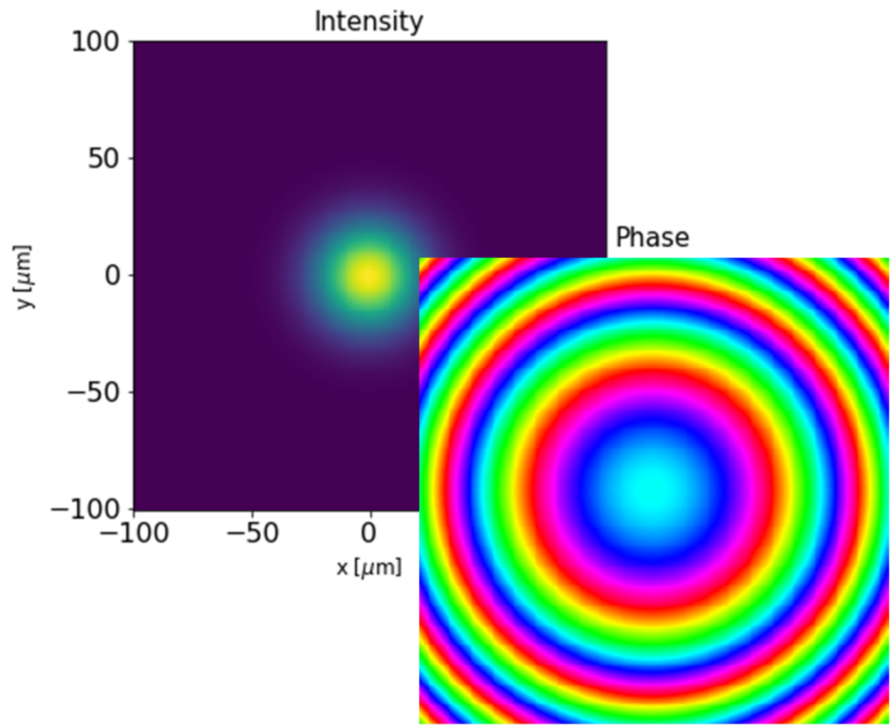
Wavefront propagation in OCELOT to simulate FEL source and image

Svitozar Serkez, FPH, European XFEL

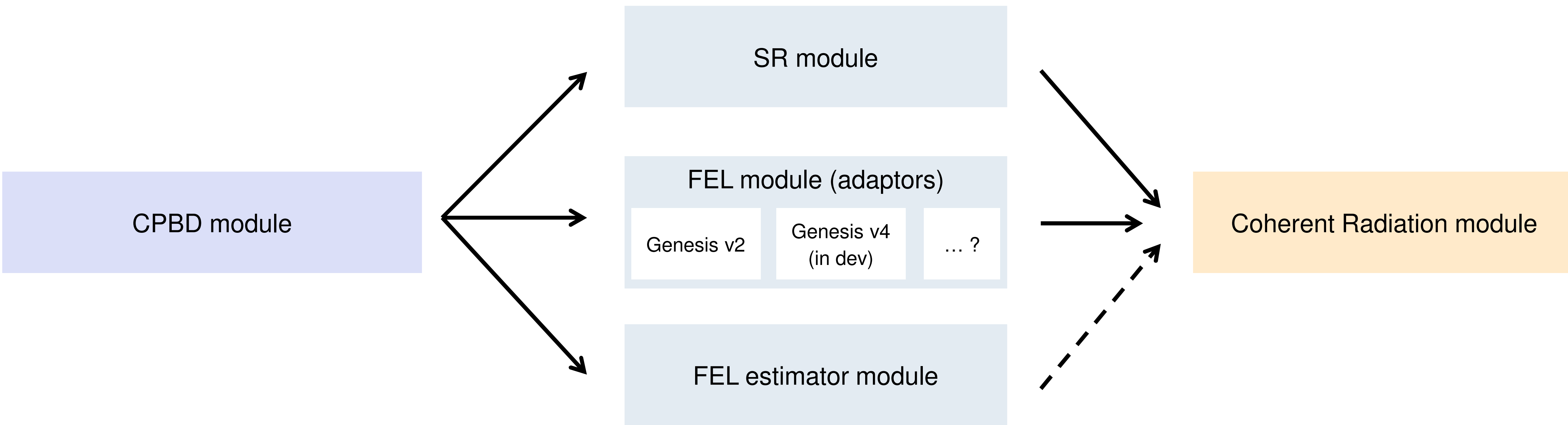


The problem

Let's say we have already simulated radiation distribution of (SASE) FEL radiation, and

- We want to characterize this radiation
- We want to know radiation properties
  - at the source
  - at the sample

(some of) Ocelot architecture



(some of) Ocelot architecture

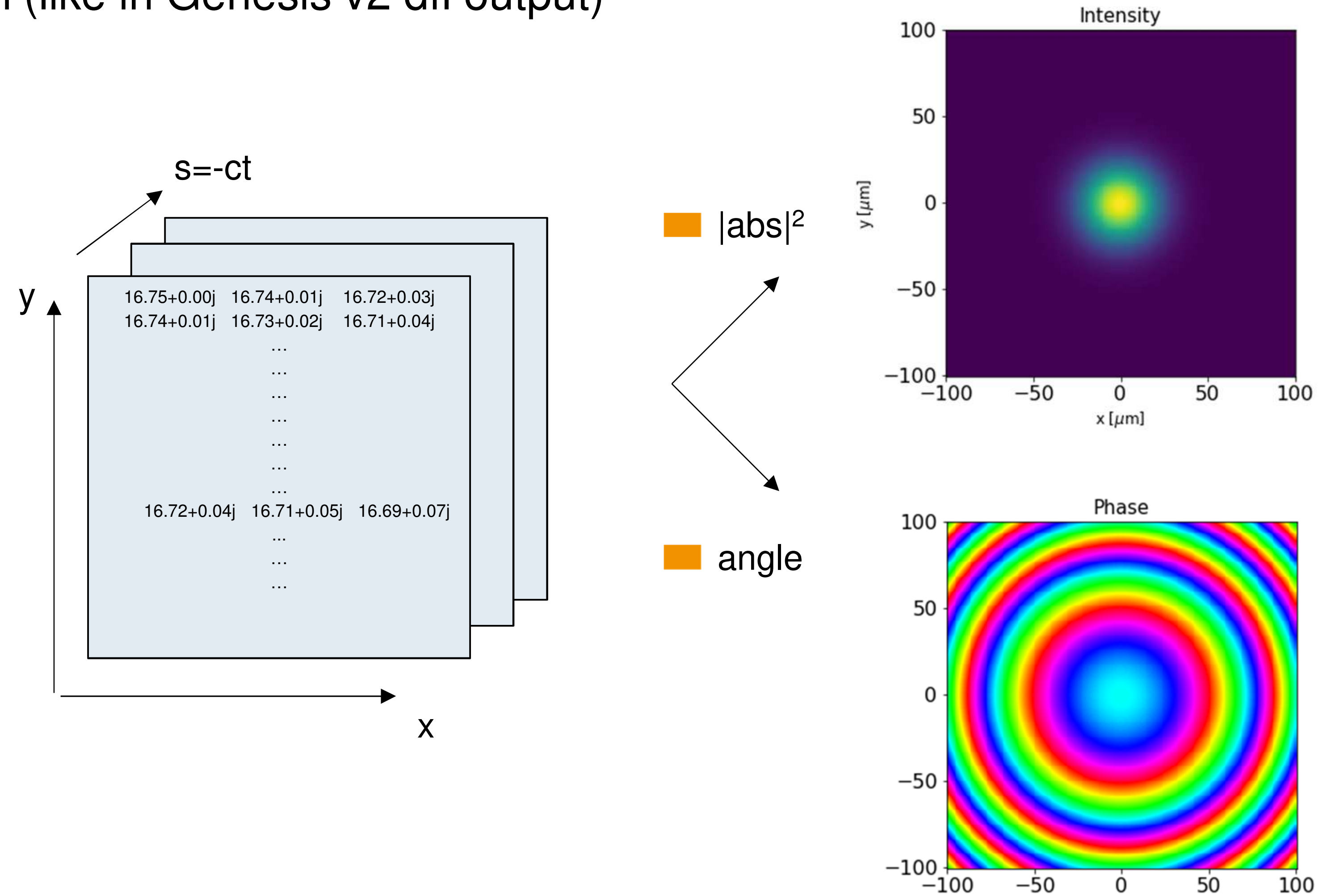
Coherent radiation field class

dfl = RadiationField()

The RadiationField object has a 3D data cube in dfl.fld attribute (numpy array) of complex numbers in SVEA approximation (like in Genesis v2 dfl output)

Methods
dfl.to_domain('s')
dfl.curve_wavefront(self, r=inf, plane='xy', . . .)
dfl.prop(self, z, fine=1, return_result=0, . . .)
dfl.prop_m(self, z, m=1, fine=1, return_result=0, . . .)
………..

Functions
plot_dfl(dfl, domains='tk' , phase=True, . . .)
dfl_waistscan(dfl, z_pos, projection=0, …)
dfl_disperse(dfl, coeff, …)
dfl_interp(dfl, interpL, inpterpN,)
dfl_trf(dfl, trf, …)
dfl_ap_rect(dfl, ap_x=np.inf, ap_y=np.inf)



See Tutorial https://nbviewer.jupyter.org/github/ocelot-collab/ocelot/blob/master/demos/ipython_tutorials/pfs_2_radiation_field.ipynb

Coherent radiation field class

dfl = RadiationField() can be imported or generated

Importing from Genesis v2

```
out = read_out_file("filepath.out")
dfl = read_dfl_file_out(out, "filepath.dfl")
```

Importing from Genesis v4

```
dfl = read_dfl4("filepath.fld.h5")
```

Modelling with Gaussian beam

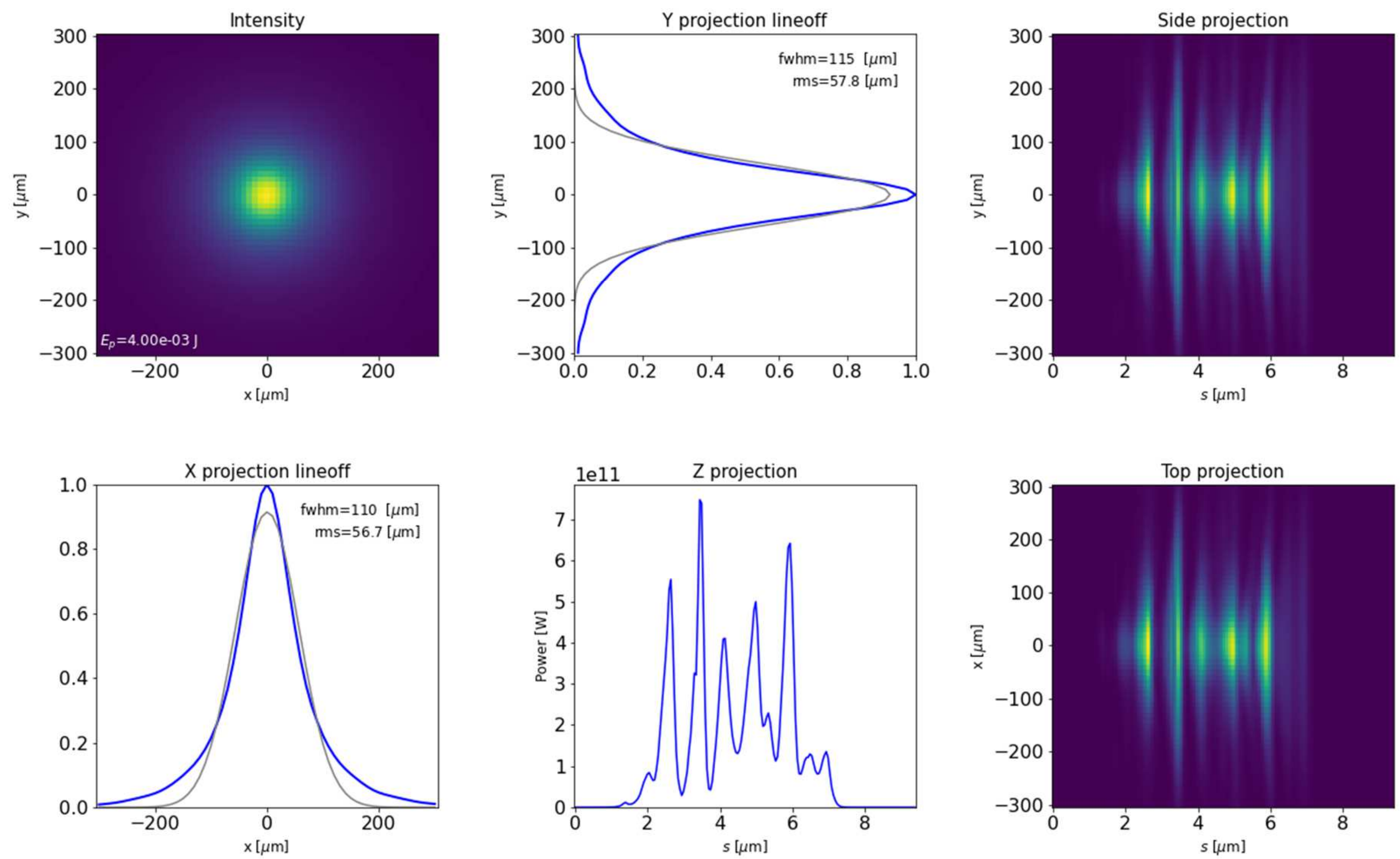
```
E_pohoton = 8000 #central photon energy [eV]
kwargs={'xlamds':(h_eV_s * speed_of_light / E_pohoton), #[m] - central wavelength
        'shape':(301,301,10),          #(x,y,z) shape of field matrix (reversed) to dfl.fld
        'dgrid':(500e-6,500e-6,1e-6), #(x,y,z) [m] - size of field matrix
        'power_rms':(3e-6,10e-6,0.1e-6),#(x,y,z) [m] - rms size of the radiation distribution (gaussian)
        'power_center':(0,0,None),     #(x,y,z) [m] - position of the radiation distribution
        'power_angle':(0,0),           #(x,y) [rad] - angle of further radiation propagation
        'power_waistpos':(-5,-15),     #(Z_x,Z_y) [m] downstrean location of the waist of the beam
        'wavelength':None,             #central frequency of the radiation, if different from xlamds
        'zsep':None,                   #distance between slices in z as zsep*xlamds
        'freq_chirp':0,                #dw/dt=[1/fs**2] - requency chirp of the beam around power_center[2]
        'en_pulse':None,               #total energy or max power of the pulse, use only one
        'power':1e6,
        }
dfl = generate_gaussian_dfl(**kwargs);
```

Characterizing the radiation

Plotting routine

plot_dfl(dfl, z_lim=[], xy_lim=[], figsize=4, cmap=def_cmap, legend=True, phase=False, fig_name=None, auto_zoom=False, column_3d=True, savefig=False, showfig=True, return_proj=False, line_off_xy=True, slice_xy=False, log_scale=0, cmap_cutoff=0, vartype_dfl=None**)**

domains=**'st'**, ('**s**pace' or '**k-**inverse space' + '**t**ime' or '**f**requency')

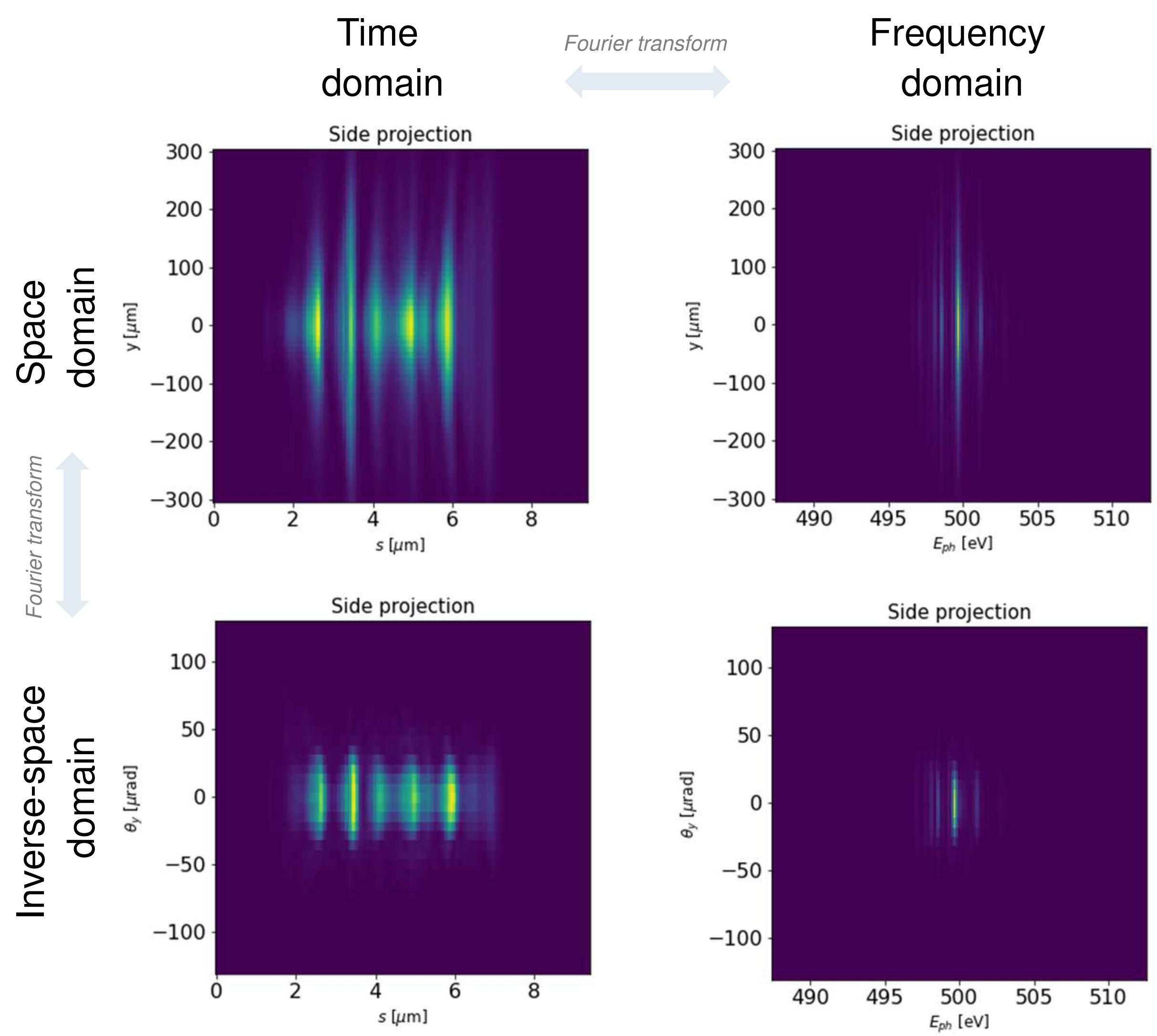


Plotting routine

plot_dfl(dfl, z_lim=[], xy_lim=[], figsize=4, cmap=def_cmap, legend=True, phase=False, fig_name=None, auto_zoom=False, column_3d=True, savefig=False, showfig=True, return_proj=False, line_off_xy=True, slice_xy=False, log_scale=0, cmap_cutoff=0, vartype_dfl=None**)**

domains=**'st'**, ('**s**pace' or '**k-**inverse space' + '**t**ime' or '**f**requency')

Inverse-space domain (angular distribution) is what we see on FEL imagers

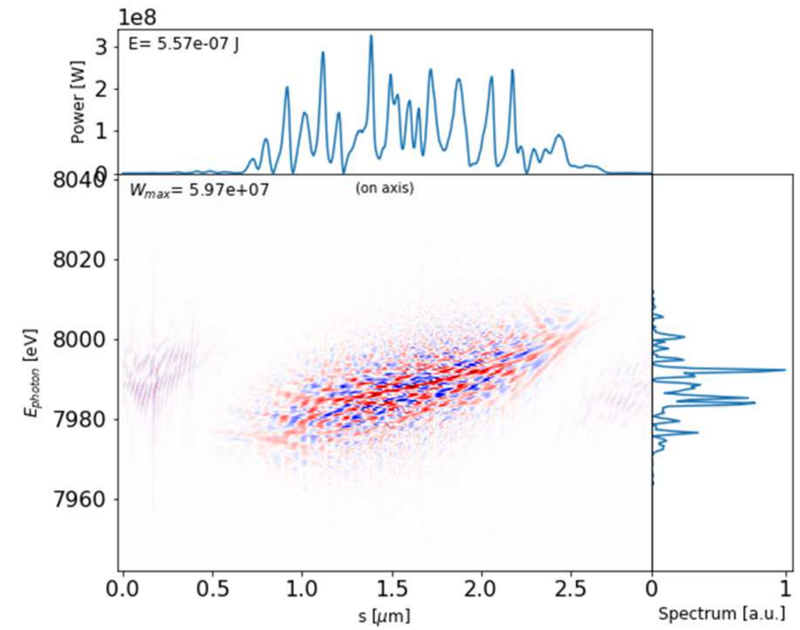


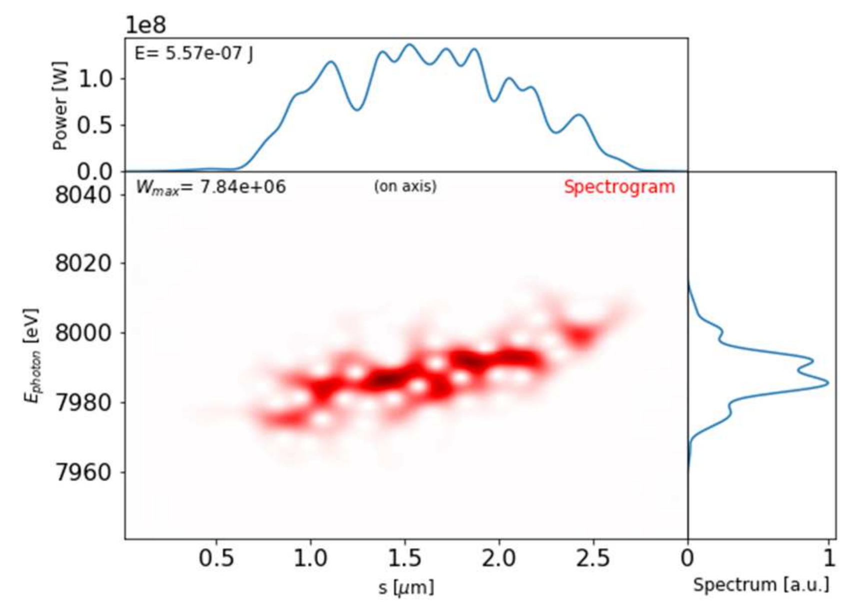
Wigner distribution (time-frequency representation to study chirps)

- wig = **WignerDistribution()**
  - wig = **wigner_dfl**(dfl, …)
  - **plot_wigner**(wig, …)
  - wig = **wigner_smear**(wig, sigma_s=0.05e-6)

- Check out the Appendices of
  S. Serkez, O. Gorobtsov, D. E. Rivas, M. Meyer, B. Sobko, N. Gerasimova, N. Kujala, and G. Geloni, *Wigner Distribution of Self-Amplified Spontaneous Emission Free-Electron Laser Pulses and Extracting Its Autocorrelation*, J Synchrotron Rad **28**, 1 (2021)

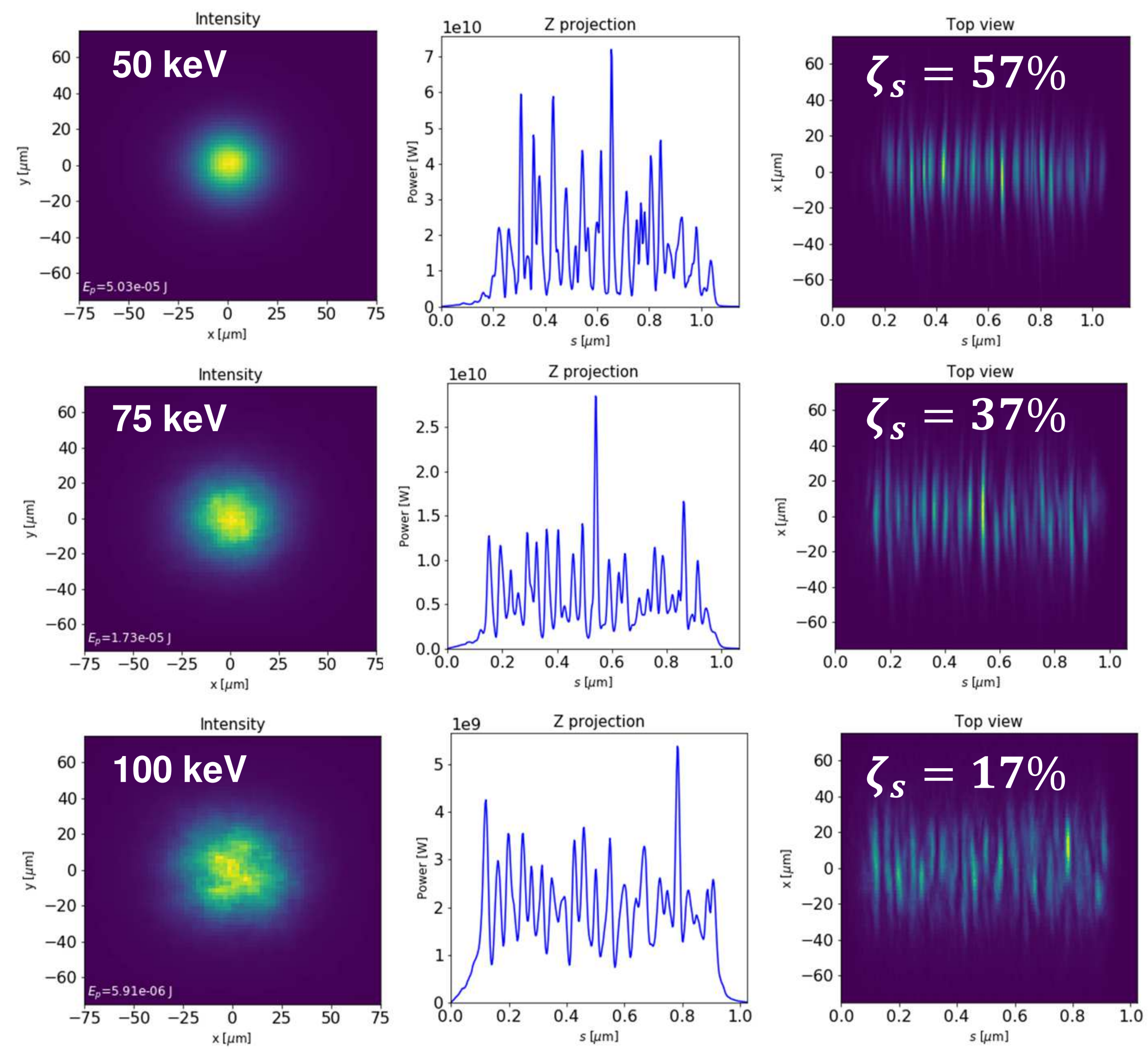
- Chirped pulses can be compressed. Group delay dispersion can be simulated by
  dfl = **dfl_disperse**(dfl, coeff=(0, 0, 0.1, …))





Coherence calculation

- $J(r_1, r_2) \equiv \langle E^*(r_1, t) E(r_2, t) \rangle$
  Mutual intensity function
- $\zeta_s = \dfrac{\int |J(r_1,r_2)|^2 dr_1 dr_2}{\left(\int I(r) dr\right)^2}$
  Degree of spatial coherence
- Is calculated with
  zeta = **dfl.coh()**
  consider first downsampling the field with
  dfl_interp(dfl, interpN=(0.3, 0.3), interpL=(1, 1), …)



Propagating and finding the waist

Free Space Field Propagation (Angular-Spectrum Propagation)

dfl.prop(z, …) method
z - propagation distance

$$H(k_x, k_y, z) = \exp\left[ik_0 z\sqrt{1 - \frac{k_x^2}{k_0} - \frac{k_y^2}{k_0}}\right]$$

- propagator for the free space

$$E(x, y, z) = \mathfrak{I}^{-1}\{\mathfrak{I}(E(x, y, 0)\ H(f_x, f_y, z)\}$$

- field at the distance z

dfl.prop_m(z, m=1, …) method with scaling parameter
m - the output mesh size in terms of input mesh size (m = L_out/L_inp)

See Schmidt, Jason Daniel. "Numerical simulation of optical wave propagation with examples in MATLAB" Bellingham, Washington, USA: SPIE, 2010

Propagation: intuitive understanding

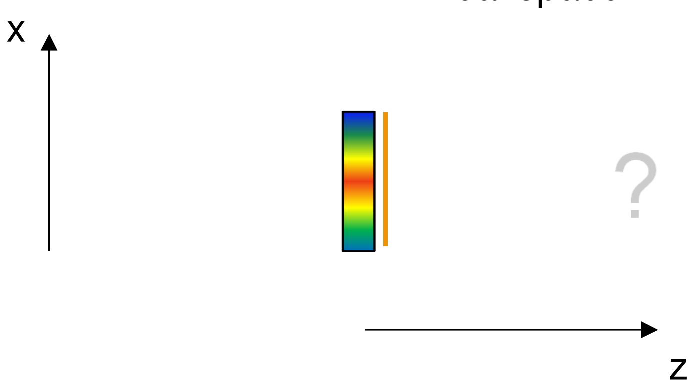
Inverse space
(angular distribution)

Real space

Fourier transform

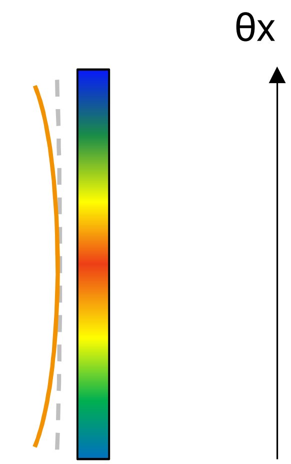
Propagation: intuitive understanding

Real space

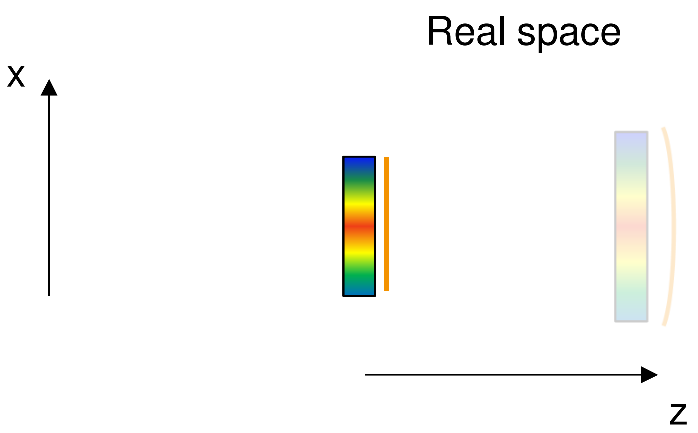


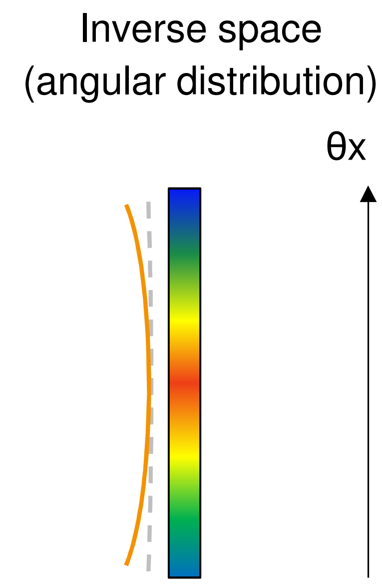
Fourier transform

Inverse space
(angular distribution)



Propagation: intuitive understanding

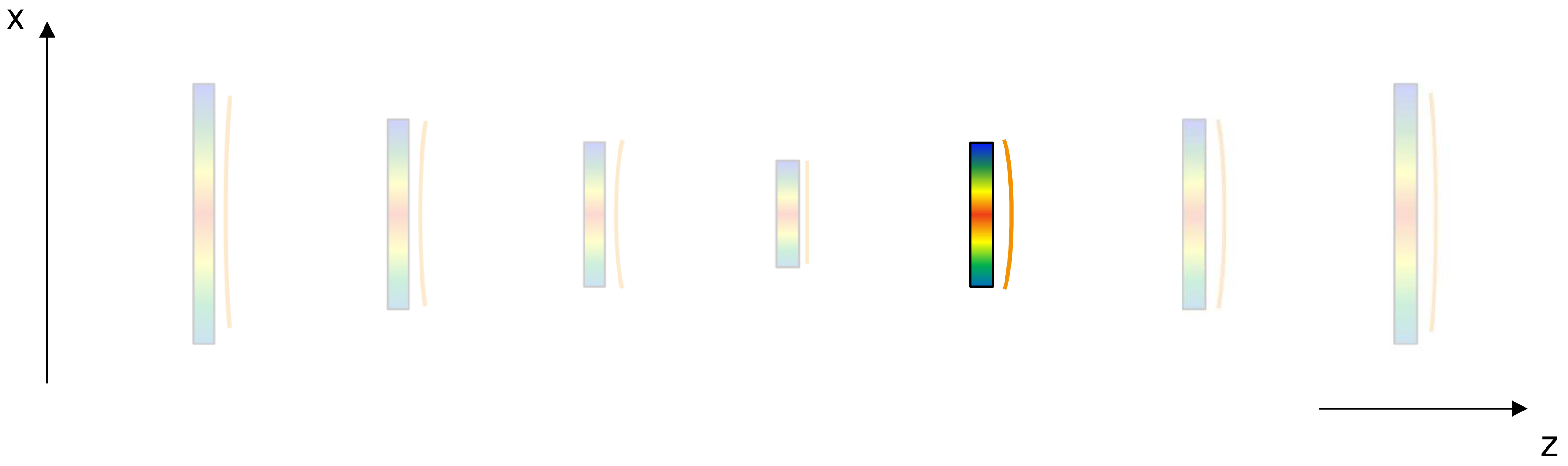




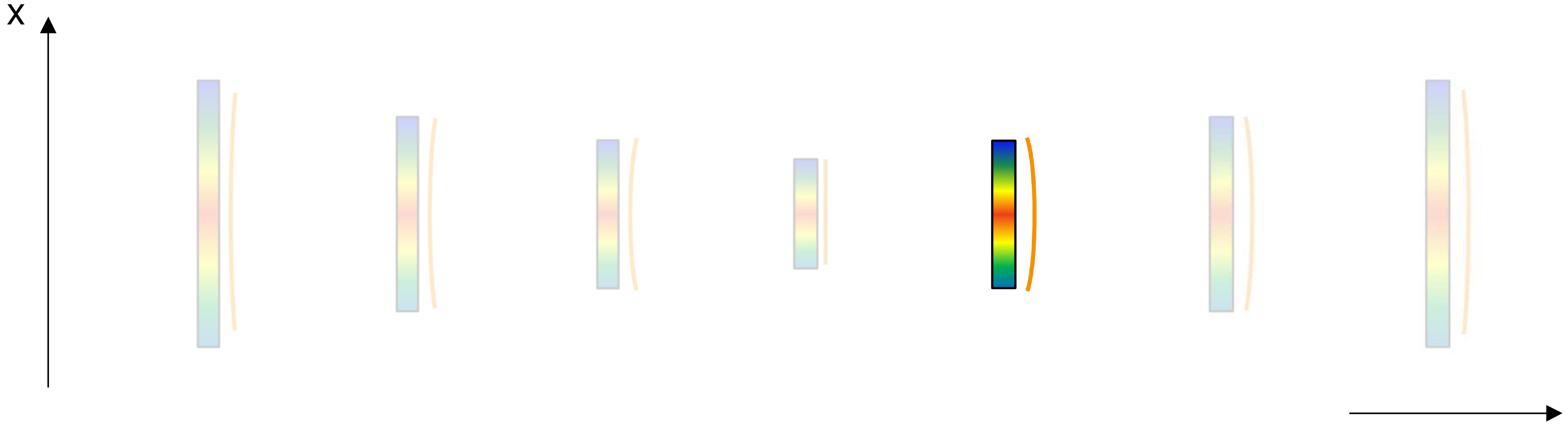
Fourier transform

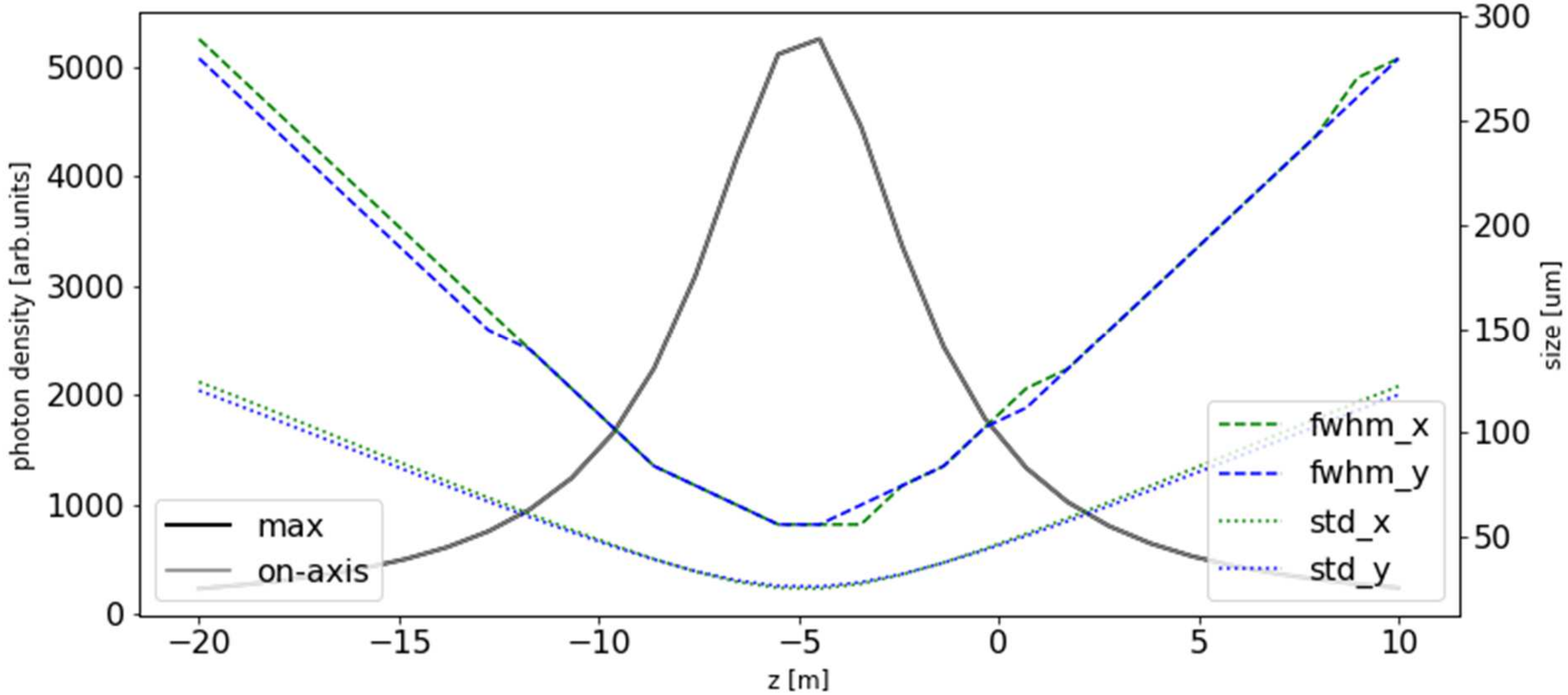
$$E(x,y,z) = \mathfrak{I}^{-1}\left\{ \mathfrak{I}(E(x,y,0)\ H(f_x,f_y,z)\right\}$$

Propagation: intuitive understanding (Gaussian beam example)



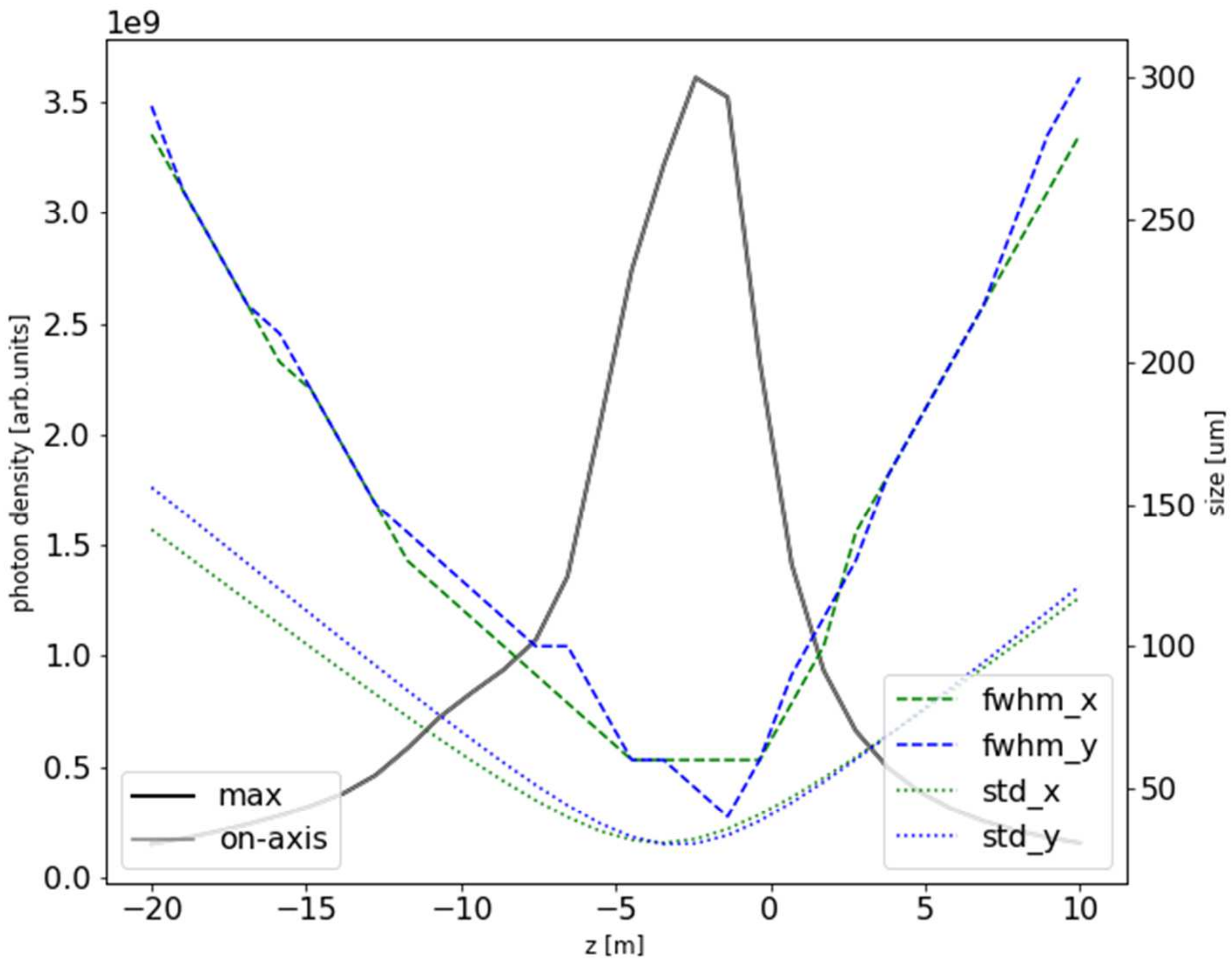
Propagation: intuitive understanding (Gaussian beam example)

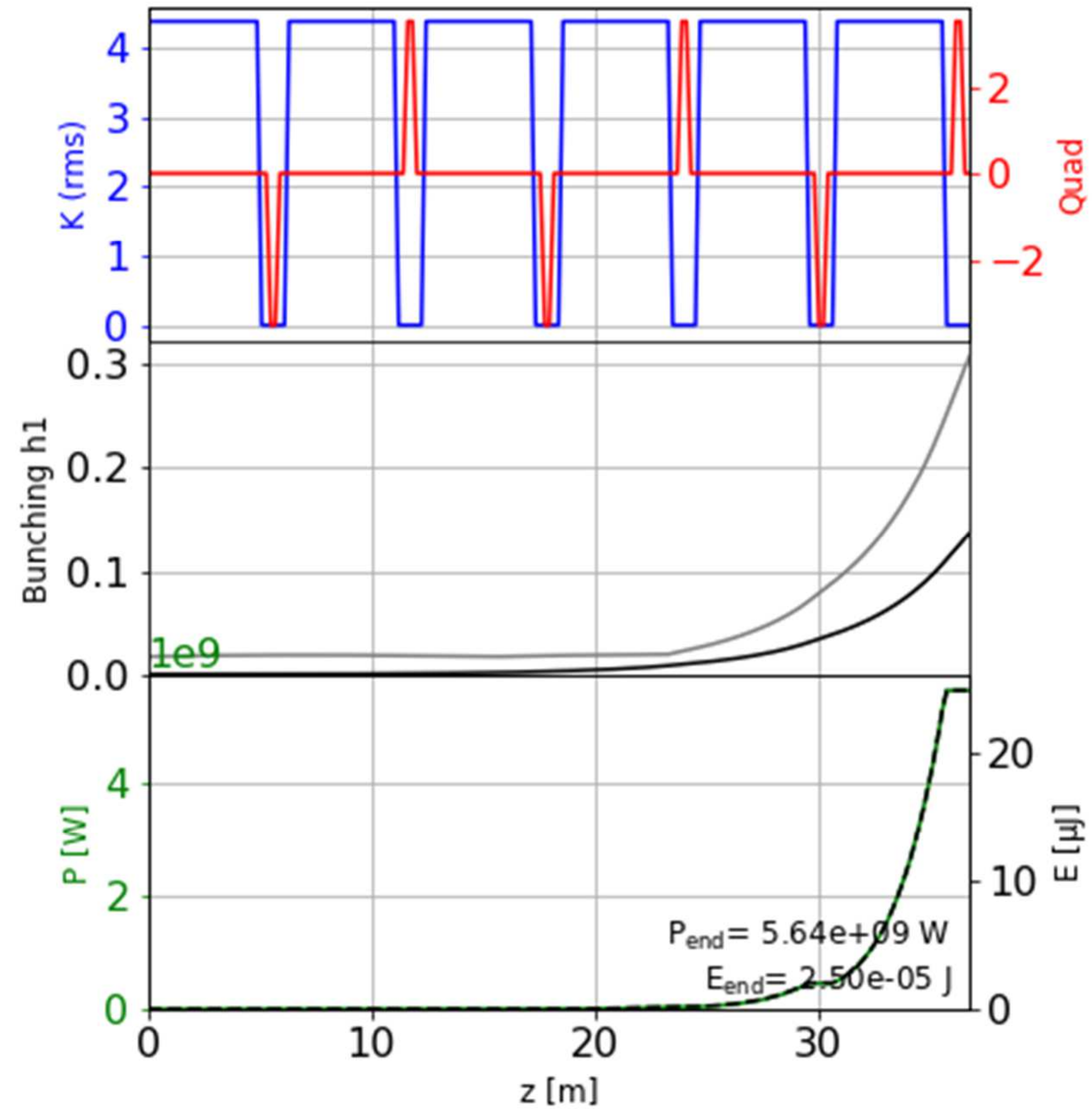




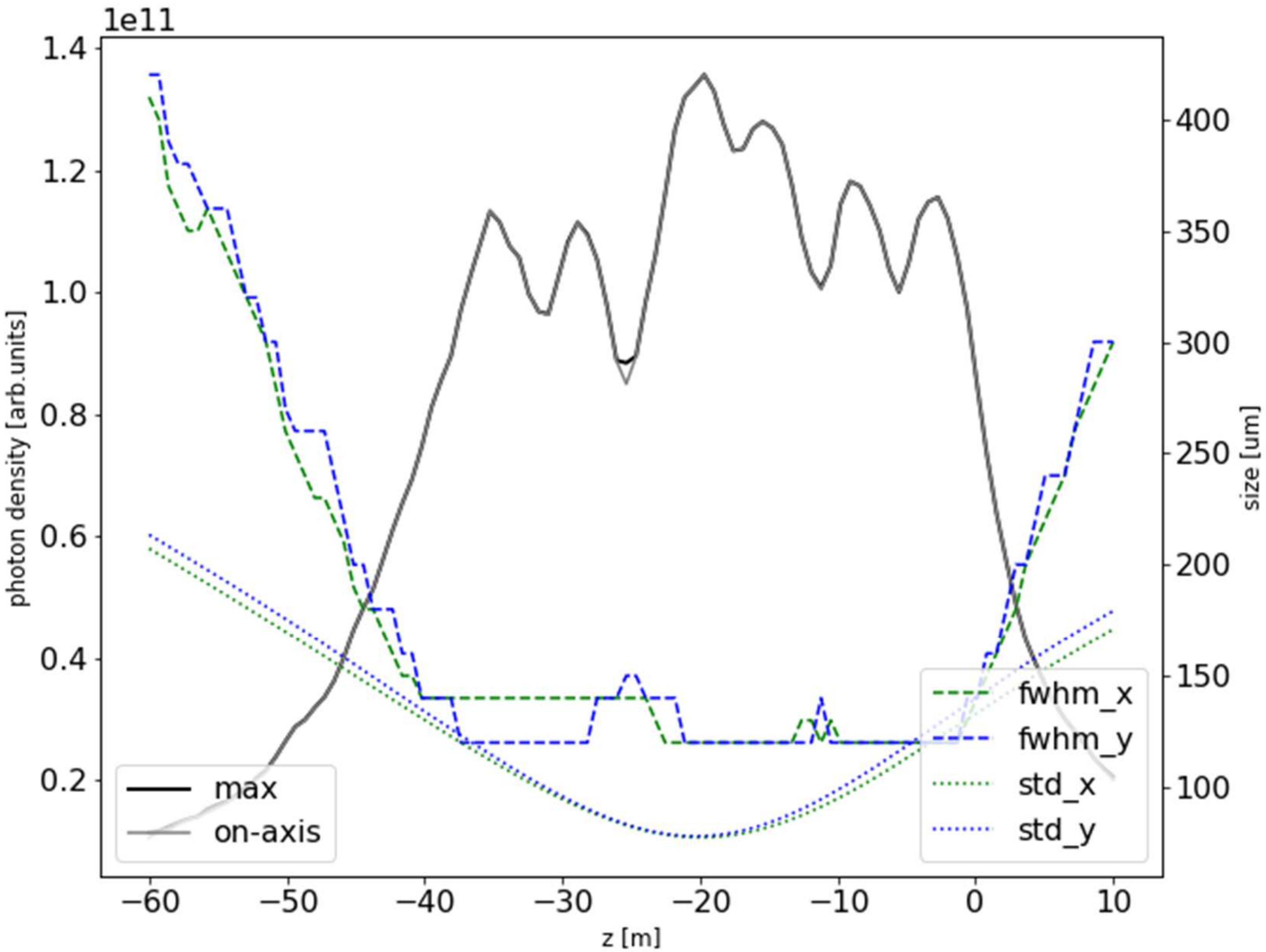

```
z_pos=np.linspace(-20,10,30)

w_scan = dfl_waistscan(dfl, z_pos)

plot_dfl_waistscan(w_scan)
```

Propagation exponential growth (SASE3, 500eV)

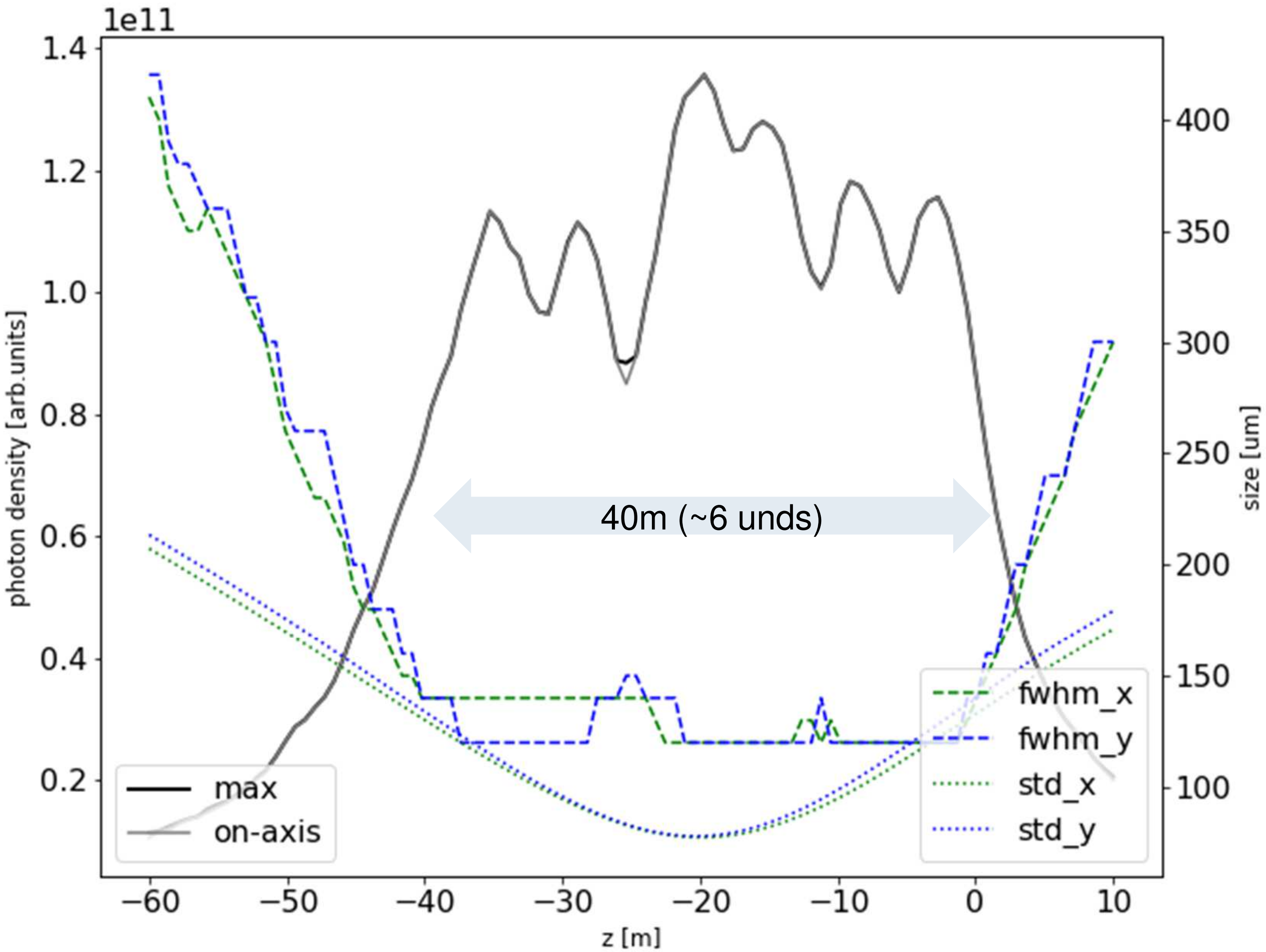


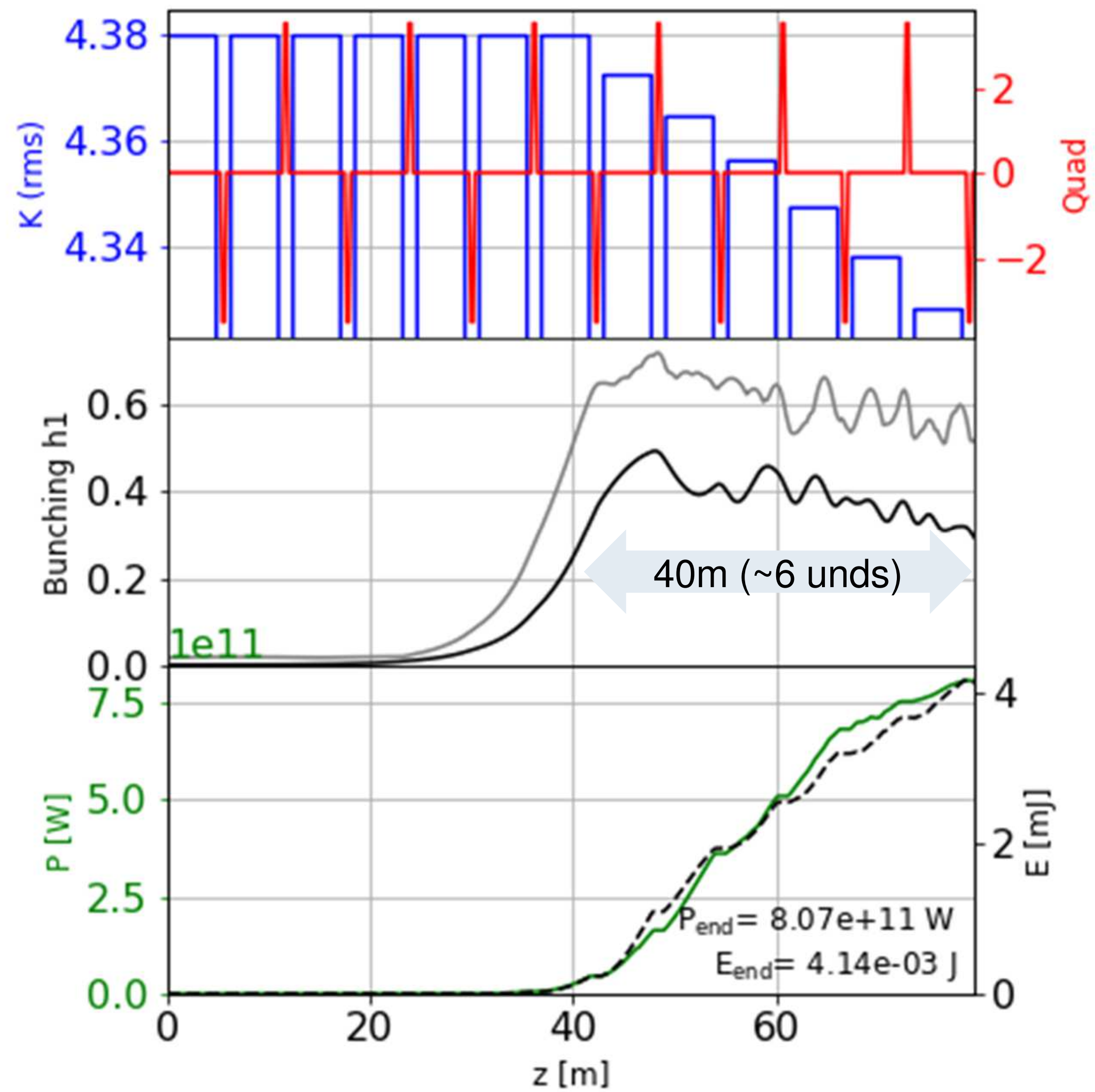


Propagation: saturation + tapering (SASE3, 500eV)

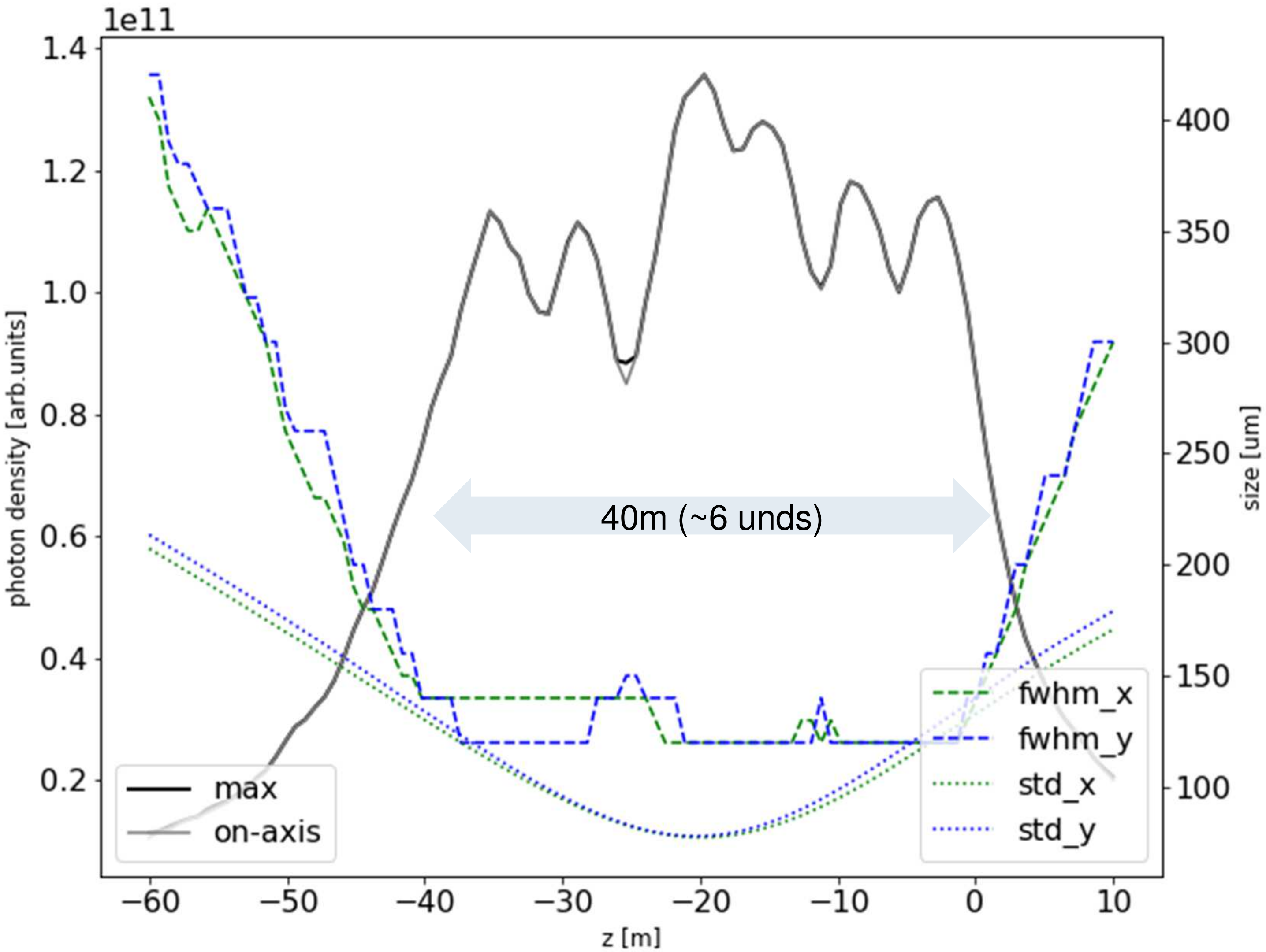


Propagation: saturation + tapering (SASE3, 500eV)

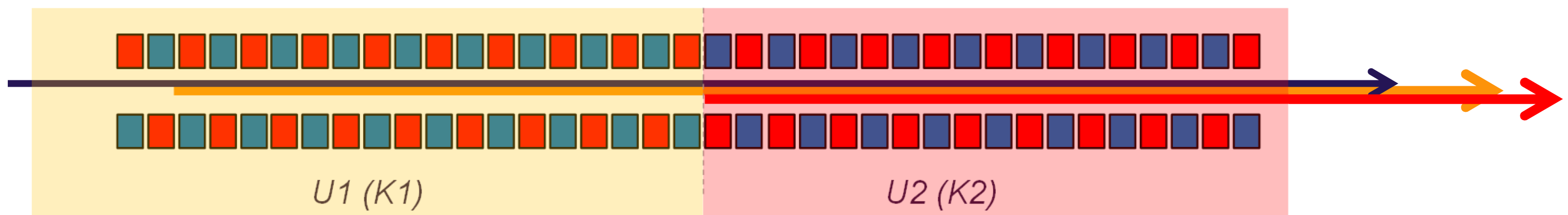


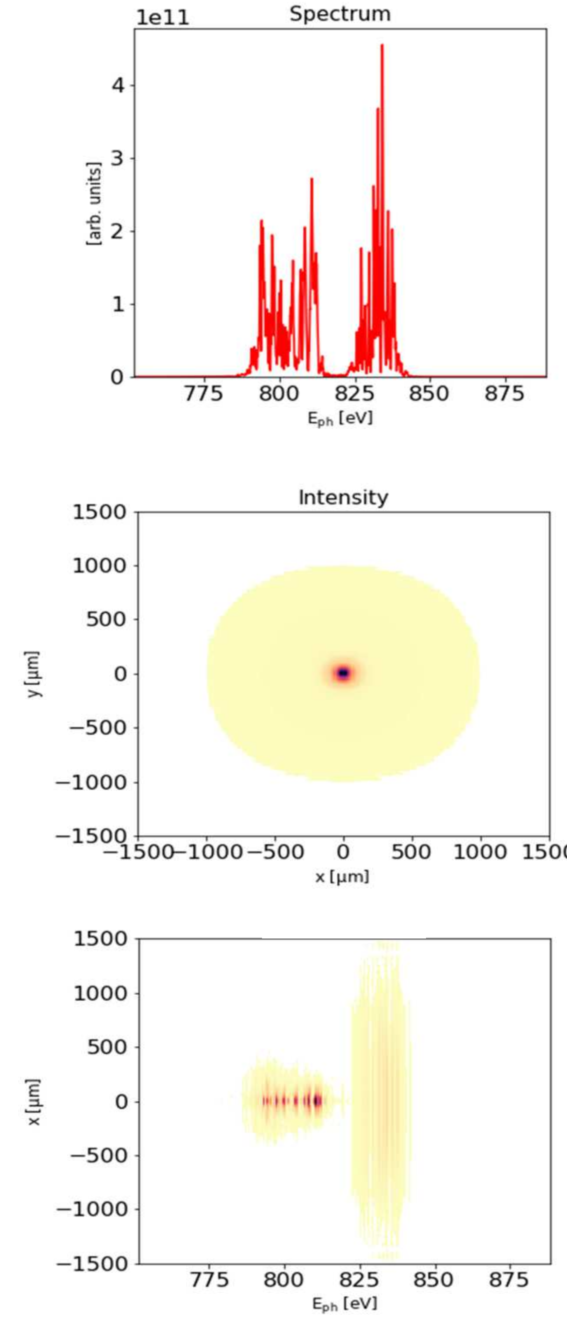


Propagation: saturation + tapering (SASE3, 500eV)

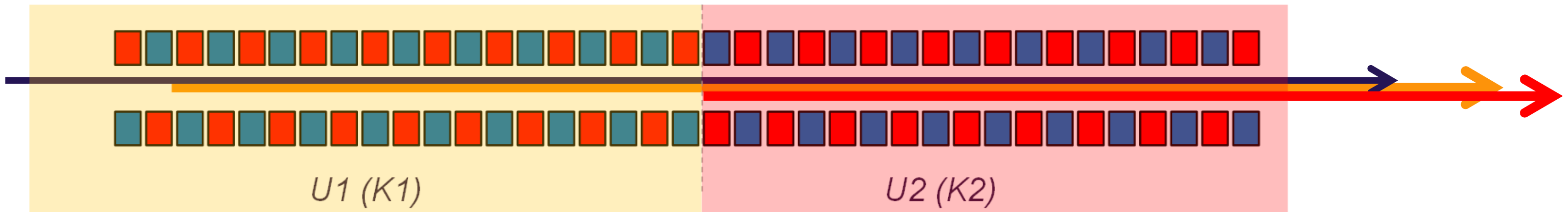


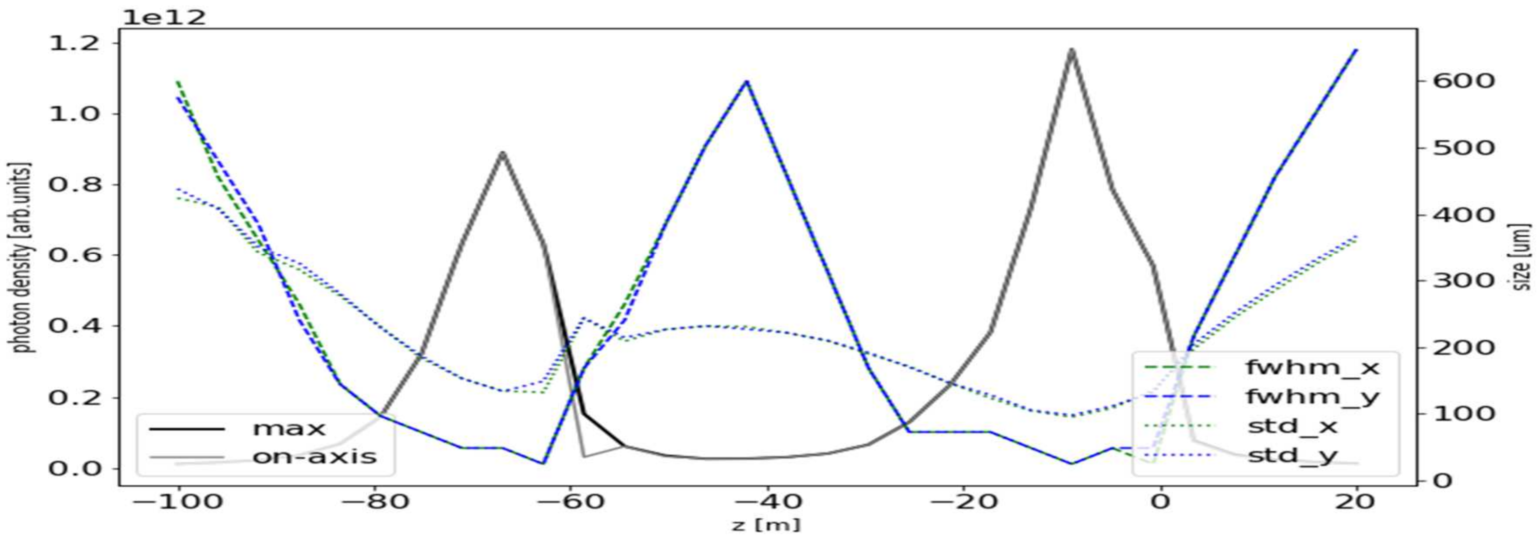
Backpropagation results -> two sources

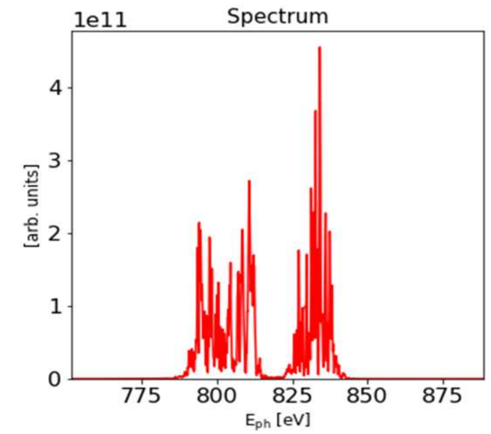


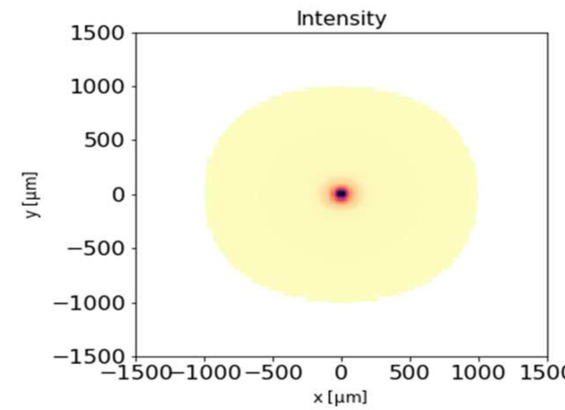


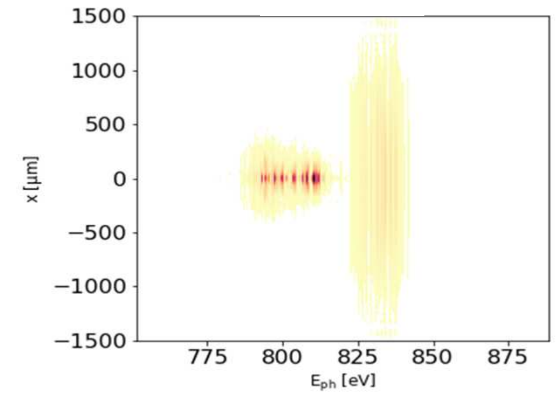
Backpropagation results -> two sources



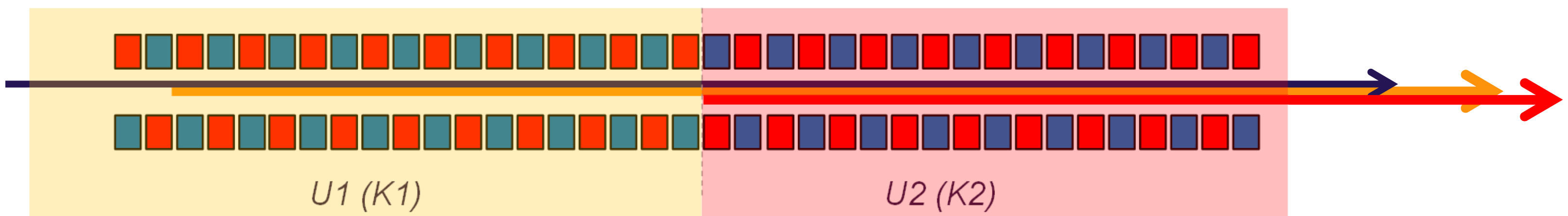


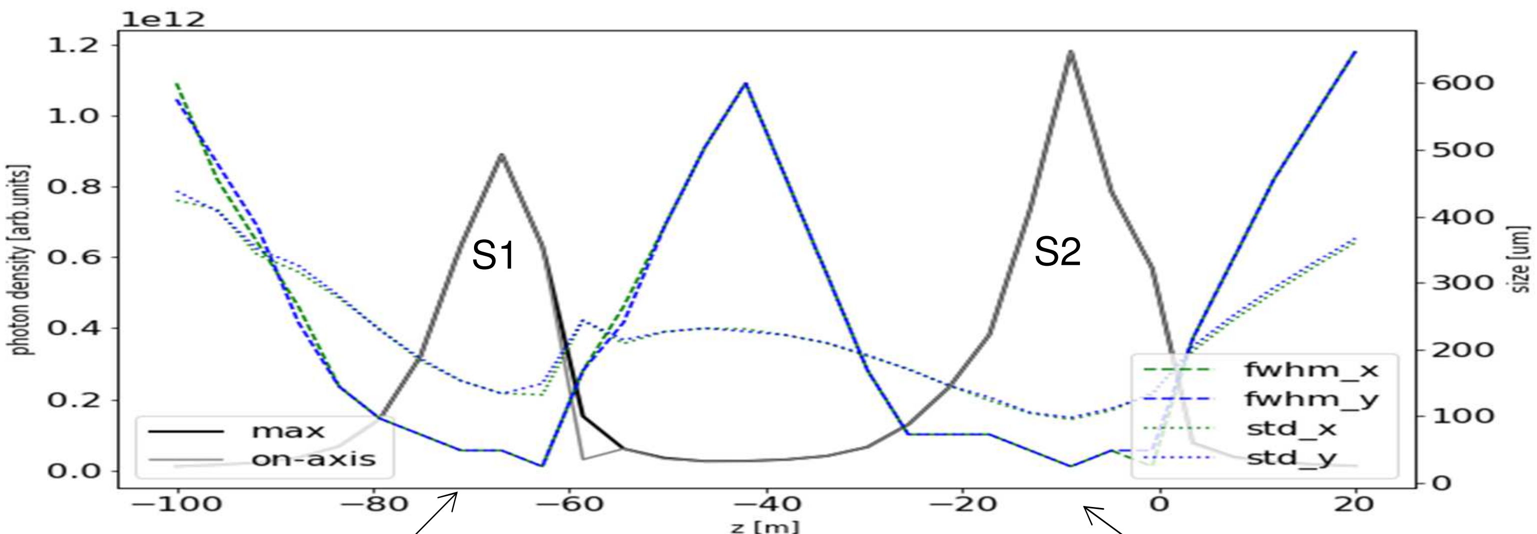


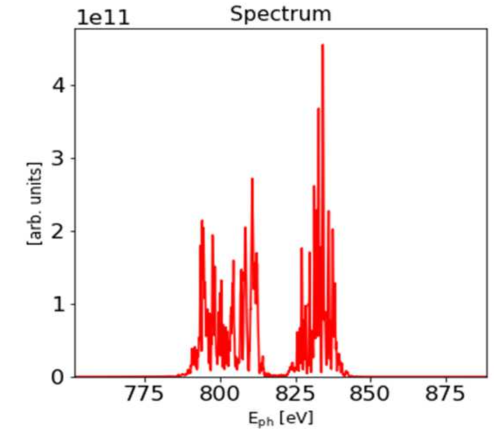


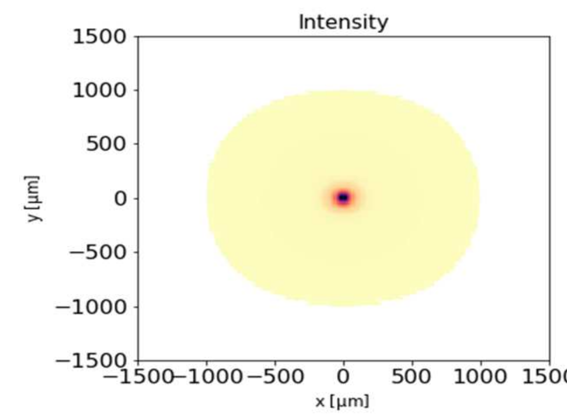


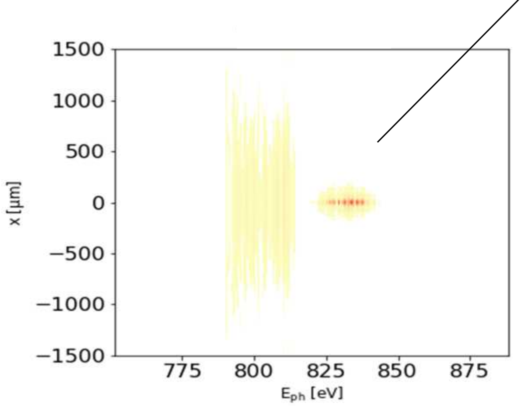
Backpropagation results -> two sources

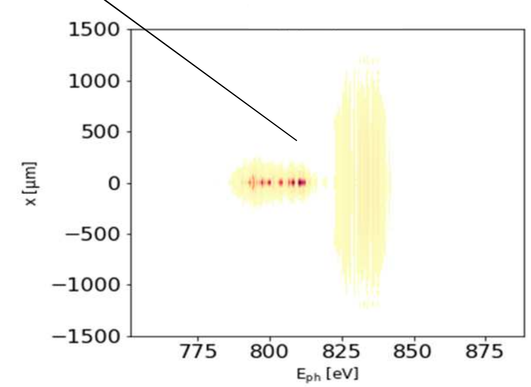


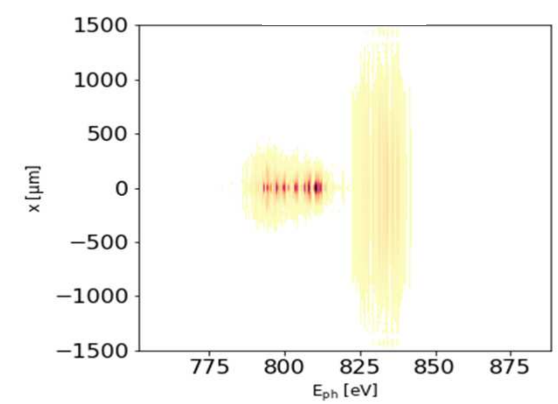




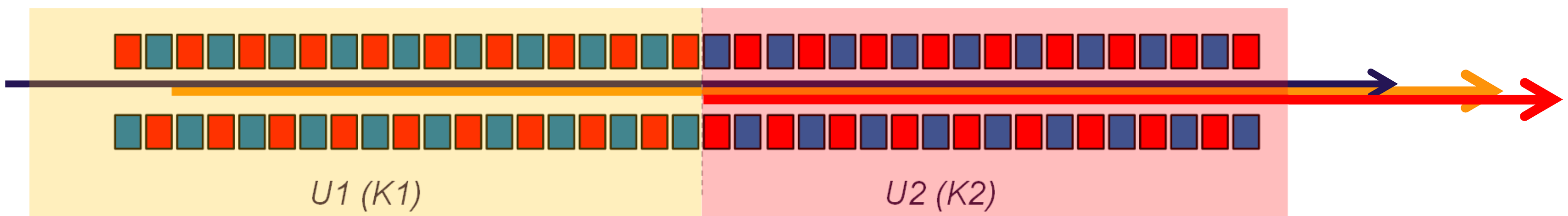


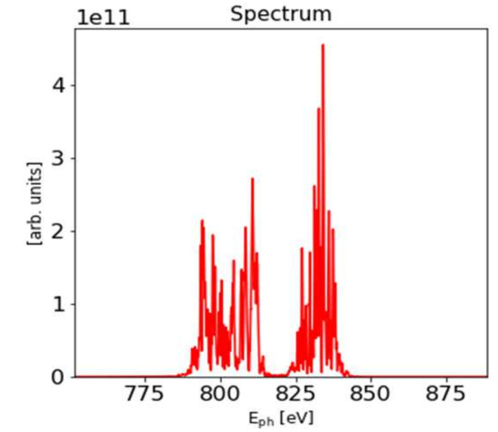


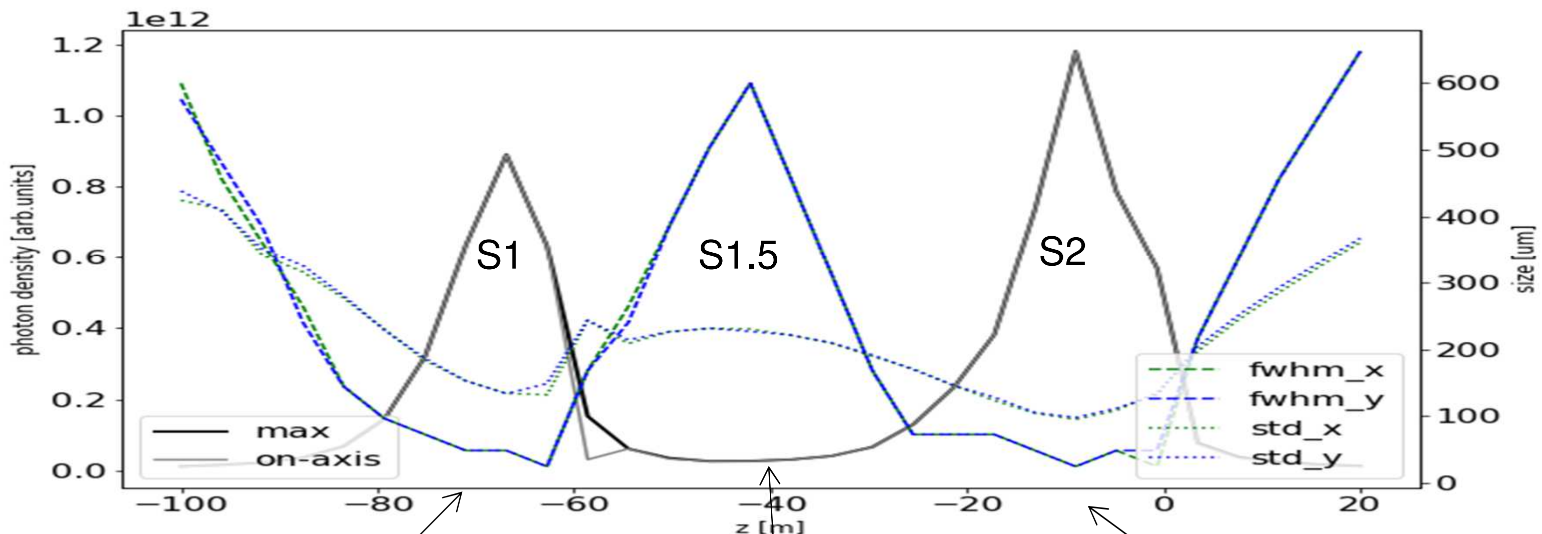


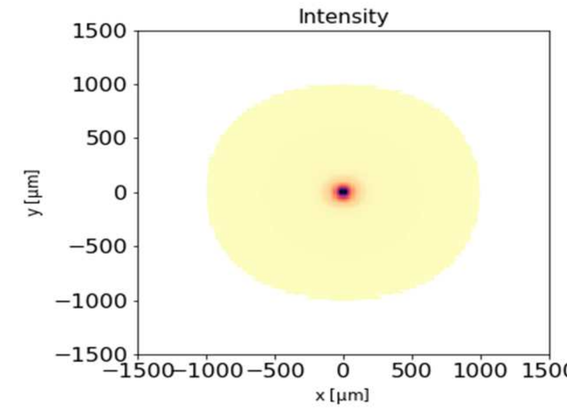


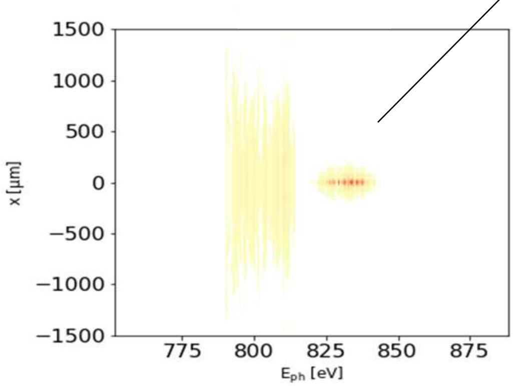
Backpropagation results -> two sources

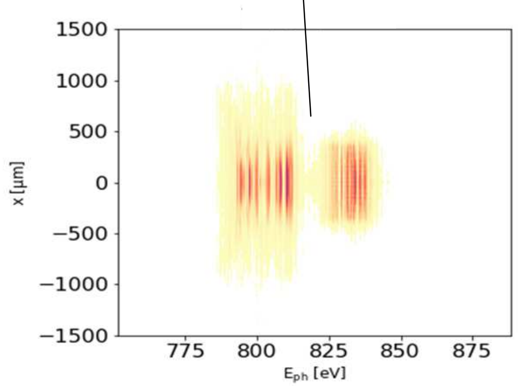


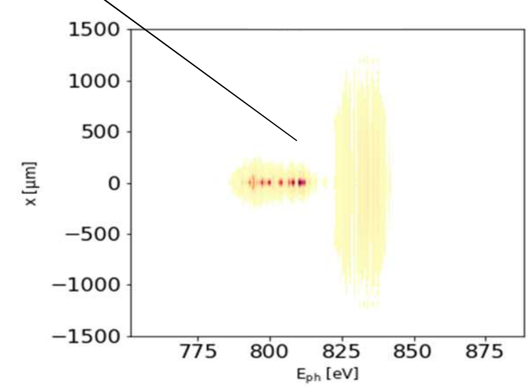


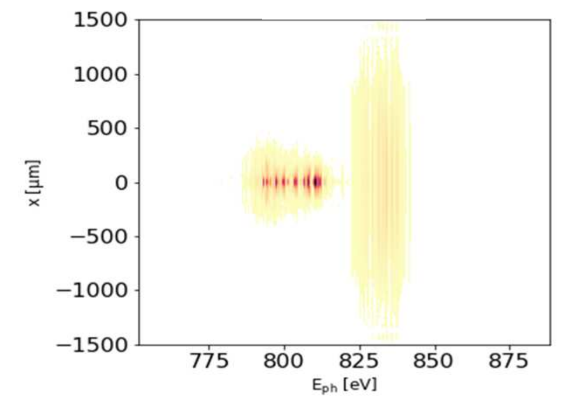








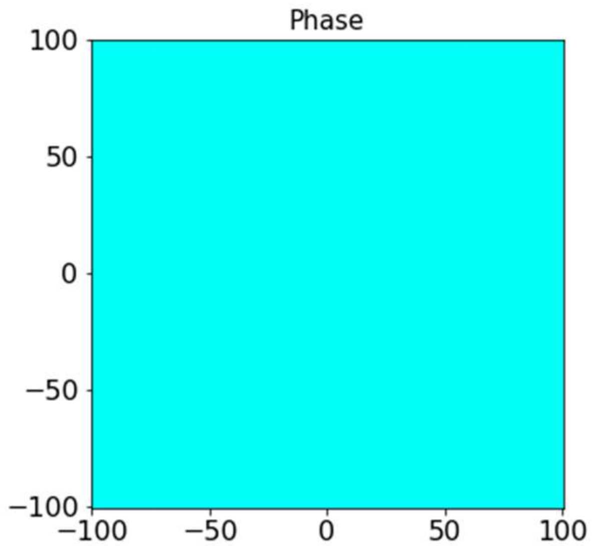




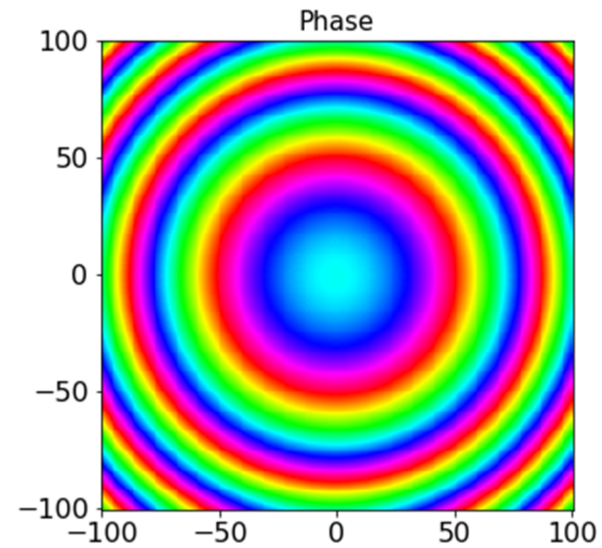
Propagating radiation to the sample

Mirrors simulation

dfl.curve_wavefront(r=inf, plane='xy', domain_z=None) method adds quadratic curvature to a wavefront in x and/or y direction

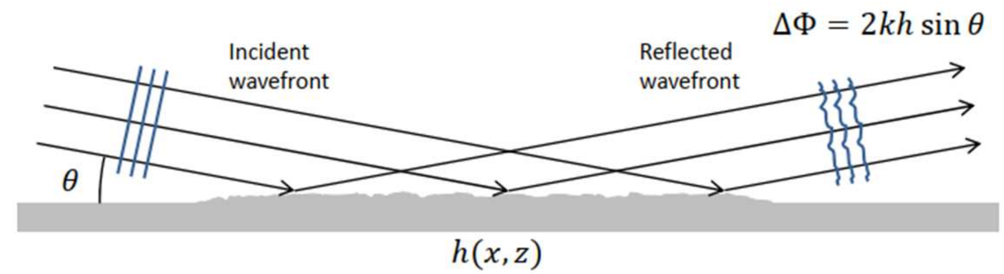


focusing == curving the phase front

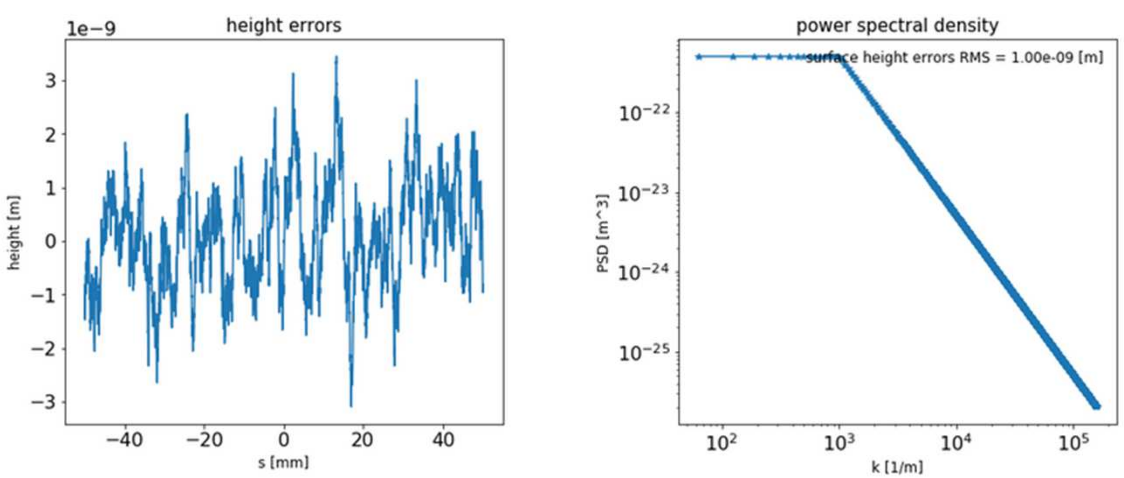


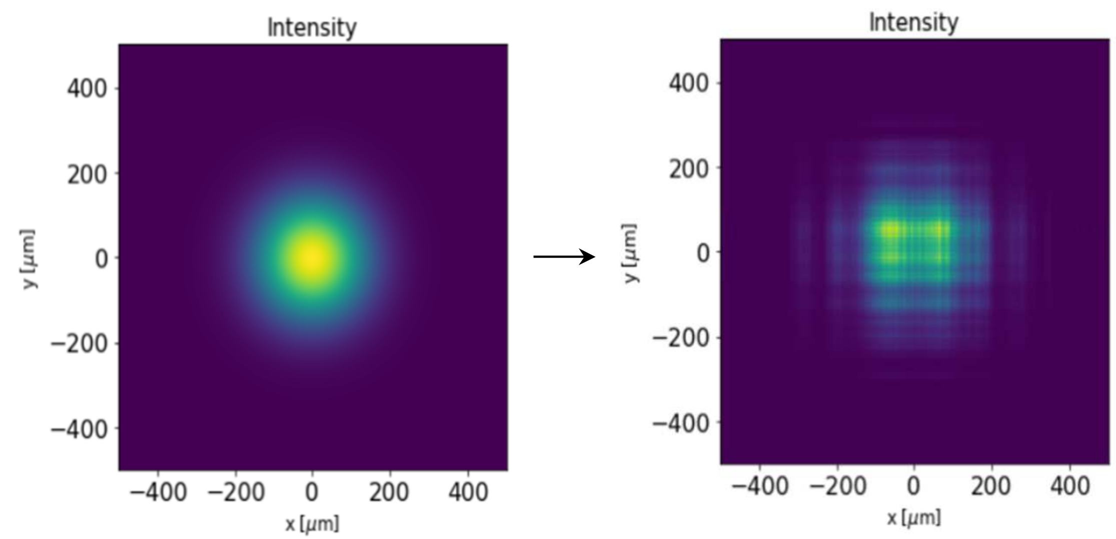
Here we ignore aberrations

Mirror reflection with height errors



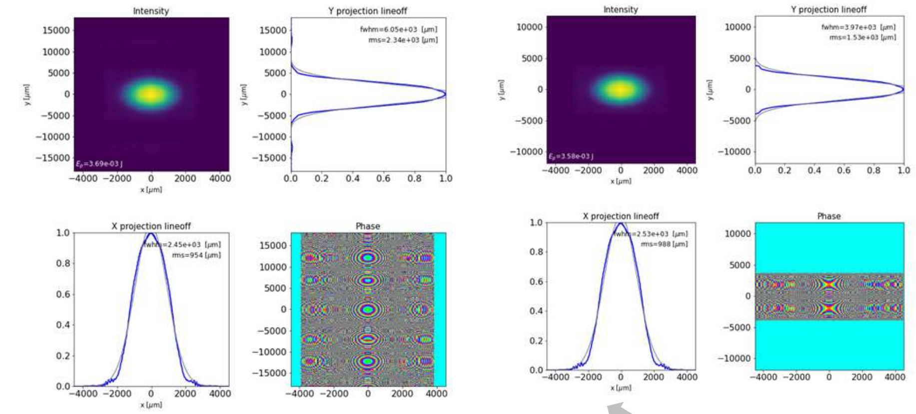
hprofile = generate_1d_profile(hrms=1e-9, …)
plot_1d_hprofile(hprofile, …)
dfl_reflect_surface(dfl1, angle=0.5, height_profile=hprofile, axis='x')

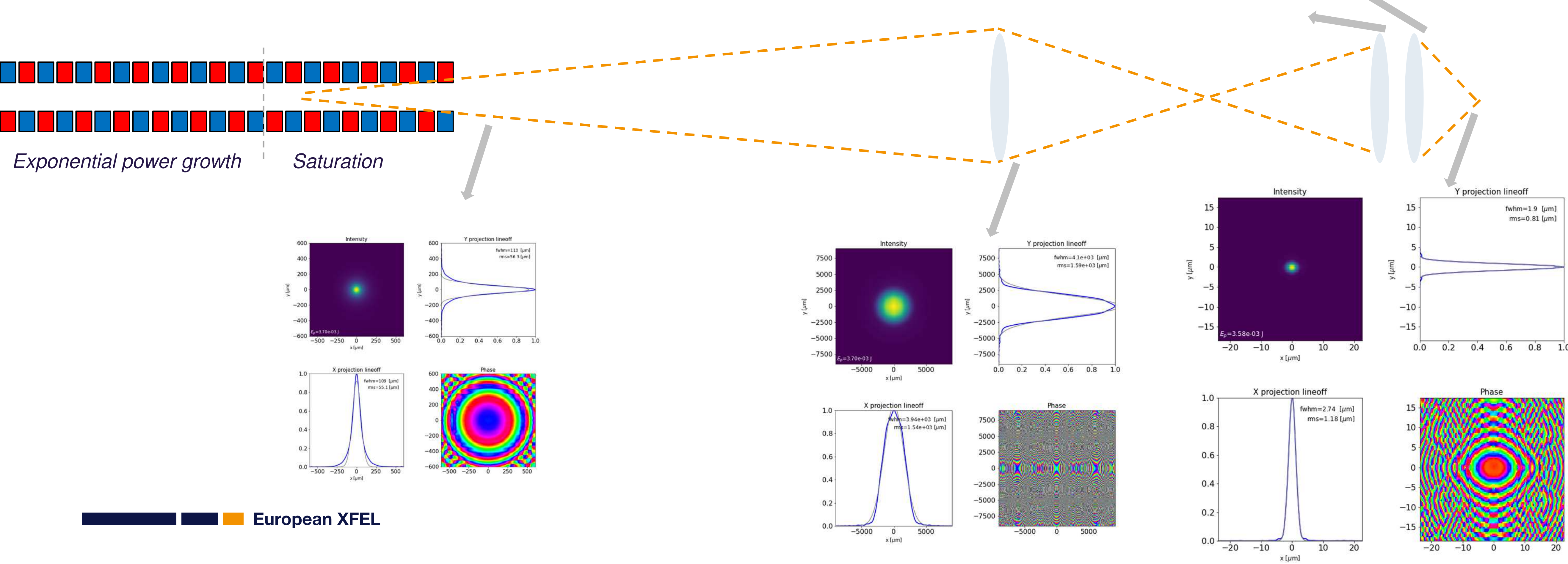




See Tutorial https://nbviewer.jupyter.org/github/ocelot-collab/ocelot/blob/master/demos/ipython_tutorials/pfs_3_imperfect_mirror.ipynb

Beamline propagation example
(500 eV at SQS with an intermediate focus)





Beamline propagation example
(500 eV at SQS with an intermediate focus)

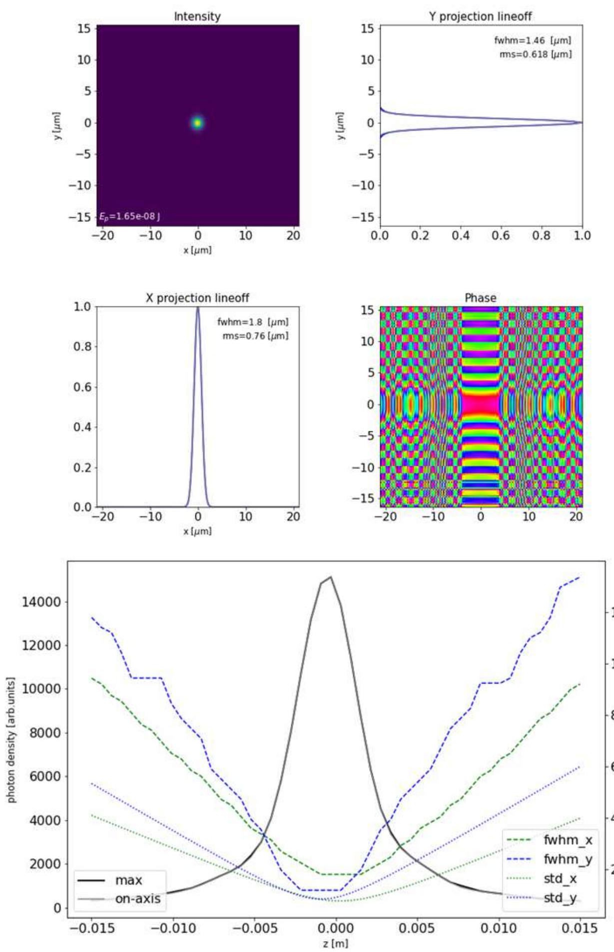
```
# -*- coding: utf-8 -*-
"""
SQS beamline
Created on Thu Oct 17 01:17:00 2019

@author: sserkez
"""

import ocelot
from ocelot.utils.xfel_utils import *
from ocelot.rad.undulator_params import *
from ocelot.gui.genesis_plot import *

plot_all = 1
scan_waist=0
modeled=0

source_size= 40e-6
kwargs={'xlamds':eV2lambda(500),
        #'rho':2.0e-4,
        'shape':(201,201,2),
        'dgrid':(1.2e-3,1.2e-3,5e-6),
        'power_rms':(source_size,source_size,1e-6),
        'power_waistpos':(-30,-30)}

def lensmaker_get_f(a,b):
    return 1/( 1/b - 1/a )

loc_source = 0
loc_bender = 283
loc_il = 374
loc_VKB = 427
loc_HKB = 429
loc_F2 = loc_HKB + 1.61 + 2.19

f_adj_VKB = -0.158
f_adj_HKB = -0.52
```

```
d_source_HKB = loc_HKB - loc_source
d_HKB_F2 = loc_F2 - loc_HKB
d_VKB_HKB = loc_HKB - loc_VKB

d_source_bender = loc_bender - loc_source
d_bender_interml = loc_il - loc_bender
d_interml_HKB = loc_HKB - loc_il

d_source_VKB = loc_source - loc_VKB

d_bender_VKB = d_source_VKB - d_source_bender
d_bender_HKB = d_source_HKB - d_source_bender


f_VKB = lensmaker_get_f(loc_VKB - loc_source, loc_F2 - loc_VKB) + f_adj_VKB
f_HKB = lensmaker_get_f(loc_HKB - loc_il, loc_F2 - loc_HKB) + f_adj_HKB

KB_angle = 13e-3
KB_length = 0.6
KB_ap = KB_angle*KB_length
KB_rms = 3e-9
KB_rms = 0e-9

hprof1 = generate_1d_profile(KB_rms, KB_length, points_number=600, seed=1)
hprof2 = generate_1d_profile(KB_rms, KB_length, points_number=600, seed=2)

if modeled:
    dfl = generate_gaussian_dfl(**kwargs)
else:
    filepath = r'D:\DESYcloud\projects\2017_03-
FEL_spectra\example_500eV_backprop\lin\run.0.s1.gout'
    filepath = r'D:\DESYcloud\projects\2017_03-
FEL_spectra\example_500eV_backprop\tap\run.0.s1.gout'
    dfl = read_dfl_file_out(filepath)
    dfl_cut_z(dfl, z=[2e-6,6e-6])

dfl = dfl_interp(dfl, interpN=(4,2), return_result=1)
if plot_all: plot_dfl(dfl, domains='st', fig_name='source', column_3d=0, phase=1)

plot_dfl(dfl)
plot_dfl(dfl, domains = 'fk')
```

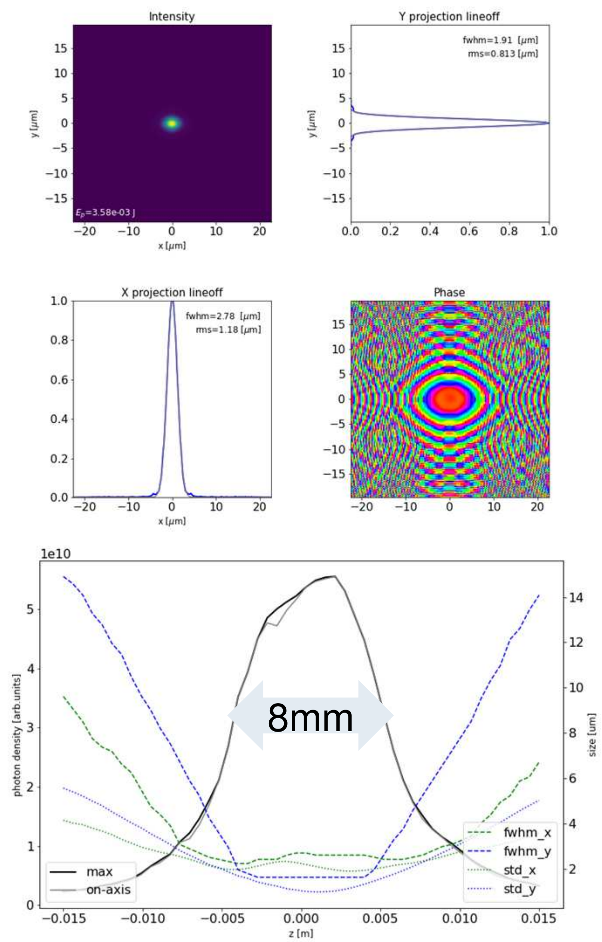
```
#%%
prop_init = 0
dfl.prop(prop_init)
dfl.prop_m(d_source_bender - prop_init, m=15)
dfl.curve_wavefront(lensmaker_get_f(d_source_bender,
d_bender_interml)-65, plane='x')
if plot_all: plot_dfl(dfl, domains='st', fig_name='bender', column_3d=0,
phase=1)


dfl.prop_m(loc_VKB - loc_bender, m=(0.5,2))


dfl_ap_rect(dfl, ap_x=KB_ap)
if plot_all: plot_dfl(dfl, phase=1, fig_name='VKB_ap', column_3d=0)
dfl.curve_wavefront(f_VKB, plane='y')
if KB_rms != 0:
    dfl_reflect_surface(dfl, KB_angle, height_profile=hprof1, axis='y')

dfl.prop_m(d_VKB_HKB, m=(1,d_HKB_F2/(d_VKB_HKB+d_HKB_F2)))

dfl_ap_rect(dfl, ap_y=KB_ap)

if plot_all: plot_dfl(dfl, phase=1, fig_name='HKB_ap', column_3d=0)

dfl.curve_wavefront(f_HKB, plane='x')
if KB_rms != 0:
    dfl_reflect_surface(dfl, KB_angle, height_profile=hprof2, axis='x')

dfl.prop_m(d_HKB_F2-0.005, m=(1/100, 1/300))


plot_dfl(dfl, phase=1, fig_name='image', column_3d=0)
#%%

if scan_waist:
    dflw = dfl_waistscan(dfl, np.linspace(-0.015, 0.015, 50))
    plot_dfl_waistscan(dflw)
```

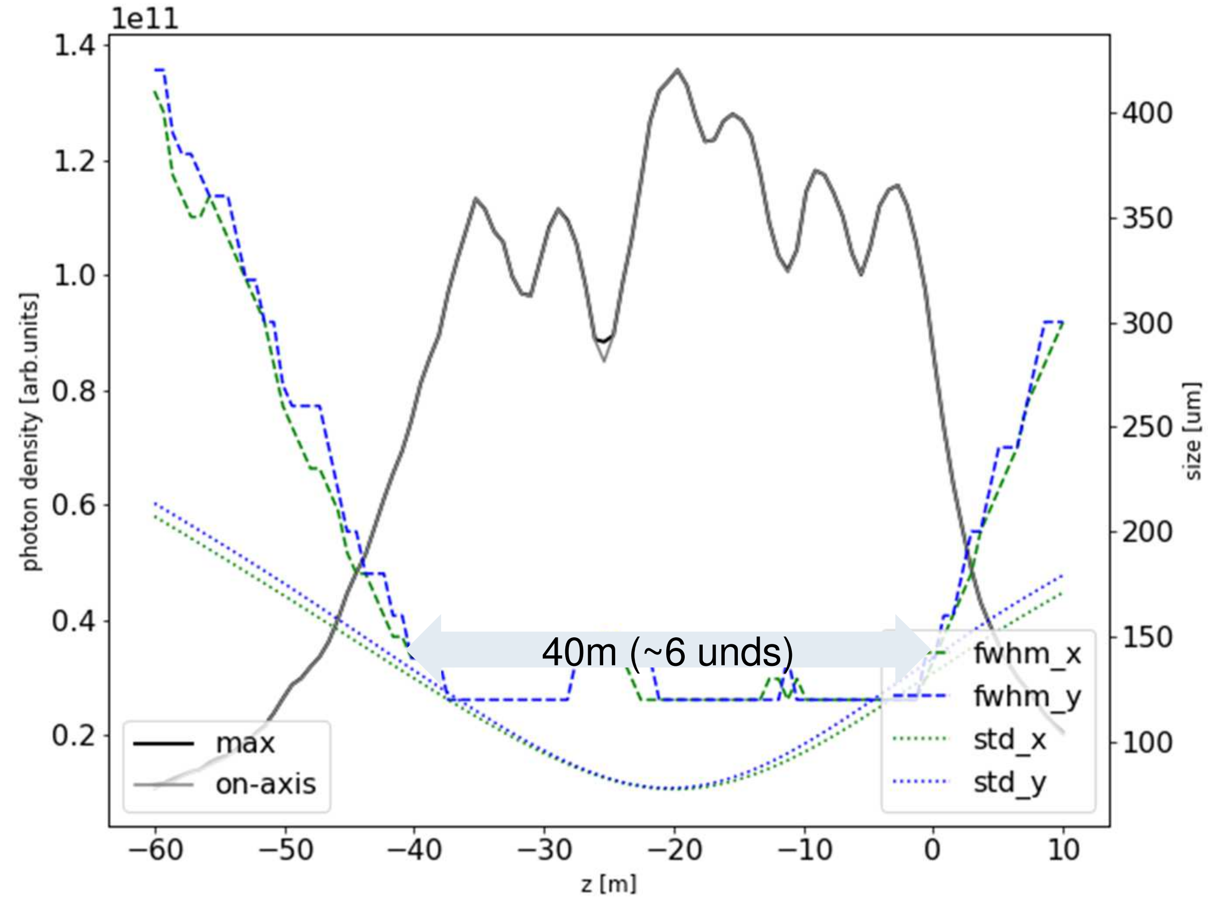
Image 3d shape depends on radiation source and transport

Gaussian model



FEL at saturation

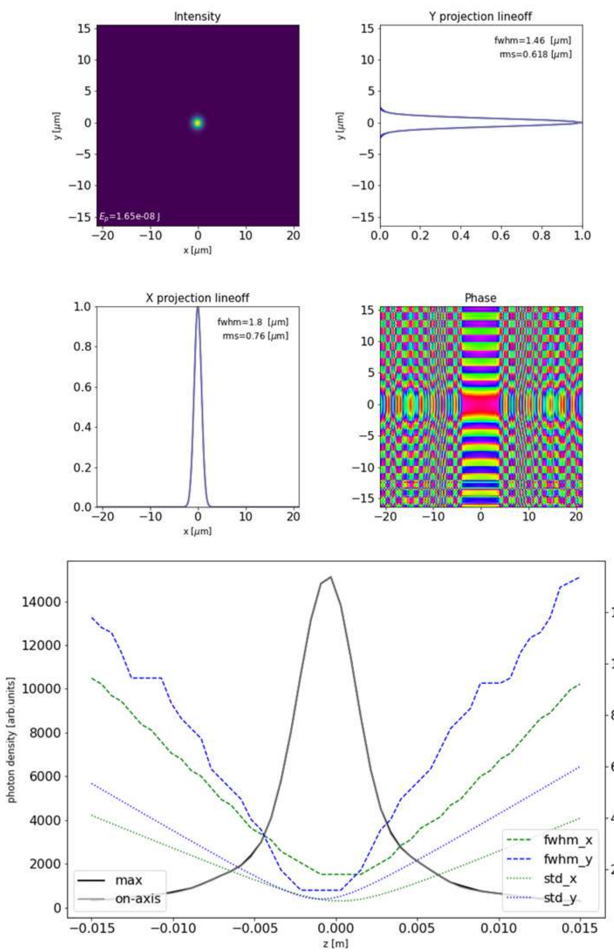




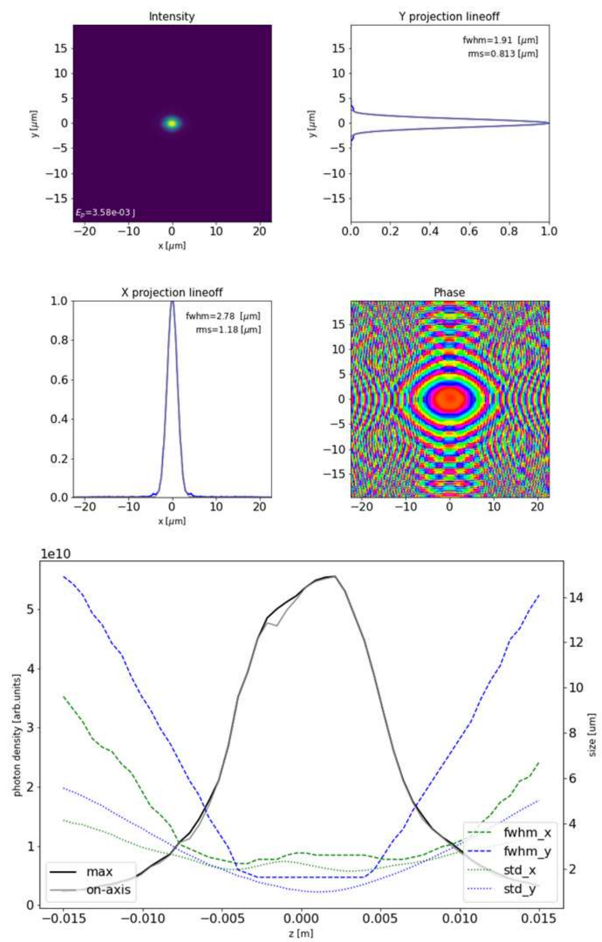
- Image focus depth is M^2 times shorter, where M is the optical demagnification

Image 3d shape depends on radiation source and transport

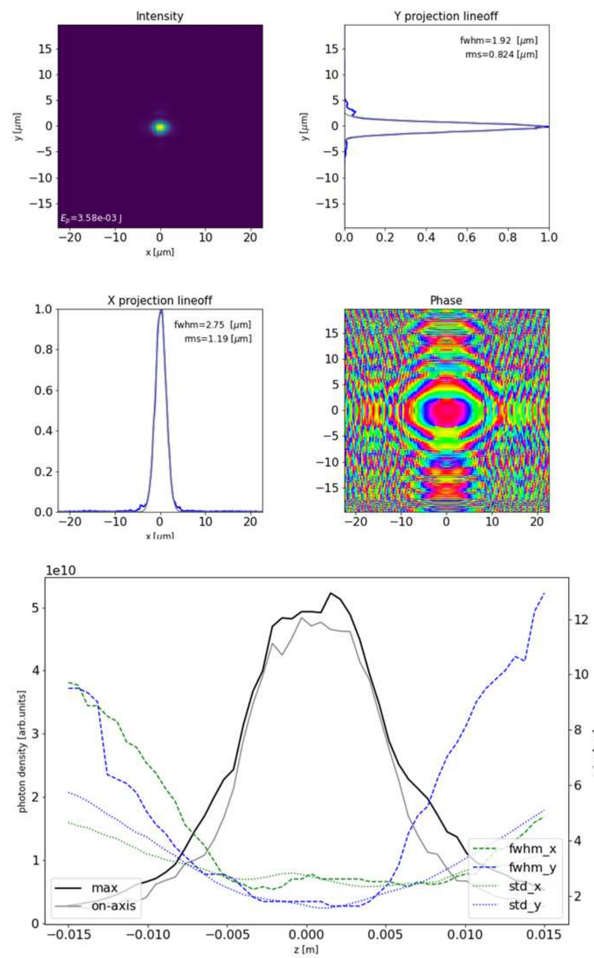
Gaussian model



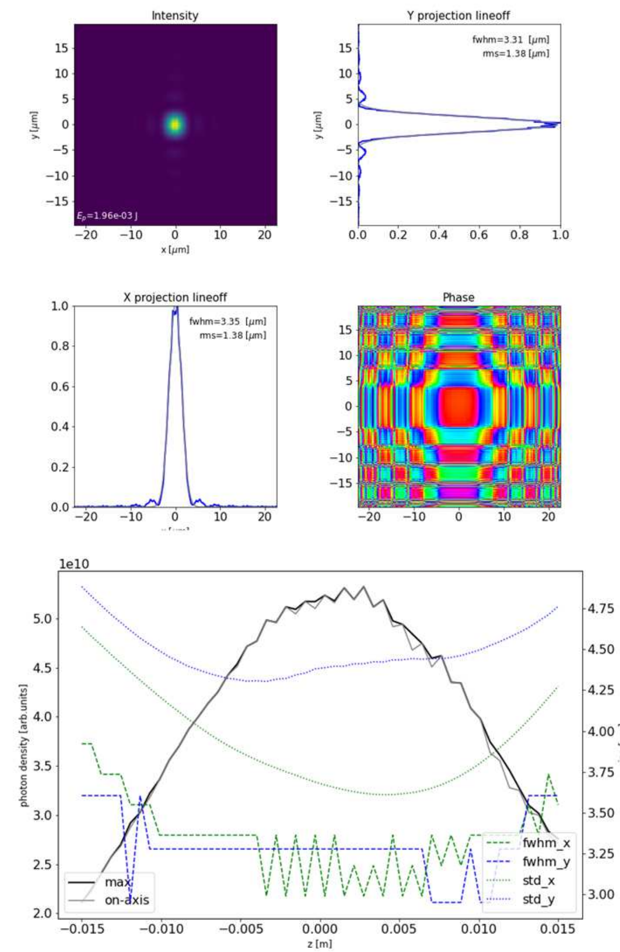
FEL at saturation



FEL at saturation
+ 10nm rms KB height error

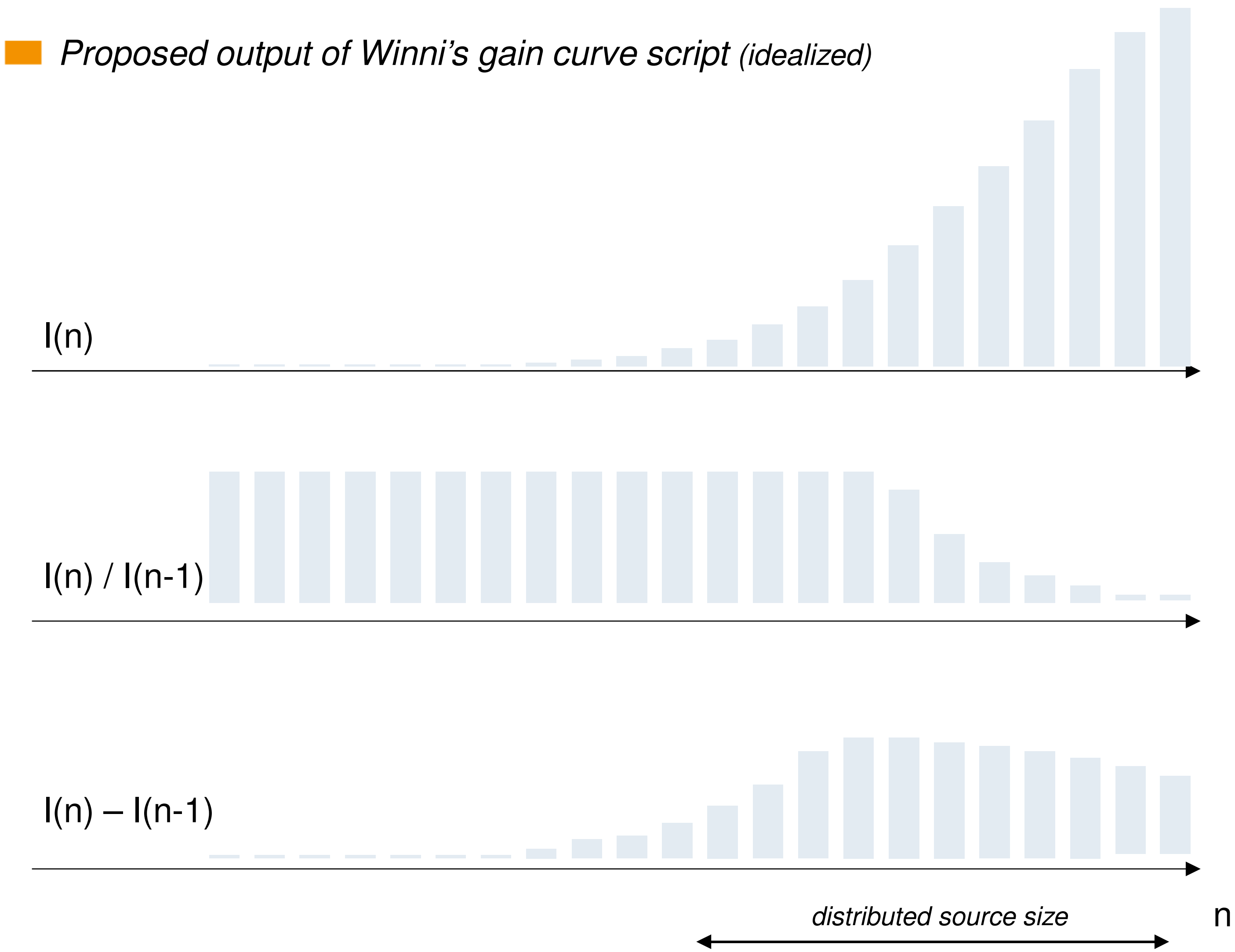


FEL at saturation
+ apertured KBs



Can we provide more info to the instruments?

- *Proposed output of Winni's gain curve script (idealized)*



- Suggestion:
  - When measure gain curve, plot $I_n - I_{(n-1)}$
  - This may be useful for instruments since shows contribution per cell
    -> how is the source distributed
  - Values $I_n / I_{(n-1)}$ would indicate gain in linear regime
- However, intensity growth does not uniquely describe the source
  - Phase of phase shifters (of tapering profile) may shape the wavefront.
  - Can such manipulation (at an expense of the flux) increase brightness?

Summary

Thanks Gianluca, Takanori and Sergey

- With Ocelot one can
  - Read Genesis2 and Genesis4 radiation output and model Gaussian beams
  - Convert radiation to different domains and plot it
  - Calculate coherence, Wigner distribution
  - Propagate through free space, rough mirrors, etc.
  - Scan radiation waists
- Adding another undulator does not necessarily increase peak photon density (for divergent soft X-rays)
- Radiation distribution is as accurate as our understanding of e-beam and imperfections/misalignments in undulator (Genesis assumes that everything is perfect by default)
- Analyzing radiation on the sample is tricky and instrument-specific. Better to limit ourselves to the source characterization
- TODOs:
  - Systematic study of the sources? Wavefront manipulation in undulator?
  - More optical elements
  - Higher abstraction levels (constructing optical beamline)
- We (FPH) usually add features when we need them. Pull requests are welcome ☺

Young developers of OCELOT

Andrei Trebushinin

Mykola Veremchuk

Tutorial links at https://github.com/ocelot-collab/ocelot