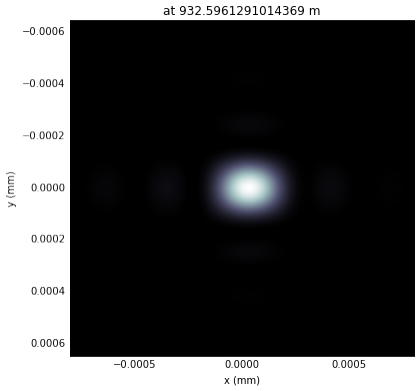
(continues on next page)

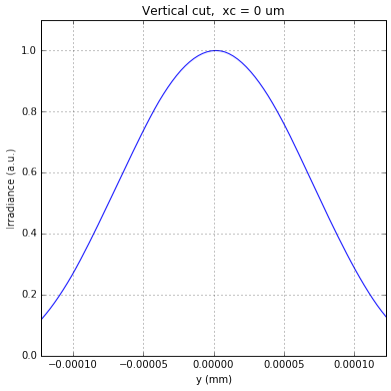
(continued from previous page)

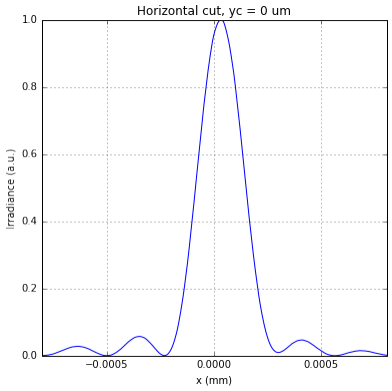
```
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 0, 0.02, 5.0, 0.02, 5.0, 0, 0, 0]


*****reading wavefront from h5 file...
R-space
nx    400   range_x [-1.4e+00, 1.4e+00] mm
ny    400   range_y [-1.4e+00, 1.4e+00] mm
*****propagating wavefront (with resizing)...
done
propagation lasted: 1.8 min
```

```
print('*****Focused beam: Focused beam: perfect KB')
bOnePlot = True
isHlog = False
isVlog = False
plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m',isHlog, isVlog, 1e-5,1e-5,'x', bOnePlot)
#plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [um], FWHMy [um]:',calculate_fwhm_x(mwf)*1e6,calculate_fwhm_y(mwf)*1e6)
```

```
*****Focused beam: Focused beam: perfect KB
FWHMx [mm]: 0.000234193434076
FWHMy [mm]: 0.000149243216593
Coordinates of center, [mm]: 2.99999853568e-05 1.55461683951e-06
stepX, stepY [um]: 0.0019354829262443274 0.0031092336790287185

R-space
FWHMx [um], FWHMy [um]: 0.234193434076 0.149243216593
```







Defining OPD for HKB and VKB

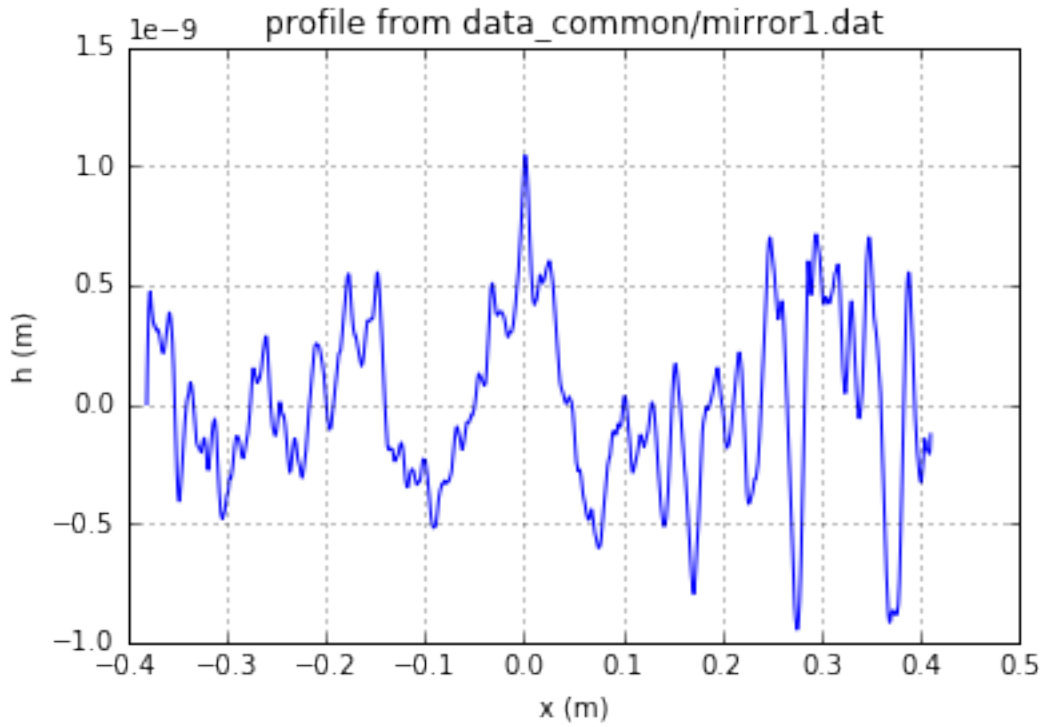
```
print('*****HKB and VKB OPD from data  profiles ')
scale = 2 #scaling factor of mirror
opTrErHKB = SRWLOptT(1500, 100, horApKB, horApKB)
defineOPD(opTrErHKB, os.path.join(strInputDataFolder,'mirror1.dat'), 2, '\t', 'x',␣
↪thetaOM, scale)
```

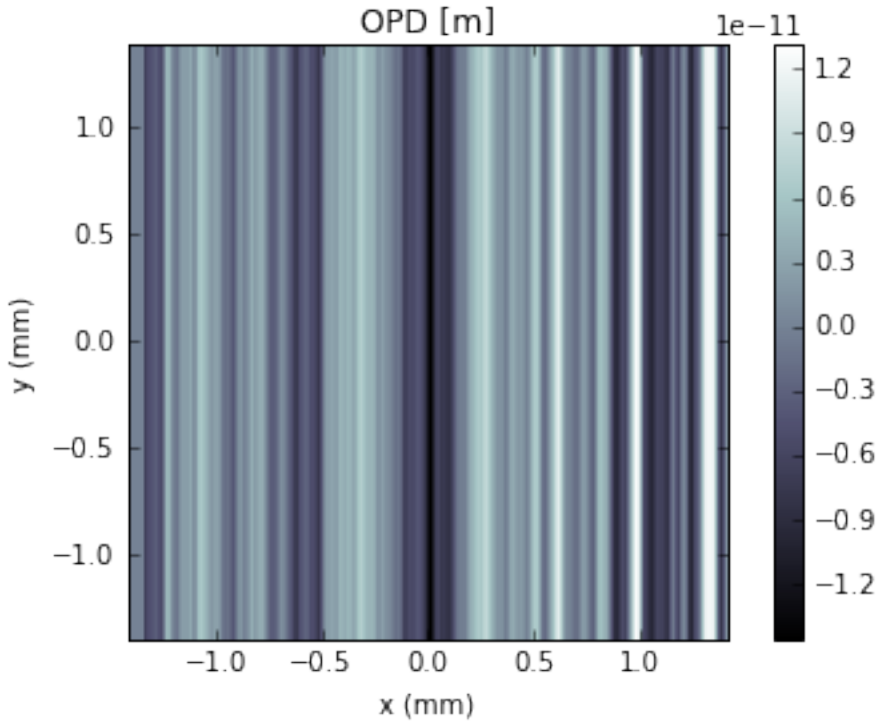
(continues on next page)

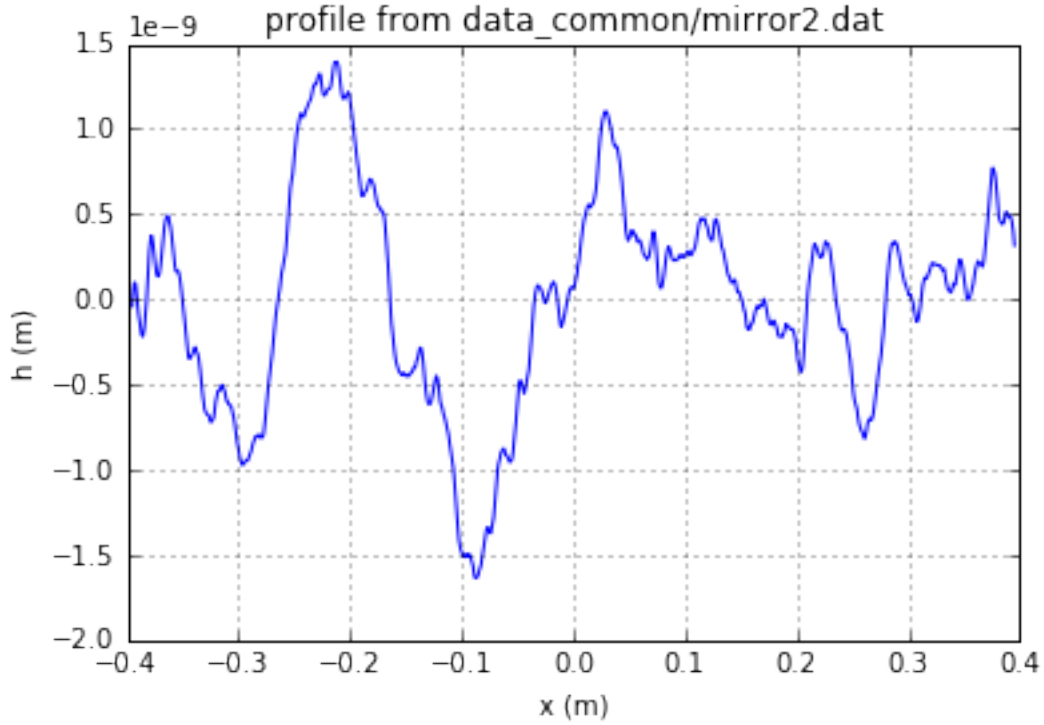
(continued from previous page)

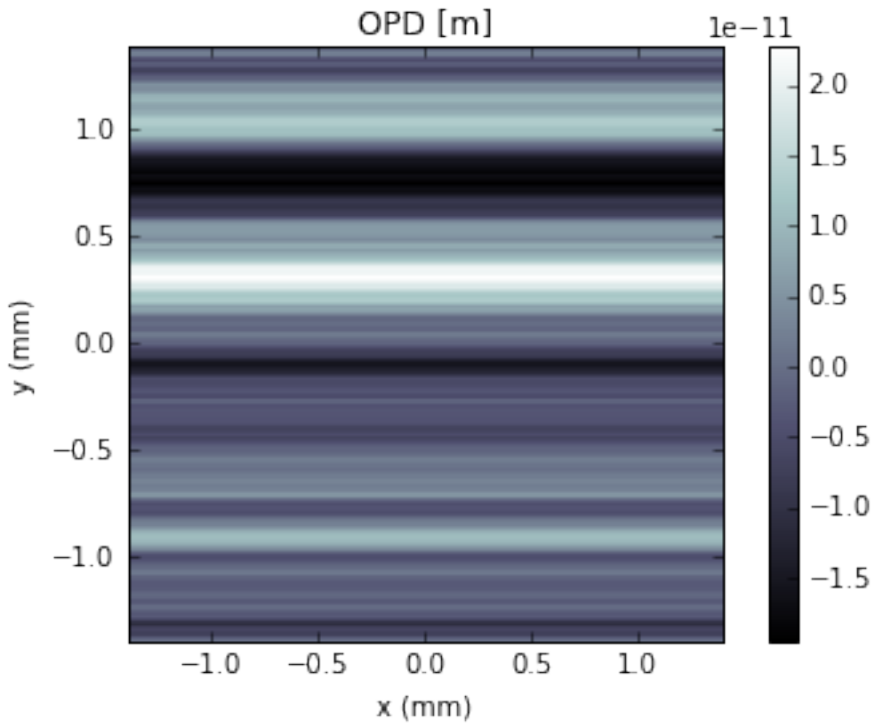
```
opdTmp=np.array(opTrErHKB.arTr)[1::2].reshape(opTrErHKB.mesh.ny,opTrErHKB.mesh.nx)
plt.figure()
plot_2d(opdTmp, opTrErM1.mesh.xStart*1e3,opTrErM1.mesh.xFin*1e3,opTrErM1.mesh.
↪yStart*1e3,opTrErM1.mesh.yFin*1e3,
        'OPD [m]', 'x (mm)', 'y (mm)')
print('*****VKB data  ')
opTrErVKB = SRWLOptT(100, 1500, horApKB, horApKB)
defineOPD(opTrErVKB, os.path.join(strInputDataFolder,'mirror2.dat'), 2, ' ', 'y', ␣
↪thetaOM, scale)
opdTmp=np.array(opTrErVKB.arTr)[1::2].reshape(opTrErVKB.mesh.ny,opTrErVKB.mesh.nx)
plt.figure()
plot_2d(opdTmp, opTrErVKB.mesh.xStart*1e3,opTrErVKB.mesh.xFin*1e3,opTrErVKB.mesh.
↪yStart*1e3,opTrErVKB.mesh.yFin*1e3,
        'OPD [m]', 'x (mm)', 'y (mm)')
```

```
*****HKB and VKB OPD from data  profiles
*****VKB data
```




OPD [m]
1e−11
1.2
0.9
0.6
0.3
0.0
−0.3
−0.6
−0.9
−1.2
1.0
0.5
0.0
−0.5
−1.0
y (mm)
−1.0
−0.5
0.0
0.5
1.0
x (mm)


profile from data_common/mirror2.dat
1e−9
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
h (m)
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
x (m)



Propagating through BL2 beamline. Focused beam: imperfect KB

```
print('*****Focused beam on focus: imperfect KB')
z3 = dhkb_vkb
z4 = dvkb_foc #distance to focal plane
#z4 = vkbfoc  #focus distance of lens

HKB = SRWLOptL(hkbfoc) #HKB as Thin Lens
#VKB = SRWLOptL(1e23,vkbfoc) #VKB as Thin Lens
#HKB = defineEFM('x', d2hkb, dhkb_foc, thetaKB, thetaKB, 0.85) #HKB Ellipsoidal Mirror
#VKB = defineEFM('y', d2vkb, dvkb_foc, thetaKB, thetaKB, 0.85) #VKB Ellipsoidal Mirror
Drift_foc = SRWLOptD(z4)
optBL2 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,opTrErHKB,  Drift_KB, ␣
↪VKB,opTrErVKB,  Drift_foc],
                    [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppTrErM1,ppDrift_KB,
↪ppVKB,ppTrErM1, ppDrift_foc])
optBL = optBL2
strBL = 'bl2'
pos_title = 'at sample position'
print('*****setting-up optical elements, beamline:', strBL)
bl = Beamline(optBL)
bl.append(Empty(), Use_PP(zoom=0.02, sampling=5.0))
print(bl)

if bSaved:
    out_file_name = os.path.join(strOutputDataFolder, fname0+'_'+strBL+'.h5')
    print('save hdf5:', out_file_name)
else:
    out_file_name = None
```

(continues on next page)

(continued from previous page)

```
startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Focused beam on focus: imperfect KB
*****setting-up optical elements, beamline: bl2
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.00141567685
            yStart = -0.00141567685
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028000000000000004
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Thin Lens
```

(continues on next page)

(continued from previous page)

```
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 2.9903495603688612
    Fy = 1e+23
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1500
             ny = 100
             xFin = 0.0014000000000000002
             xStart = -0.0014000000000000002
             yFin = 0.0014000000000000002
             yStart = -0.0014000000000000002
             zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.1
    treat = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.0035
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0
    nvy = 0.999993875006
    nvz = -0.00349999285417
    p = 930.7
    q = 1.9
    radSag = 1e+40
```

(continues on next page)

(continued from previous page)

```
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = 0
    tvy = -0.00349999285417
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 100
            ny = 1500
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.0014000000000000002
            yStart = -0.0014000000000000002
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.9
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 0, 0.02, 5.0, 0.02, 5.0, 0, 0, 0]


Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
```

(continues on next page)

(continued from previous page)

```
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.00141567685
            yStart = -0.00141567685
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028000000000000004
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Thin Lens
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 2.9903495603688612
    Fy = 1e+23
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
```

(continues on next page)

(continued from previous page)

```
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.0014000000000000002
            yStart = -0.0014000000000000002
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.1
    treat = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.0035
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0
    nvy = 0.999993875006
    nvz = -0.00349999285417
    p = 930.7
    q = 1.9
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = 0
    tvy = -0.00349999285417
```

(continues on next page)

(continued from previous page)

```
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 100
            ny = 1500
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.0014000000000000002
            yStart = -0.0014000000000000002
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.9
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 0, 0.02, 5.0, 0.02, 5.0, 0, 0, 0]


*****reading wavefront from h5 file...
R-space
nx   400  range_x [-1.4e+00, 1.4e+00] mm
ny   400  range_y [-1.4e+00, 1.4e+00] mm
*****propagating wavefront (with resizing)...
done
propagation lasted: 1.8 min
```

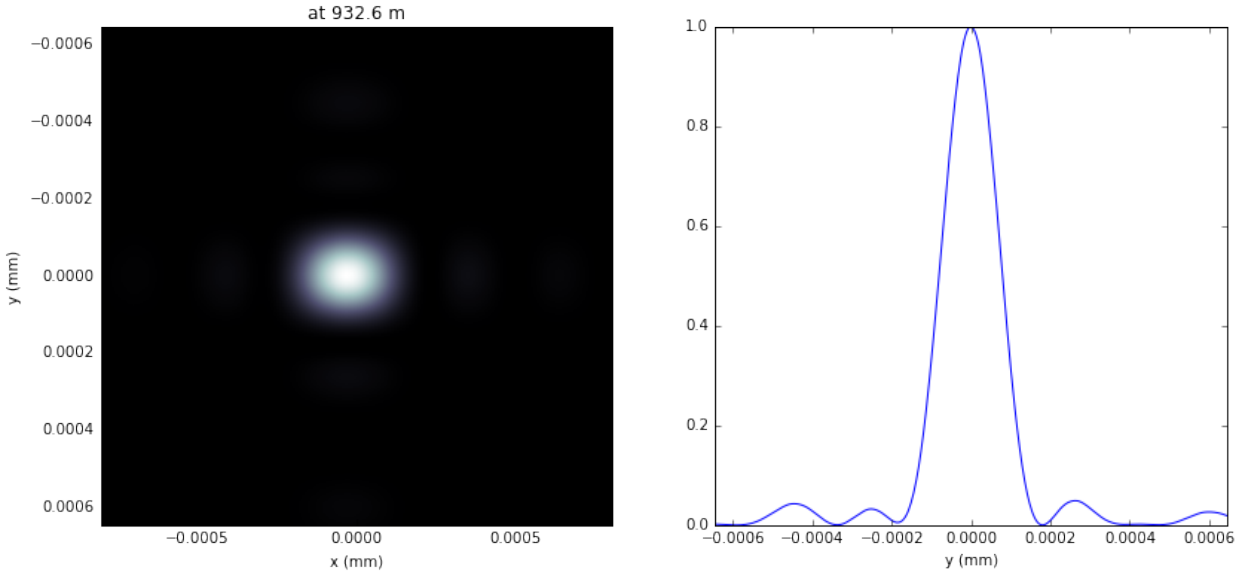
```
print('*****Focused beam behind focus: imperfect KB')
bOnePlot= True
isHlog = False
isVlog = False
bSaved = True
try:
    plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m',isHlog, isVlog, 1e-3,1e-3,'x',␣
↪bOnePlot)
```

(continues on next page)

(continued from previous page)

```
except ValueError as e:
    print(e)
#plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [um], FWHMy [um]:',calculate_fwhm_x(mwf)*1e6,calculate_fwhm_y(mwf)*1e6)
```

```
*****Focused beam behind focus: imperfect KB
FWHMx [mm]: 0.000236128917001
FWHMy [mm]: 0.00015857091763
Coordinates of center, [mm]: -3.38709512091e-05 -1.55461683951e-06
stepX, stepY [um]: 0.001935482926236052 0.0031092336790287185

R-space
zero-size array to reduction operation minimum which has no identity
FWHMx [um], FWHMy [um]: 0.236128917001 0.15857091763
```



Propagating through BL4 beamline. Focused beam: perfect KB

```
print('*****Focused beam behind focus: misaligned perfect KB')
z3 = dhkb_vkb
#z4 = dvkb_foc #distance to focal plane
theta0 = thetaKB + 50e-6
p = d2hkb
q = dhkb_foc
R0 = 2./(1./p+1./q)/thetaKB
q_mis = 1./(2/(R0*theta0)-1./p)
offset = q_mis - q #79e-3 if \Delta\theta 10 urad#0. if thetaKB0 = thetaKB
print('Distance to focus, without and with misalignment:', q,q_mis, 'm')
z4 = dvkb_foc+(q_mis-q) #distance to focal plane
Drift_foc = SRWLOptD(z4)
HKB = defineEFM('x', d2hkb, dhkb_foc, thetaKB, theta0, 0.85) #HKB Ellipsoidal Mirror
VKB = defineEFM('y', d2vkb, dvkb_foc, thetaKB, thetaKB, 0.85) #VKB Ellipsoidal Mirror
optBL4 = SRWLOptC([opApM1,opTrErM1,  DriftM1_KB,opApKB, HKB,   Drift_KB,  VKB,  Drift_
↪foc],
```

(continues on next page)

(continued from previous page)

```
                    [ppM1,ppTrErM1,ppDriftM1_KB,ppApKB,ppHKB,ppDrift_KB,ppVKB,␣
↪ppDrift_foc])
optBL = optBL4
strBL = 'bl4'
pos_title = 'at new focal plane, misalidned KB angle:'+str(theta0)
print('*****setting-up optical elements, beamline:', strBL)
bl = Beamline(optBL)
bl.append(Empty(), Use_PP(zoom_h=0.2, sampling_h=5.0))
print(bl)

if bSaved:
    out_file_name = os.path.join(strOutputDataFolder, fname0+'_'+strBL+'.h5')
    print('save hdf5:', out_file_name)
else:
    out_file_name = None

startTime = time.time()
mwf = propagate_wavefront(ifname, bl,out_file_name)
print('propagation lasted:', round((time.time() - startTime) / 6.) / 10., 'min')
```

```
*****Focused beam behind focus: misaligned perfect KB
Distance to focus, without and with misalignment: 3.0 3.0429974334936767 m
*****setting-up optical elements, beamline: bl4
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
             arSurf = None
             eFin = 0
             eStart = 0
             hvx = 1
             hvy = 0
             hvz = 0
             ne = 1
             nvx = 0
             nvy = 0
             nvz = 1
             nx = 1500
             ny = 100
             xFin = 0.0014000000000000002
             xStart = -0.0014000000000000002
             yFin = 0.00141567685
             yStart = -0.00141567685
             zStart = 0
```

(continues on next page)

(continued from previous page)

```
Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028000000000000004
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.0035
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0.999993698757
    nvy = 0
    nvz = -0.00354999254353
    p = 929.6
    q = 3.0
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = -0.00354999254353
    tvy = 0
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.1
    treat = 0

Optical Element: Mirror: Ellipsoid
```

(continues on next page)

(continued from previous page)

```
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.0035
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0
    nvy = 0.999993875006
    nvz = -0.00349999285417
    p = 930.7
    q = 1.9
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = 0
    tvy = -0.00349999285417
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.9429974334936766
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 0, 0.2, 5.0, 1.0, 1.0, 0, 0, 0]


save hdf5: Tutorial_case_3/g5_0kev_b14.h5
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028313537
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
```

(continues on next page)

(continued from previous page)

```
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.0014000000000000002
            xStart = -0.0014000000000000002
            yFin = 0.00141567685
            yStart = -0.00141567685
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 2.4, 1.8, 2.4, 1.8, 0, 0, 0]
    L = 683.1
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 0.6, 8.0, 0.6, 4.0, 0, 0, 0]
    Dx = 0.0028000000000000004
    Dy = 0.0028000000000000004
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.0035
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0.999993698757
    nvy = 0
    nvz = -0.00354999254353
    p = 929.6
    q = 3.0
```

(continues on next page)

(continued from previous page)

```
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = -0.00354999254353
    tvy = 0
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.1
    treat = 0

Optical Element: Mirror: Ellipsoid
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 0
    Fy = 0
    angGraz = 0.0035
    apShape = r
    arRefl = array of size 2
    ds = 1
    dt = 0.85
    extIn = 0
    extOut = 0
    meth = 2
    nps = 500
    npt = 500
    nvx = 0
    nvy = 0.999993875006
    nvz = -0.00349999285417
    p = 930.7
    q = 1.9
    radSag = 1e+40
    reflAngFin = 0
    reflAngScaleType = lin
    reflAngStart = 0
    reflNumAng = 1
    reflNumComp = 1
    reflNumPhEn = 1
    reflPhEnFin = 1000.0
    reflPhEnScaleType = lin
    reflPhEnStart = 1000.0
    treatInOut = 1
    tvx = 0
    tvy = -0.00349999285417
    x = 0
    y = 0

Optical Element: Drift Space
```

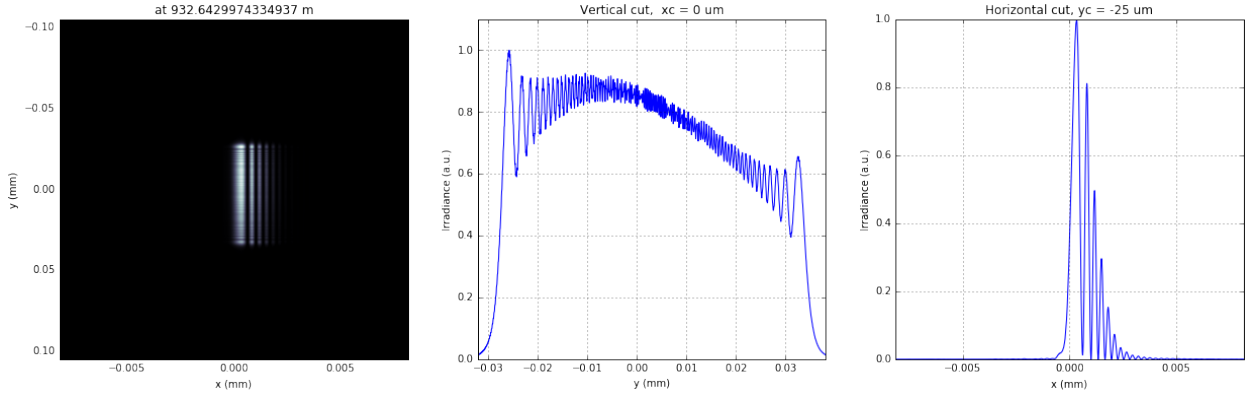
(continues on next page)

(continued from previous page)

```
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 1.9429974334936766
    treat = 0

Optical element: Empty.
    This is empty propagator used for sampling and zooming wavefront

Prop. parameters = [0, 0, 1.0, 0, 0, 0.2, 5.0, 1.0, 1.0, 0, 0, 0]


*****reading wavefront from h5 file...
R-space
nx    400  range_x [-1.4e+00, 1.4e+00] mm
ny    400  range_y [-1.4e+00, 1.4e+00] mm
*****propagating wavefront (with resizing)...
save hdf5: Tutorial_case_3/g5_0kev_bl4.h5
done
propagation lasted: 2.1 min
```

```
print('*****Focused beam behind focus: misaligned ideal KB')
pos_title = 'at new focal plane, misalidned KB angle:'+str(theta0)
bOnePlot= True
isHlog = False
isVlog = False
bSaved = True
plot_wfront(mwf, 'at '+str(z1+z2+z3+z4)+' m',isHlog, isVlog, 1e-3,1e-3,'x', bOnePlot)
#plt.set_cmap('bone') #set color map, 'bone', 'hot', etc
plt.axis('tight')
print('FWHMx [um], FWHMy [um]:',calculate_fwhm_x(mwf)*1e6,calculate_fwhm_y(mwf)*1e6)
```

```
*****Focused beam behind focus: misaligned ideal KB
FWHMx [mm]: 0.000825975119108
FWHMy [mm]: 0.0608721123876
Coordinates of center, [mm]: 0.000336472049708 -0.0257885588165
stepX, stepY [um]: 0.0019619361498999575 0.050516275840308565

R-space
FWHMx [um], FWHMy [um]: 0.825975119108 60.8721123876
```




```
print('at new focal plane, misalidned KB angle:'+str(theta0))
x,y = intensity_cut(mwf, axis='x', polarization='v', x0=-2.5e-6, x1=5.e-6)
```

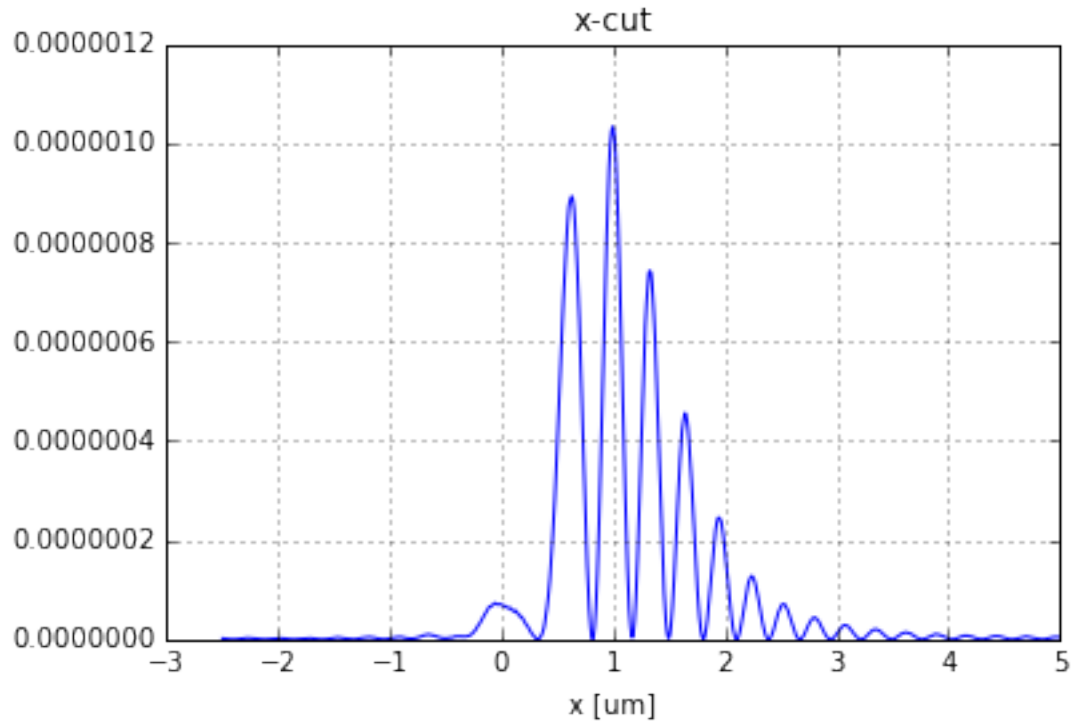
(continues on next page)

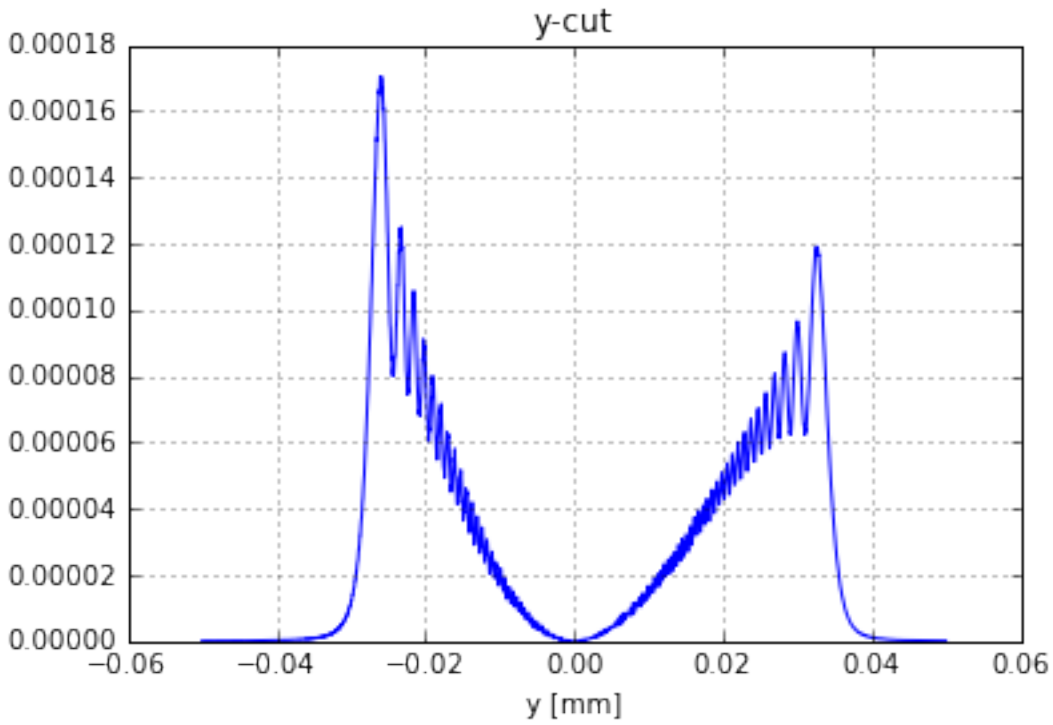
(continued from previous page)

```
plt.figure()
plt.title('x-cut')
plt.plot(x*1e6,y) # x in [um]
plt.grid(True)
plt.xlabel('x [um]')

x,y = intensity_cut(mwf, axis='y', polarization='v', x0=-0.05e-3, x1=0.05e-3)
plt.figure()
plt.title('y-cut')
plt.plot(x*1e3,y) # x in [mm]
plt.grid(True)
plt.xlabel('y [mm]')
```

```
at new focal plane, misalidned KB angle:0.00355
```

```
<matplotlib.text.Text at 0x7fca6c343da0>
```





1.11 XFEL beamlines examples

This section contain examples of simulation of real beamlines.

1.11.1 S1 SPB Day1 simplified beamline

```
%matplotlib inline
```

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals
```

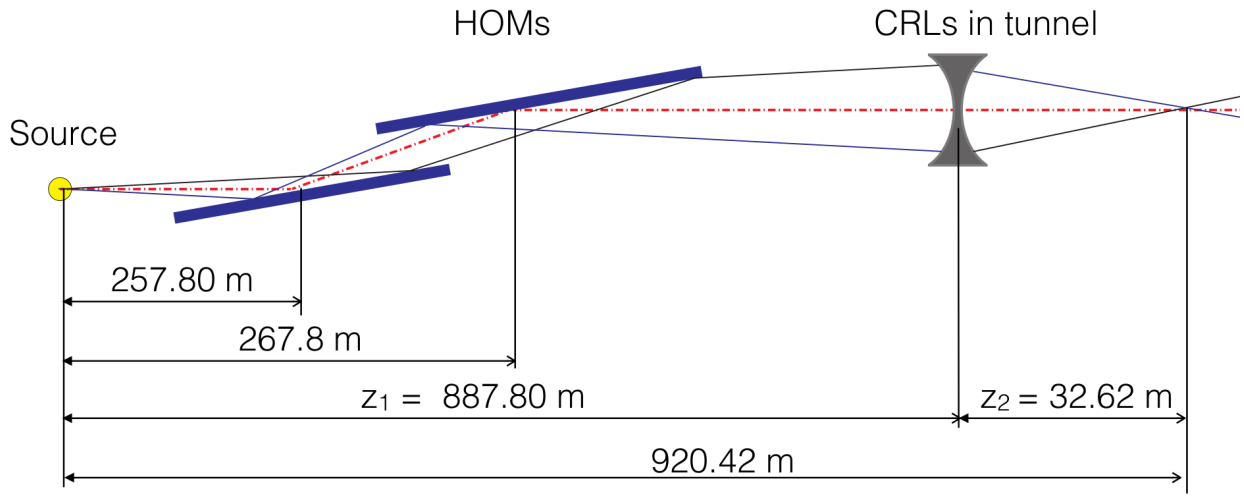
```
import os
import sys


# wpg_path = '/afs/desy.de/group/exfel/software/wpg/latest/' # DESY installation
wpg_path = os.path.join('..','..','..')
sys.path.insert(0, wpg_path)

import numpy as np
import pylab as plt


from wpg import Wavefront, Beamline
from wpg.optical_elements import Aperture, Drift, CRL, Empty, Use_PP
```

(continues on next page)

(continued from previous page)

```
from wpg.generators import build_gauss_wavefront

from wpg.srwlib import srwl

from wpg.wpg_uti_exfl import calculate_theta_fwhm_cdr_s1
from wpg.wpg_uti_wf import calculate_fwhm, averaged_intensity, look_at_q_space, plot_
↪t_wf
from wpg.wpg_uti_oe import show_transmission
```

```
from IPython.display import Image
Image(filename='day1_1.png')
```




```
%%file bl_S1_SPB_day1_simplified.py


def get_beamline():
    from wpg import Beamline
    from wpg.optical_elements import Aperture, Drift, CRL, Empty, Use_PP
    #S1 beamline layout
    ### Geometry ###
    src_to_hom1 = 257.8 # Distance source to HOM 1 [m]
    src_to_hom2 = 267.8 # Distance source to HOM 2 [m]
    src_to_crl = 887.8  # Distance source to CRL [m]
#     src_to_exp = 920.42 # Distance source to experiment [m]
    z0 = src_to_hom1

    # Drift to focus aperture
    #crl_to_exp_drift = Drift( src_to_exp - src_to_crl )
    z = 34.0
    #define distances, angles, etc
    #...
    #Incidence angle at HOM
    theta_om = 3.6e-3       # [rad]

    om_mirror_length = 0.8 # [m]
    om_clear_ap = om_mirror_length*theta_om
```

(continues on next page)

(continued from previous page)

```
    #define the beamline:
    bl0 = Beamline()
    zoom=1

    # Define HOM1.
    aperture_x_to_y_ratio = 1
    hom1 = Aperture(shape='r',ap_or_ob='a',Dx=om_clear_ap,Dy=om_clear_ap/aperture_x_
↪to_y_ratio)
    bl0.append( hom1, Use_PP(semi_analytical_treatment=0, zoom=zoom, sampling=zoom) )

    # Free space propagation from hom1 to hom2
    hom1_to_hom2_drift = Drift(src_to_hom2 - src_to_hom1); z0 = z0+(src_to_hom2 - src_
↪to_hom1)
    bl0.append( hom1_to_hom2_drift, Use_PP(semi_analytical_treatment=0))


    # Define HOM2.
    zoom = 1.0
    hom2 = Aperture('r','a', om_clear_ap, om_clear_ap/aperture_x_to_y_ratio)
    bl0.append( hom2,  Use_PP(semi_analytical_treatment=0, zoom=zoom, sampling=zoom/0.
↪75))

    #drift to CRL aperture
    hom2_to_crl_drift = Drift( src_to_crl - src_to_hom2 );z0 = z0+( src_to_crl - src_
↪to_hom2 )
    #bl0.append( hom2_to_crl_drift, Use_PP(semi_analytical_treatment=0))
    bl0.append( hom2_to_crl_drift, Use_PP(semi_analytical_treatment=1))


    # Define CRL
    crl_focussing_plane = 3 # Both horizontal and vertical.
    crl_delta = 4.7177e-06 # Refractive index decrement (n = 1- delta - i*beta)
    crl_attenuation_length  = 6.3e-3    # Attenuation length [m], Henke data.
    crl_shape = 1         # Parabolic lenses
    crl_aperture = 5.0e-3 # [m]
    crl_curvature_radius = 5.8e-3 # [m]
    crl_number_of_lenses = 19
    crl_wall_thickness = 8.0e-5 # Thickness
    crl_center_horizontal_coordinate = 0.0
    crl_center_vertical_coordinate = 0.0
    crl_initial_photon_energy = 8.48e3 # [eV] ### OK ???
    crl_final_photon_energy = 8.52e3 # [eV]   ### OK ???

    crl = CRL( _foc_plane=crl_focussing_plane,
               _delta=crl_delta,
               _atten_len=crl_attenuation_length,
               _shape=crl_shape,
               _apert_h=crl_aperture,
               _apert_v=crl_aperture,
               _r_min=crl_curvature_radius,
               _n=crl_number_of_lenses,
               _wall_thick=crl_wall_thickness,
               _xc=crl_center_horizontal_coordinate,
               _yc=crl_center_vertical_coordinate,
               _void_cen_rad=None,
               _e_start=crl_initial_photon_energy,
               _e_fin=crl_final_photon_energy,
```

(continues on next page)

(continued from previous page)

```
            )
    zoom=0.6

    bl0.append( crl, Use_PP(semi_analytical_treatment=1, zoom=zoom, sampling=zoom/0.
↪1) )


    crl_to_exp_drift = Drift( z ); z0 = z0+z
    bl0.append( crl_to_exp_drift, Use_PP(semi_analytical_treatment=1, zoom=1,␣
↪sampling=1))
#     bl0.append(Empty(),Use_PP(zoom=0.25, sampling=0.25))

    return bl0
```

```
Overwriting bl_S1_SPB_day1_simplified.py
```

initial Gaussian wavefront

With the calculated beam parameters the initial wavefront is build with 400x400 data points and at distance of the first flat offset mirror at 257.8 m. For further propagation the built wavefront should be stored.

After plotting the wavefront the FWHM could be printed out and compared with Gaussian beam divergence value.
Gaussian beam radius and size at distance z from the waist: $\omega(z) = \omega_0 * \sqrt{1 + \left(\frac{z}{z_R}\right)^2}$, where $\frac{1}{z_R} = \frac{\lambda}{\pi\omega_0^2}$

Expected FWHM at first screen or focusing mirror: $\theta_{FWHM} * z$

```
src_to_hom1 = 257.8 # Distance source to HOM 1 [m]

# Central photon energy.
ekev = 8.5 # Energy [keV]

# Pulse parameters.
qnC = 0.5                # e-bunch charge, [nC]
pulse_duration = 9.e-15 # [s] <-is not used really, only ~coh time pulse duration has␣
↪physical meaning
pulseEnergy = 1.5e-3     # total pulse energy, J
coh_time = 0.8e-15      # [s]<-should be SASE coherence time, then spectrum will be␣
↪the same as for SASE
                         # check coherence time for 8 keV 0.5 nC SASE1

# Angular distribution
theta_fwhm = calculate_theta_fwhm_cdr_s1(ekev,qnC) # From tutorial
#theta_fwhm = 2.124e-6 # Beam divergence         # From Patrick's raytrace.

# Gaussian beam parameters
wlambda = 12.4*1e-10/ekev # wavelength
w0 = wlambda/(np.pi*theta_fwhm) # beam waist;
zR = (np.pi*w0**2)/wlambda # Rayleigh range
fwhm_at_zR = theta_fwhm*zR # FWHM at Rayleigh range
sigmaAmp = w0/(2*np.sqrt(np.log(2))) # sigma of amplitude

print('expected FWHM at distance {:.1f} m is {:.2f} mm'.format(src_to_hom1,theta_
↪fwhm*src_to_hom1*1e3))
```

(continues on next page)

(continued from previous page)

```
# expected beam radius at M1 position to get the range of the wavefront
sig_num = 5.5
range_xy = w0 * np.sqrt(1+(src_to_hom1/zR)**2) *sig_num;#print('range_xy at HOM1: {:.
↪1f} mm'.format(range_xy*1e3))
fname = 'at_{:.0f}_m'.format(src_to_hom1)
```

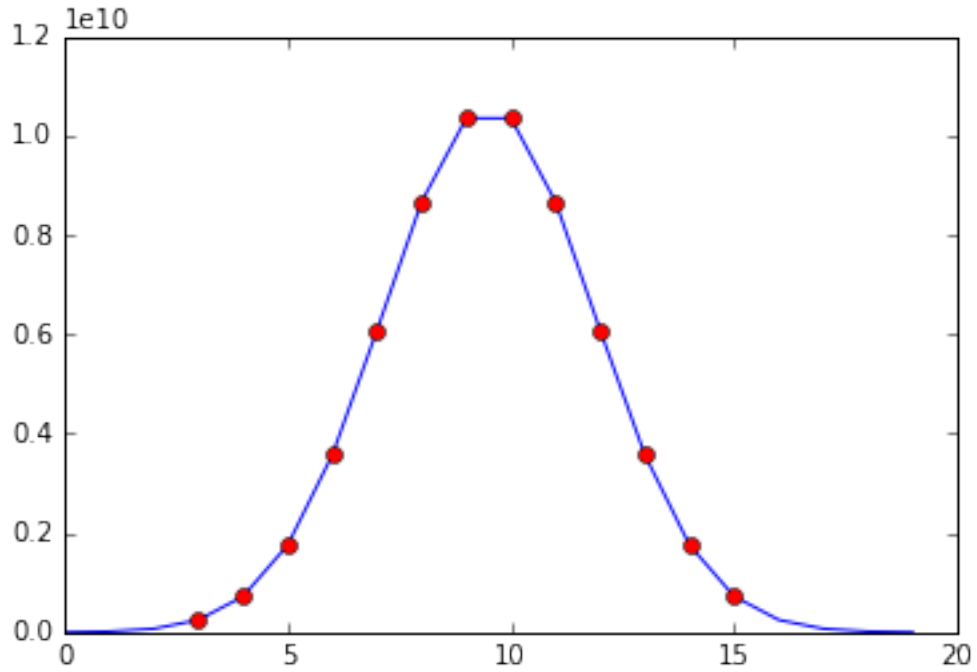
```
expected FWHM at distance 257.8 m is 0.53 mm
```

```
bSaved=False
num_points = 400 #number of points
dx = 10.e-6; range_xy = dx*(num_points-1);#print('range_xy :', range_xy)
nslices = 20;

srwl_wf = build_gauss_wavefront(num_points, num_points, nslices, ekev, -range_xy/2,␣
↪range_xy/2,
                                -range_xy/2, range_xy/2 ,coh_time/np.sqrt(2),
                                sigmaAmp, sigmaAmp, src_to_hom1,
                                pulseEn=pulseEnergy, pulseRange=8.)
wf = Wavefront(srwl_wf)
z0 = src_to_hom1
#defining name HDF5 file for storing wavefront
strOutInDataFolder = 'data_common'
#store wavefront to HDF5 file
if bSaved:
    wf.store_hdf5(fname+'.h5'); print('saving WF to %s' %fname+'.h5')

xx=calculate_fwhm(wf);
print('FWHM at distance {:.1f} m: {:.2f} x {:.2f} mm2'.format(z0,xx[u'fwhm_x']*1e3,
↪xx[u'fwhm_y']*1e3));
```

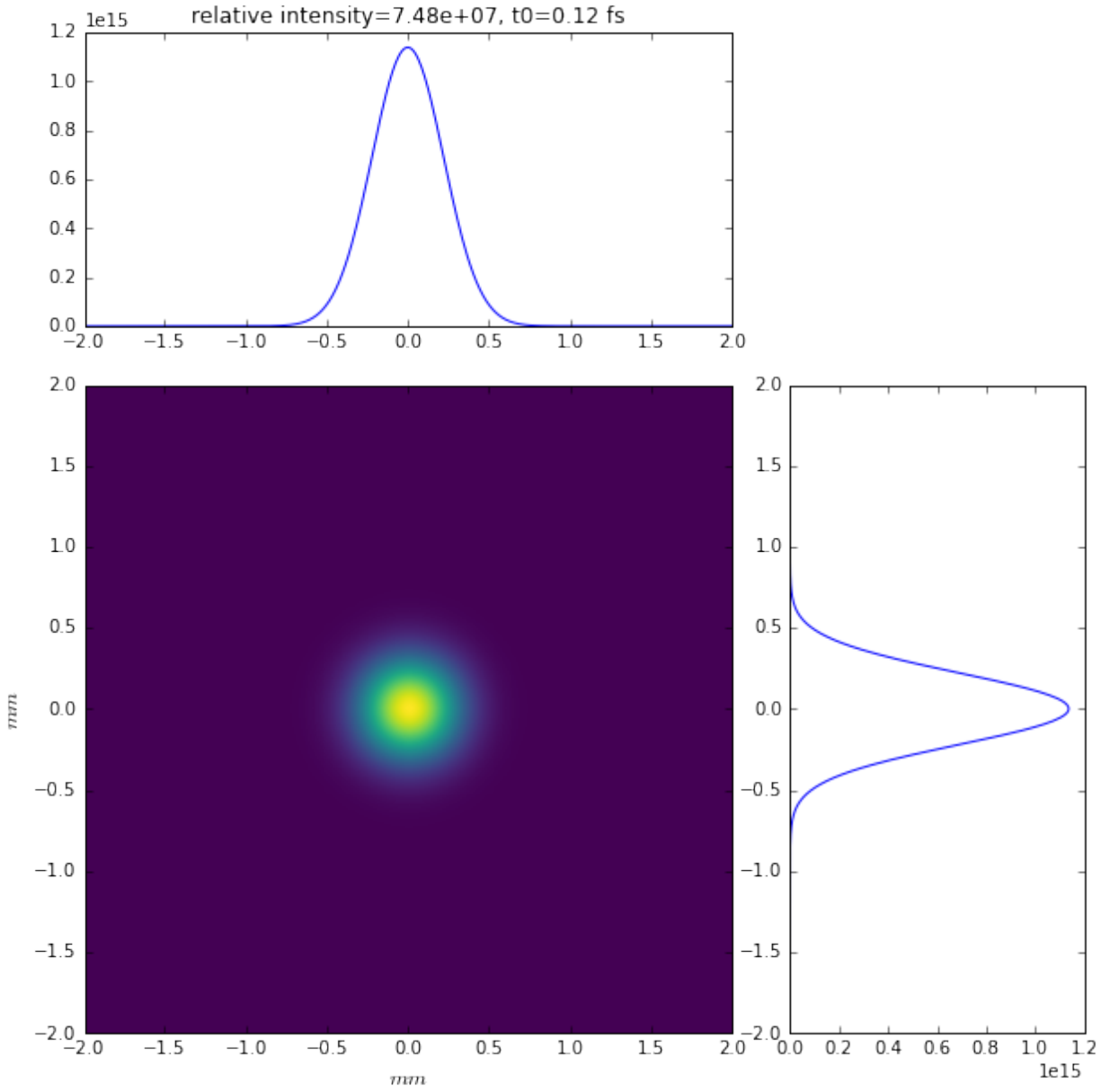
```
FWHM at distance 257.8 m: 0.52 x 0.52 mm2
```

```
#input gaussian beam
print( 'dy {:.1f} um'.format((wf.params.Mesh.yMax-wf.params.Mesh.yMin)*1e6/(wf.params.
↪Mesh.ny-1.)))
print( 'dx {:.1f} um'.format((wf.params.Mesh.xMax-wf.params.Mesh.xMin)*1e6/(wf.params.
↪Mesh.nx-1.)))
plot_t_wf(wf)
look_at_q_space(wf)
```

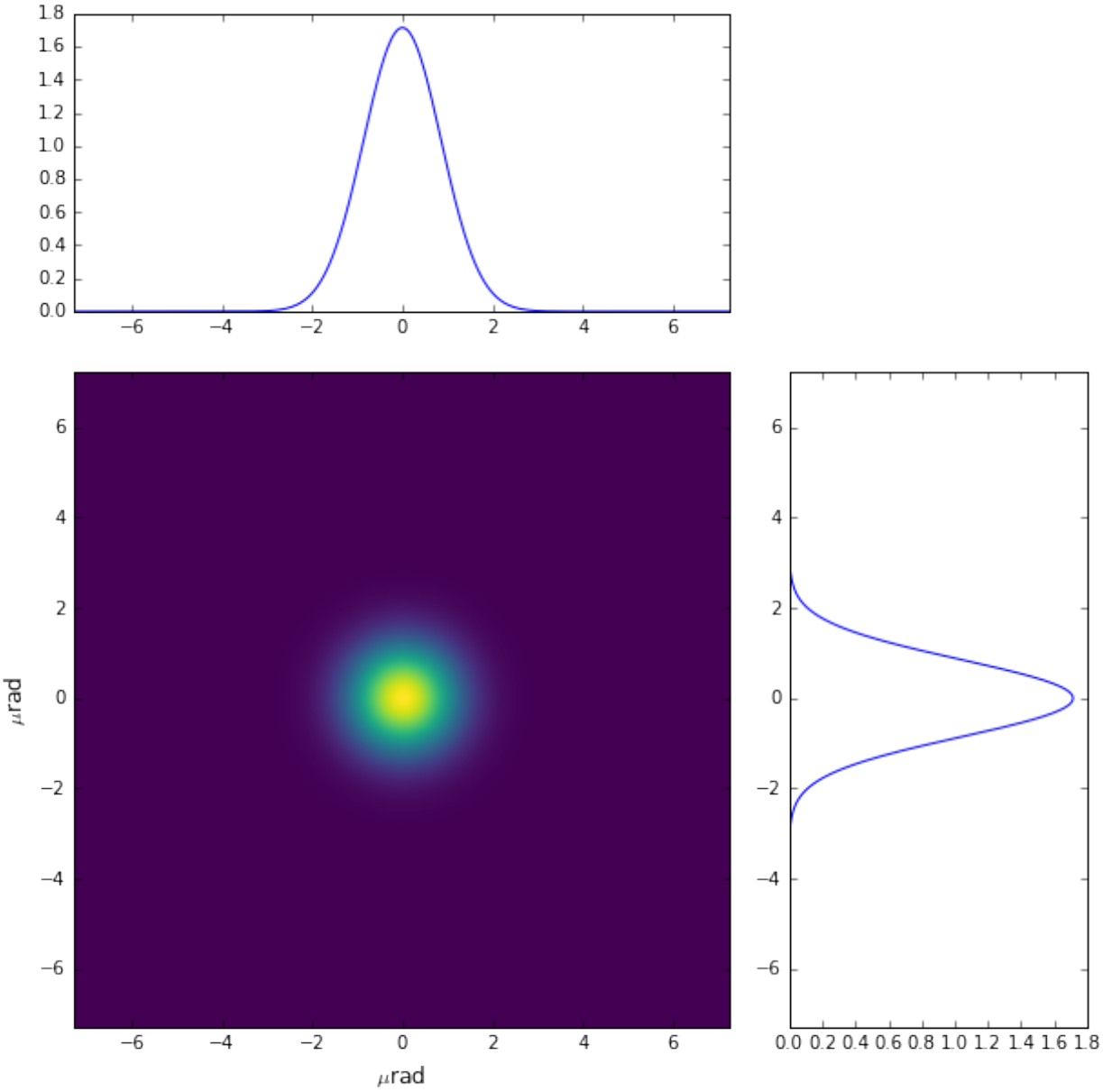
```
dy 10.0 um
dx 10.0 um
```




```
number of meaningful slices: 13
R-space
(400,) (400,)
```




```
Q-space
{'fwhm_x': 1.999254044117647e-06, 'fwhm_y': 1.999254044117647e-06}
Q-space
(400,) (400,)
```




```
#loading beamline from file
import imp
custom_beamline = imp.load_source('custom_beamline', 'bl_S1_SPB_day1_simplified.py')
get_beamline = custom_beamline.get_beamline
bl = get_beamline()
print(bl)
```

```
Optical Element Setup: CRL Focal Length: 32.35296414510639 m
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.00288
    Dy = 0.00288
    ap_or_ob = a
    shape = r
```

(continues on next page)

(continued from previous page)

```
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 10.0
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.3333333333333333, 1.0, 1.3333333333333333,
↪ 0, 0, 0]
    Dx = 0.00288
    Dy = 0.00288
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 620.0
    treat = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 1, 0, 0.6, 5.999999999999999, 0.6, 5.999999999999999,␣
↪0, 0, 0]
    Fx = 32.35296414510639
    Fy = 32.35296414510639
    arTr = array of size 2004002
    extTr = 1
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 8520.0
            eStart = 8480.0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1001
            ny = 1001
            xFin = 0.0027500000000000003
            xStart = -0.0027500000000000003
            yFin = 0.0027500000000000003
            yStart = -0.0027500000000000003
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 34.0
    treat = 0
```

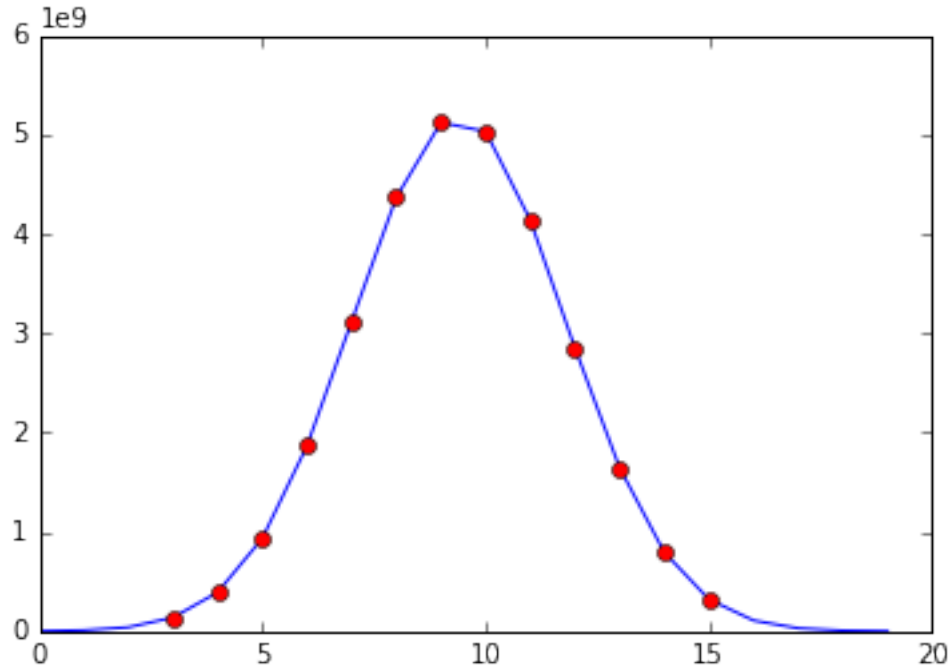
```
#propagated gaussian beam
```

(continues on next page)

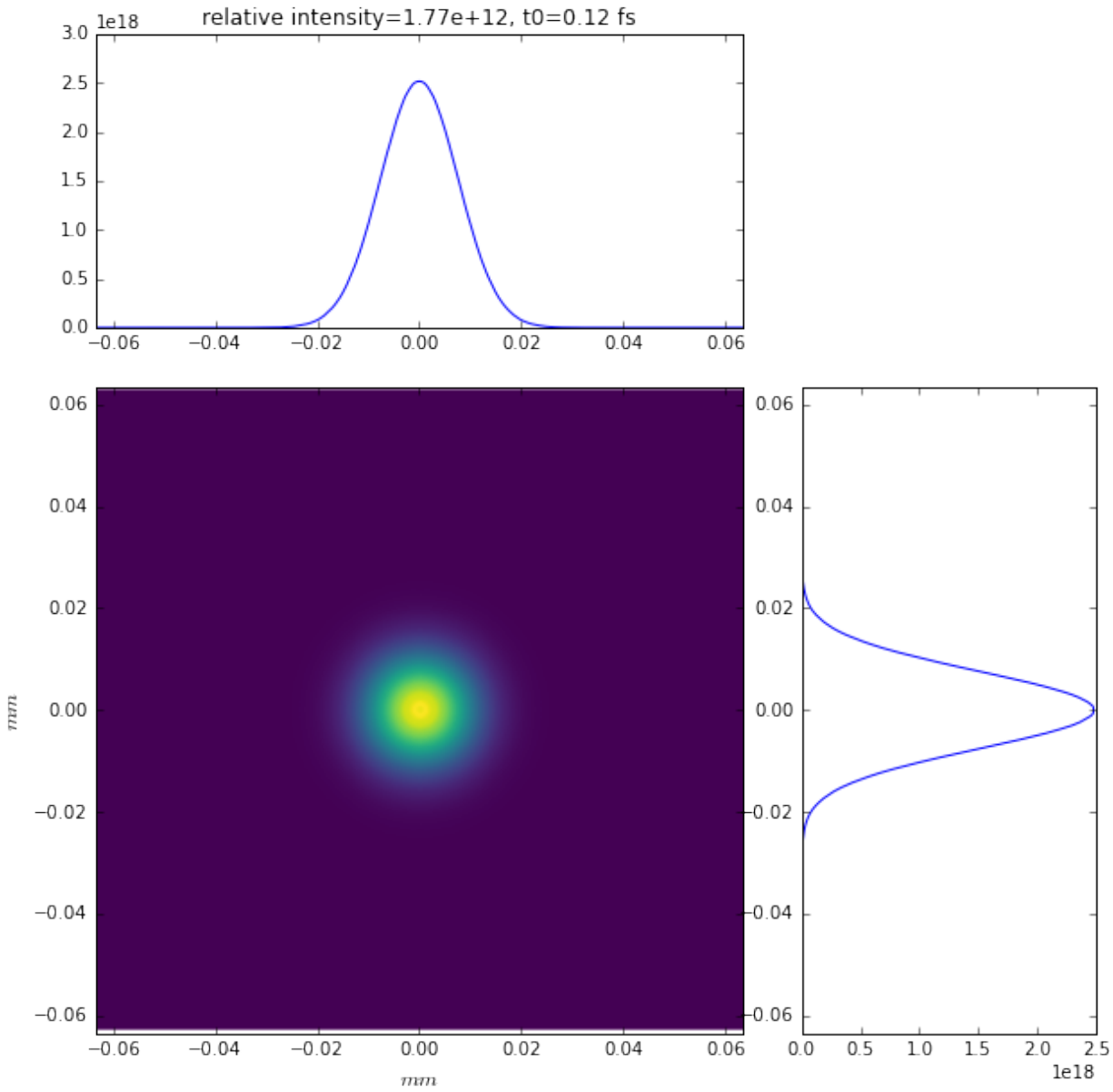
(continued from previous page)

```
srwl.SetRepresElecField(wf._srwl_wf, 'f') # <---- switch to frequency domain
bl.propagate(wf)
srwl.SetRepresElecField(wf._srwl_wf, 't')
print('FWHM after CRLs:');print(calculate_fwhm(wf))
print('FWHM at distance {:.1f} m:'.format(wf.params.Mesh.zCoord));print(calculate_
↪fwhm(wf))
plot_t_wf(wf)
look_at_q_space(wf)
```

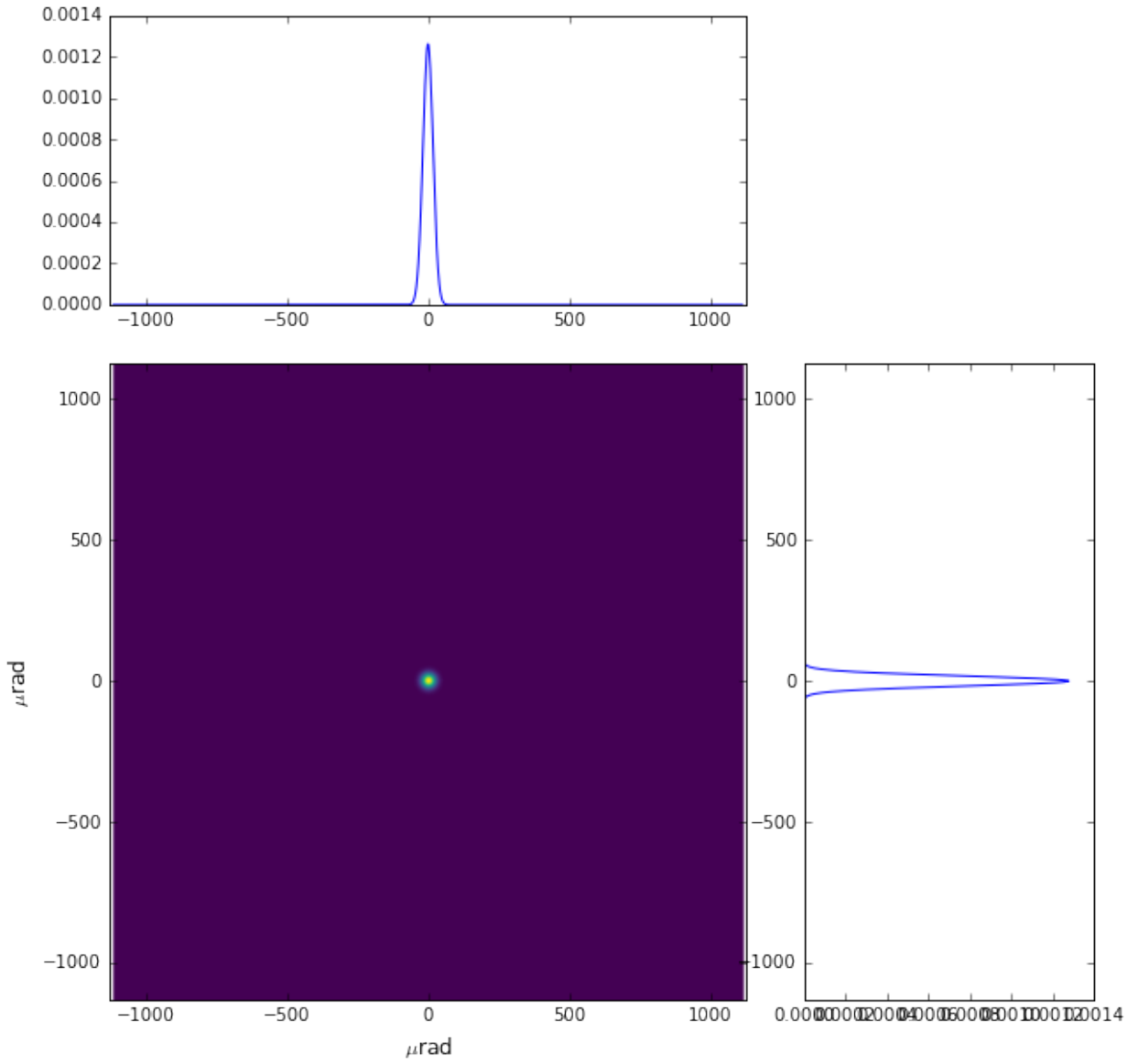
```
FWHM after CRLs:
{'fwhm_x': 1.7897682395761173e-05, 'fwhm_y': 1.779350912766013e-05}
FWHM at distance 921.8 m:
{'fwhm_x': 1.7897682395761173e-05, 'fwhm_y': 1.779350912766013e-05}
```




```
number of meaningful slices: 13
R-space
(1944,) (1944,)
```




```
Q-space
{'fwhm_x': 4.242918502417042e-05, 'fwhm_y': 4.298910472684863e-05}
Q-space
(1944,) (1944,)
```



1.11.2 S1 SPB Day1 simplified beamline with imperfect offset mirrors

```
%matplotlib inline
```

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals
```

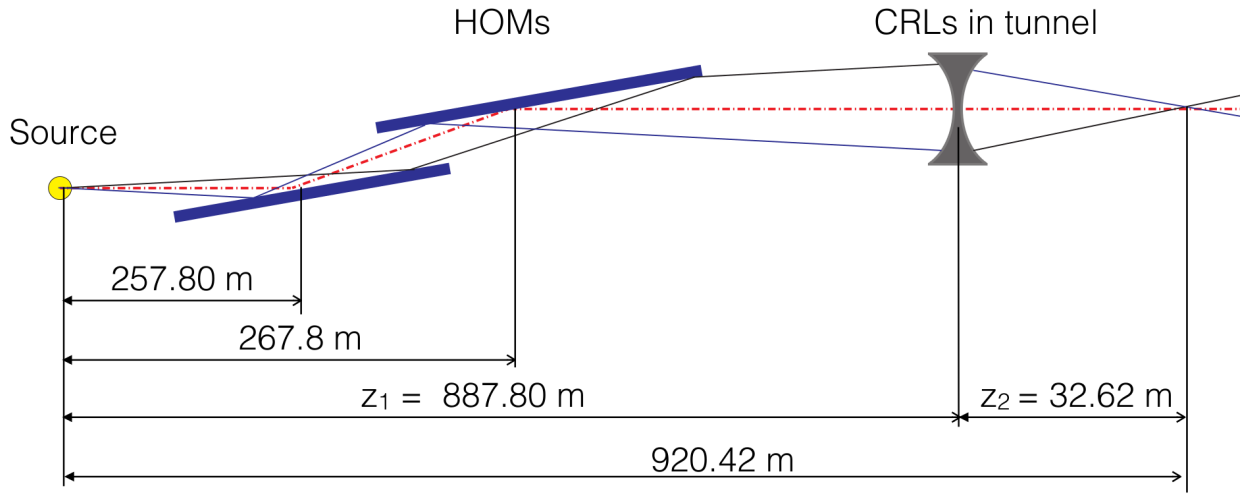
```
import os
import sys


# wpg_path = '/afs/desy.de/group/exfel/software/wpg/latest/' # DESY installation
```

(continues on next page)

(continued from previous page)

```
wpg_path = os.path.join('..','..','..')
sys.path.insert(0, wpg_path)

import numpy as np
import pylab as plt


from wpg import Wavefront, Beamline
from wpg.optical_elements import Aperture, Drift, CRL, Empty, Use_PP
from wpg.generators import build_gauss_wavefront

from wpg.srwlib import srwl

from wpg.wpg_uti_exfl import calculate_theta_fwhm_cdr_s1
from wpg.wpg_uti_wf import calculate_fwhm, averaged_intensity, look_at_q_space, plot_
↪t_wf
from wpg.wpg_uti_oe import show_transmission
```

```
from IPython.display import Image
Image(filename='day1_1.png')
```




```
%%file bl_S1_SPB_day1_mirrors.py

def get_beamline():
    import os
    import wpg
    from wpg import Beamline
    from wpg.optical_elements import Aperture, Drift, CRL, Empty, Use_PP, WF_dist,␣
↪calculateOPD

    wpg_path = os.path.abspath(os.path.dirname(wpg.__file__))

    # S1 beamline layout
    # Geometry ###
    src_to_hom1 = 257.8  # Distance source to HOM 1 [m]
    src_to_hom2 = 267.8  # Distance source to HOM 2 [m]
    src_to_crl = 887.8  # Distance source to CRL [m]
    #     src_to_exp = 920.42 # Distance source to experiment [m]
```

(continues on next page)

(continued from previous page)

```
    # Drift to focus aperture
    # crl_to_exp_drift = Drift( src_to_exp - src_to_crl )
    z = 34.0
    # define distances, angles, etc
    # ...
    # Incidence angle at HOM

      # should be checked for other beams !!!

    theta_om = 3.6e-3  # [rad]

    om_mirror_length = 0.8  # [m]
    om_clear_ap = om_mirror_length * theta_om

    # define the beamline:
    bl0 = Beamline()
    zoom = 1

    # Define HOM1.
    aperture_x_to_y_ratio = 1
    hom1 = Aperture(
        shape='r', ap_or_ob='a', Dx=om_clear_ap, Dy=om_clear_ap / aperture_x_to_y_
↪ratio)
    bl0.append(
        hom1, Use_PP(semi_analytical_treatment=0, zoom=zoom, sampling=zoom))

    # Define mirror profile
    hom1_wavefront_distortion = WF_dist(nx=1500, ny=100,
                                        Dx=om_clear_ap, Dy=om_clear_ap / aperture_x_
↪to_y_ratio)
    # Apply distortion.
    mirrors_path = os.path.join(wpg_path, '..', 'samples', 'data_common')
    hom1_wavefront_distortion = calculateOPD(wf_dist=hom1_wavefront_distortion,
                                             mdatafile=os.path.join(
                                                 mirrors_path, 'mirror1.dat'),
                                             ncol=2,
                                             delim=' ',
                                             Orient='x',
                                             theta=theta_om,
                                             scale=1.,
                                             stretching=1.)
    bl0.append(hom1_wavefront_distortion,
               Use_PP(semi_analytical_treatment=0, zoom=zoom, sampling=zoom))

    # Free space propagation from hom1 to hom2
    hom1_to_hom2_drift = Drift(src_to_hom2 - src_to_hom1)
    bl0.append(hom1_to_hom2_drift, Use_PP(semi_analytical_treatment=0))

    # Define HOM2.
    zoom = 1.0
    hom2 = Aperture('r', 'a', om_clear_ap, om_clear_ap / aperture_x_to_y_ratio)
    bl0.append(hom2, Use_PP(semi_analytical_treatment=0,
                            zoom=zoom, sampling=zoom / 0.75))

    # define mirror 2
    # nx, ny from tutorial #3 (new).
```

(continues on next page)

(continued from previous page)

```
    hom2_wavefront_distortion = WF_dist(nx=1500, ny=100,
                                        Dx=om_clear_ap, Dy=om_clear_ap / aperture_x_
↪to_y_ratio)
    # Apply distortion.
    hom2_wavefront_distortion = calculateOPD(wf_dist=hom2_wavefront_distortion,
                                             mdatafile=os.path.join(
                                                 mirrors_path, 'mirror2.dat'),
                                             ncol=2,
                                             delim=' ',
                                             Orient='x',
                                             theta=theta_om,
                                             scale=1.,
                                             stretching=1.)

    bl0.append(hom2_wavefront_distortion, Use_PP(
        semi_analytical_treatment=0, zoom=zoom, sampling=zoom))

    # drift to CRL aperture
    hom2_to_crl_drift = Drift(src_to_crl - src_to_hom2)

    bl0.append(hom2_to_crl_drift, Use_PP(semi_analytical_treatment=1))

    # Define CRL
    crl_focussing_plane = 3  # Both horizontal and vertical.
    # Refractive index decrement (n = 1- delta - i*beta)
    crl_delta = 4.7177e-06
    crl_attenuation_length = 6.3e-3    # Attenuation length [m], Henke data.
    crl_shape = 1          # Parabolic lenses
    crl_aperture = 5.0e-3  # [m]
    crl_curvature_radius = 5.8e-3  # [m]
    crl_number_of_lenses = 19
    crl_wall_thickness = 8.0e-5  # Thickness
    crl_center_horizontal_coordinate = 0.0
    crl_center_vertical_coordinate = 0.0
    crl_initial_photon_energy = 8.48e3  # [eV] ### OK ???
    crl_final_photon_energy = 8.52e3  # [eV]   ### OK ???

    crl = CRL(_foc_plane=crl_focussing_plane,
              _delta=crl_delta,
              _atten_len=crl_attenuation_length,
              _shape=crl_shape,
              _apert_h=crl_aperture,
              _apert_v=crl_aperture,
              _r_min=crl_curvature_radius,
              _n=crl_number_of_lenses,
              _wall_thick=crl_wall_thickness,
              _xc=crl_center_horizontal_coordinate,
              _yc=crl_center_vertical_coordinate,
              _void_cen_rad=None,
              _e_start=crl_initial_photon_energy,
              _e_fin=crl_final_photon_energy,
              )
    zoom = 0.6

    bl0.append(
        crl, Use_PP(semi_analytical_treatment=1, zoom=zoom, sampling=zoom/0.1))
```

(continues on next page)

(continued from previous page)

```
    crl_to_exp_drift = Drift(z)
    bl0.append(crl_to_exp_drift, Use_PP(
        semi_analytical_treatment=1, zoom=1, sampling=1))
    #     bl0.append(Empty(),Use_PP(zoom=0.25, sampling=0.25))

    return bl0
```

```
Overwriting bl_S1_SPB_day1_mirrors.py
```

initial Gaussian wavefront

With the calculated beam parameters the initial wavefront is build with 400x400 data points and at distance of the first flat offset mirror at 257.8 m. For further propagation the built wavefront should be stored.

After plotting the wavefront the FWHM could be printed out and compared with Gaussian beam divergence value. #### Gaussian beam radius and size at distance z from the waist: $\omega(z) = \omega_0 * \sqrt{1 + \left(\frac{z}{z_R}\right)^2}$, where $\frac{1}{z_R} = \frac{\lambda}{\pi\omega_0^2}$

Expected FWHM at first screen or focusing mirror: $\theta_{FWHM} * z$

```
src_to_hom1 = 257.8 # Distance source to HOM 1 [m]

# Central photon energy.
ekev = 8.5 # Energy [keV]

# Pulse parameters.
qnC = 0.5                # e-bunch charge, [nC]
pulse_duration = 9.e-15 # [s] <-is not used really, only ~coh time pulse duration has␣
↪physical meaning
pulseEnergy = 1.5e-3     # total pulse energy, J
coh_time = 0.8e-15      # [s]<-should be SASE coherence time, then spectrum will be␣
↪the same as for SASE
                        # check coherence time for 8 keV 0.5 nC SASE1

# Angular distribution
theta_fwhm = calculate_theta_fwhm_cdr_s1(ekev,qnC) # From tutorial
#theta_fwhm = 2.124e-6 # Beam divergence         # From Patrick's raytrace.

# Gaussian beam parameters
wlambda = 12.4*1e-10/ekev # wavelength
w0 = wlambda/(np.pi*theta_fwhm) # beam waist;
zR = (np.pi*w0**2)/wlambda # Rayleigh range
fwhm_at_zR = theta_fwhm*zR # FWHM at Rayleigh range
sigmaAmp = w0/(2*np.sqrt(np.log(2))) # sigma of amplitude

print('expected FWHM at distance {:.1f} m is {:.2f} mm'.format(src_to_hom1,theta_
↪fwhm*src_to_hom1*1e3))

# expected beam radius at M1 position to get the range of the wavefront
sig_num = 5.5
range_xy = w0 * np.sqrt(1+(src_to_hom1/zR)**2) *sig_num;#print('range_xy at HOM1: {:.
↪1f} mm'.format(range_xy*1e3))
fname = 'at_{:.0f}_m'.format(src_to_hom1)
```

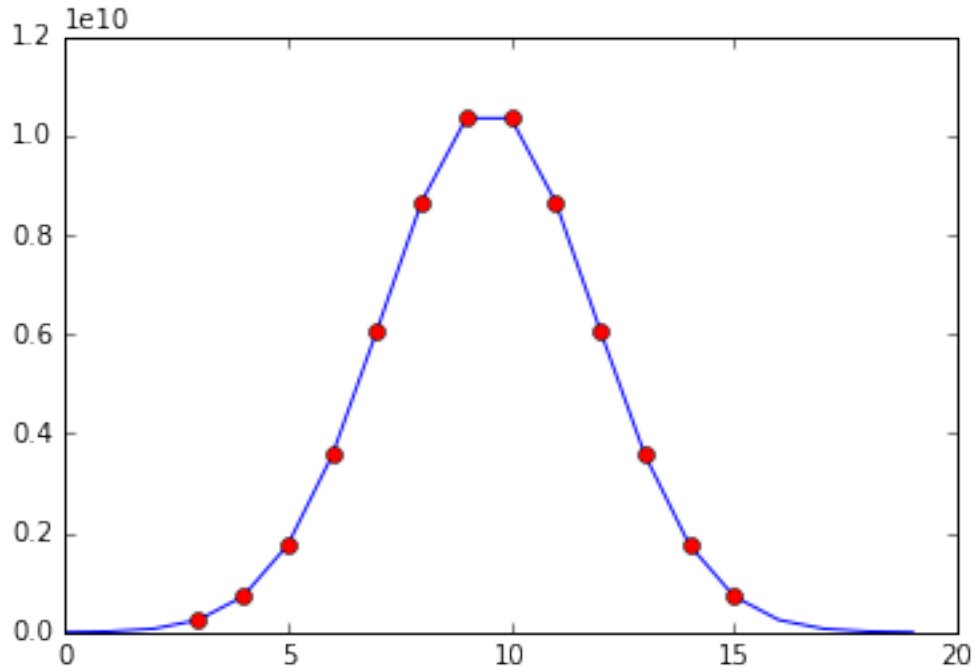
```
expected FWHM at distance 257.8 m is 0.53 mm
```

```
bSaved=False
num_points = 400 #number of points
dx = 10.e-6; range_xy = dx*(num_points-1);#print('range_xy :', range_xy)
nslices = 20;

srwl_wf = build_gauss_wavefront(num_points, num_points, nslices, ekev, -range_xy/2,␣
↪range_xy/2,
                                -range_xy/2, range_xy/2 ,coh_time/np.sqrt(2),
                                sigmaAmp, sigmaAmp, src_to_hom1,
                                pulseEn=pulseEnergy, pulseRange=8.)
wf = Wavefront(srwl_wf)
z0 = src_to_hom1
#defining name HDF5 file for storing wavefront
strOutInDataFolder = 'data_common'
#store wavefront to HDF5 file
if bSaved:
    wf.store_hdf5(fname+'.h5'); print('saving WF to %s' %fname+'.h5')

xx=calculate_fwhm(wf);
print('FWHM at distance {:.1f} m: {:.2f} x {:.2f} mm2'.format(z0,xx[u'fwhm_x']*1e3,
↪xx[u'fwhm_y']*1e3));
```

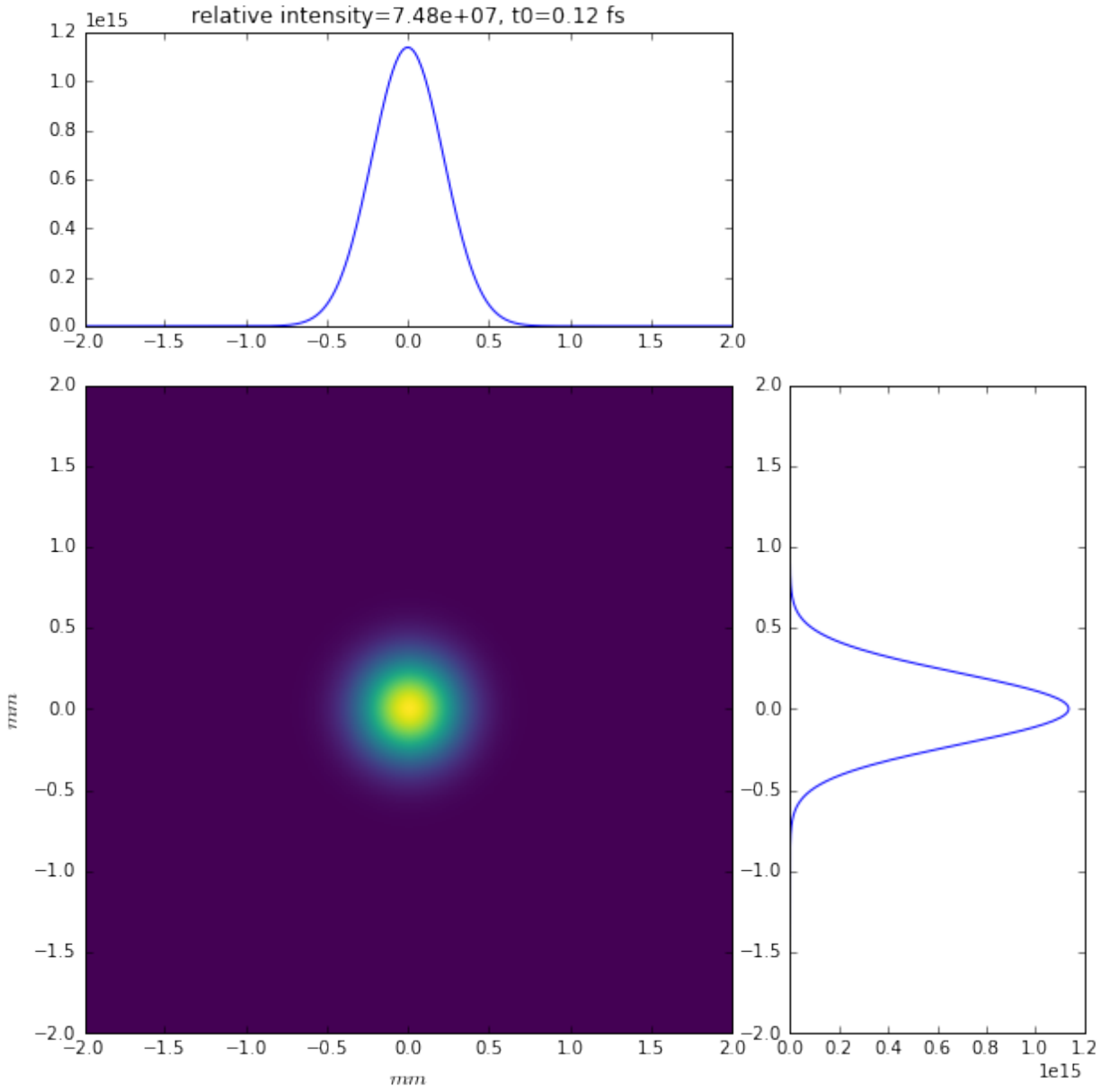
```
FWHM at distance 257.8 m: 0.52 x 0.52 mm2
```

```
#input gaussian beam
print( 'dy {:.1f} um'.format((wf.params.Mesh.yMax-wf.params.Mesh.yMin)*1e6/(wf.params.
↪Mesh.ny-1.)))
print( 'dx {:.1f} um'.format((wf.params.Mesh.xMax-wf.params.Mesh.xMin)*1e6/(wf.params.
↪Mesh.nx-1.)))
plot_t_wf(wf)
look_at_q_space(wf)
```

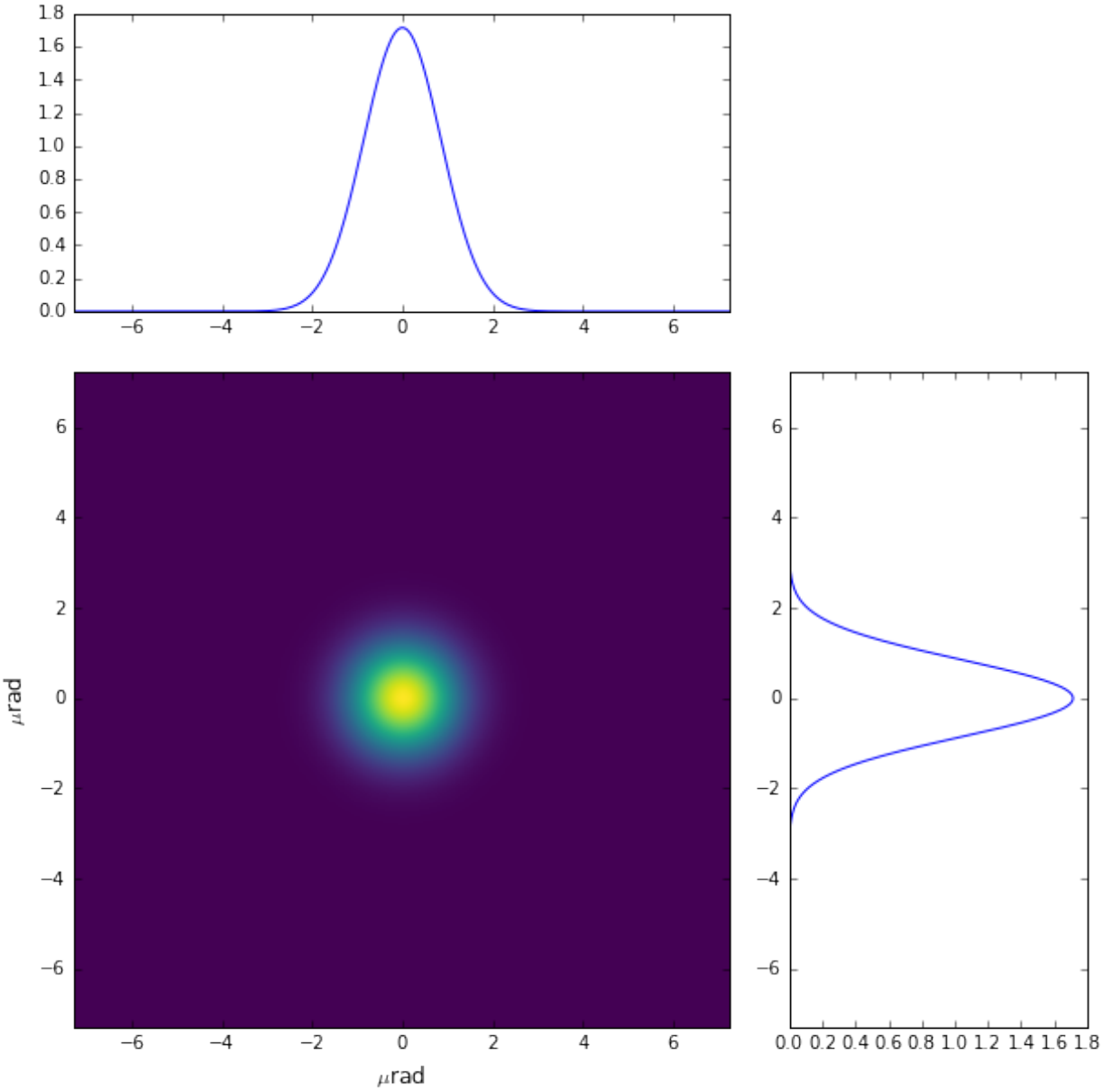
```
dy 10.0 um
dx 10.0 um
```




```
number of meaningful slices: 13
R-space
(400,) (400,)
```




```
Q-space
{'fwhm_x': 1.999254044117647e-06, 'fwhm_y': 1.999254044117647e-06}
Q-space
(400,) (400,)
```




```
#loading beamline from file
import imp
custom_beamline = imp.load_source('custom_beamline', 'bl_S1_SPB_day1_mirrors.py')
get_beamline = custom_beamline.get_beamline
bl = get_beamline()
print(bl)
```

```
Optical Element Setup: CRL Focal Length: 32.35296414510639 m
Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Dx = 0.00288
    Dy = 0.00288
    ap_or_ob = a
    shape = r
```

(continues on next page)

(continued from previous page)

```
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.00144
            xStart = -0.00144
            yFin = 0.00144
            yStart = -0.00144
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 10.0
    treat = 0

Optical Element: Aperture / Obstacle
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.3333333333333333, 1.0, 1.3333333333333333,
↪ 0, 0, 0]
    Dx = 0.00288
    Dy = 0.00288
    ap_or_ob = a
    shape = r
    x = 0
    y = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 0, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    Fx = 1e+23
    Fy = 1e+23
    arTr = array of size 300000
    extTr = 0
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 0
            eStart = 0
            hvx = 1
            hvy = 0
            hvz = 0
```

(continues on next page)

(continued from previous page)

```
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1500
            ny = 100
            xFin = 0.00144
            xStart = -0.00144
            yFin = 0.00144
            yStart = -0.00144
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 620.0
    treat = 0

Optical Element: Transmission (generic)
Prop. parameters = [0, 0, 1.0, 1, 0, 0.6, 5.999999999999999, 0.6, 5.999999999999999,␣
↪0, 0, 0]
    Fx = 32.35296414510639
    Fy = 32.35296414510639
    arTr = array of size 2004002
    extTr = 1
    mesh = Radiation Mesh (Sampling)
            arSurf = None
            eFin = 8520.0
            eStart = 8480.0
            hvx = 1
            hvy = 0
            hvz = 0
            ne = 1
            nvx = 0
            nvy = 0
            nvz = 1
            nx = 1001
            ny = 1001
            xFin = 0.0027500000000000003
            xStart = -0.0027500000000000003
            yFin = 0.0027500000000000003
            yStart = -0.0027500000000000003
            zStart = 0


Optical Element: Drift Space
Prop. parameters = [0, 0, 1.0, 1, 0, 1.0, 1.0, 1.0, 1.0, 0, 0, 0]
    L = 34.0
    treat = 0
```

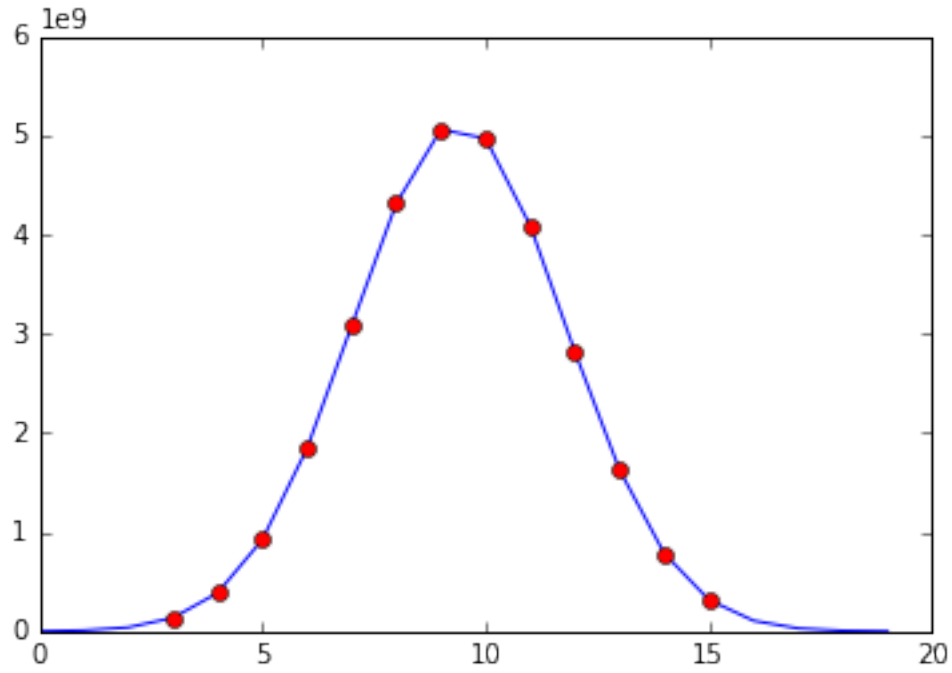
```
#propagated gaussian beam
srwl.SetRepresElecField(wf._srwl_wf, 'f') # <---- switch to frequency domain
bl.propagate(wf)
srwl.SetRepresElecField(wf._srwl_wf, 't')
print('FWHM after CRLs:');print(calculate_fwhm(wf))
print('FWHM at distance {:.1f} m:'.format(wf.params.Mesh.zCoord));print(calculate_
↪fwhm(wf))
```

(continues on next page)

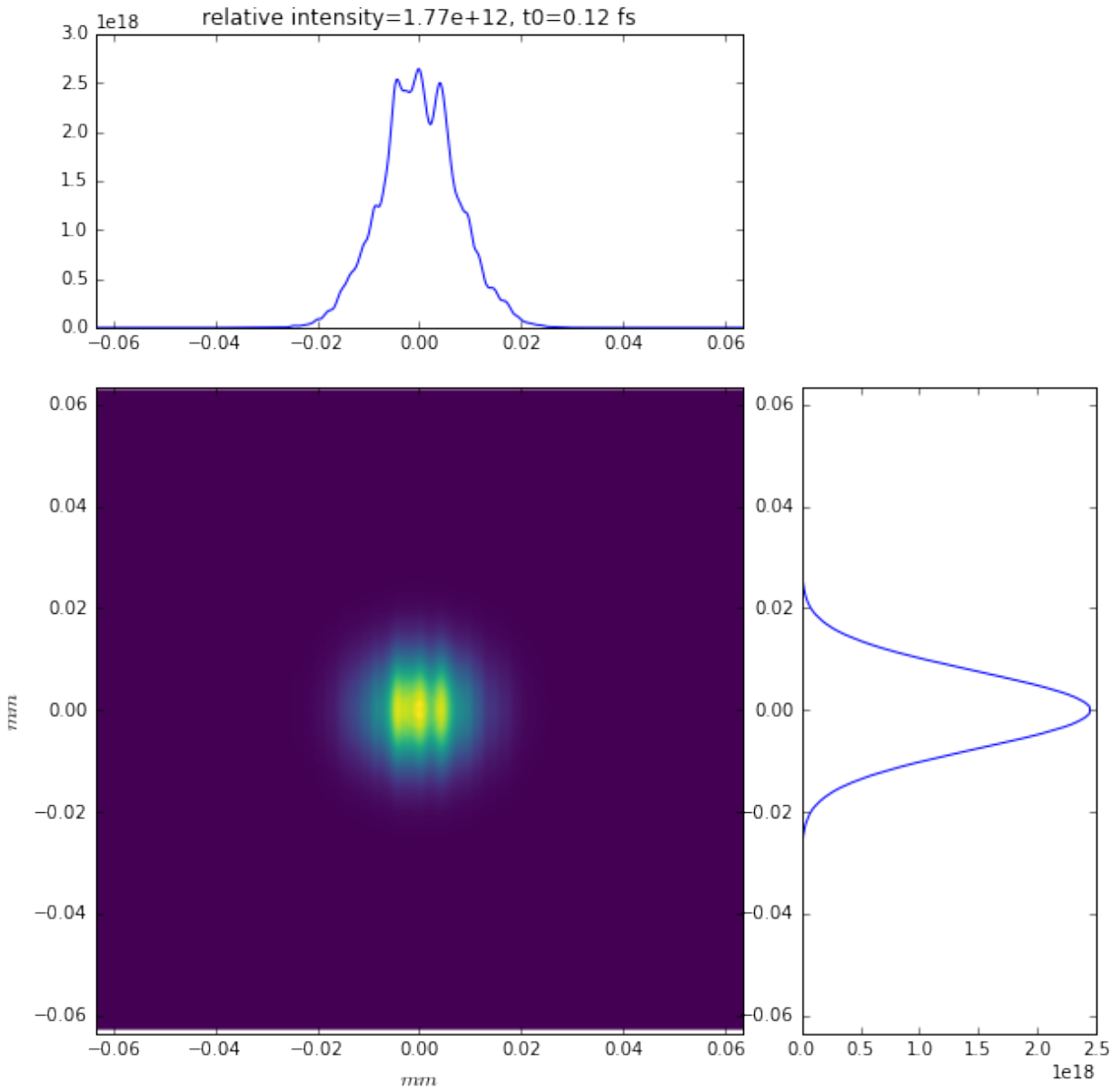
(continued from previous page)

```
plot_t_wf(wf)
look_at_q_space(wf)
```

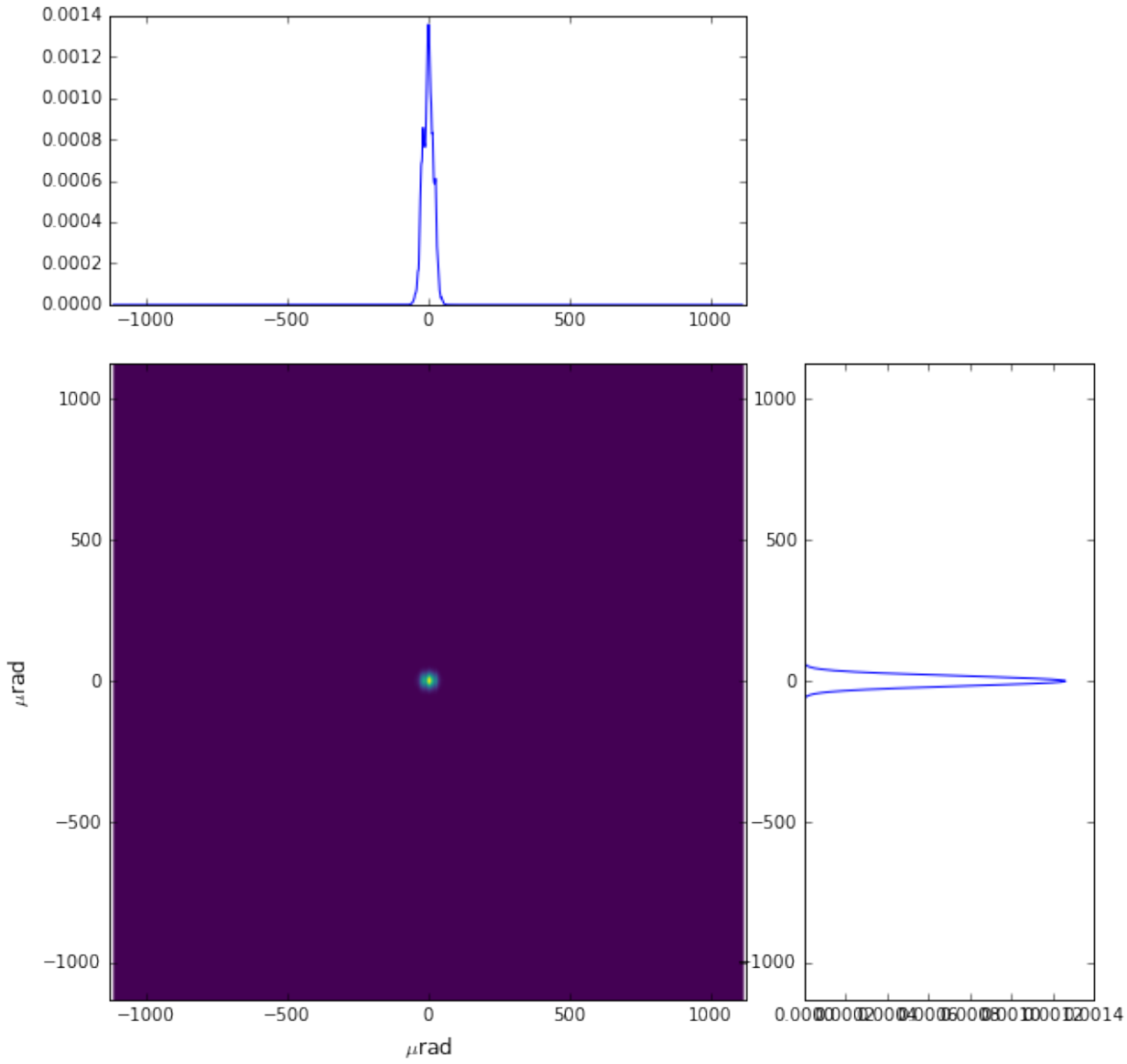
```
FWHM after CRLs:
{'fwhm_x': 1.5219562037271364e-05, 'fwhm_y': 1.779350912766013e-05}
FWHM at distance 921.8 m:
{'fwhm_x': 1.5219562037271364e-05, 'fwhm_y': 1.779350912766013e-05}
```




```
number of meaningful slices: 13
R-space
(1944,) (1944,)
```




```
Q-space
{'fwhm_x': 4.242918502417042e-05, 'fwhm_y': 4.298910472684863e-05}
Q-space
(1944,) (1944,)
```


0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
−1000
−500
0
500
1000
1000
500
0
−500
−1000
μrad
μrad

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

W

Index

K

L

M

P

R

S

T

U

V

W

X

Z