**E2S simulation tutorial**

**JI LI**

**22/09/2020 Version 2.0**

**Introduction**

This tutorial will give an overview of the brightness and tuning curves, power density simulations in E2S and the way to build a beamline optimisation by E2S-SRW optimiser and E2S-SHADOW optimiser. It will describe the code and the command to execute and analyse data. The purpose of this report is to enable readers to understand how the code works and where necessary, to make adjustments to achieve new objectives.

**Content**

[Flux and intensity simulation 3](#_Toc50386586)

[Brightness and tuning curves calculation 4](#_Toc50386587)

[Power density simulation 6](#_Toc50386588)

[Power density simulation in free space propagation 6](#_Toc50386589)

[Power density simulation along beamline 6](#_Toc50386590)

[E2S-SRW optimiser 7](#_Toc50386591)

[Set up an optimisation 7](#_Toc50386592)

[Beamline optimisation only 7](#_Toc50386593)

[BL+twiss optimisations 8](#_Toc50386594)

[Results analyse 8](#_Toc50386595)

[E2S-SHADOW optimiser 10](#_Toc50386596)

[Create a beamline model 10](#_Toc50386597)

[Build SHADOW3-python 11](#_Toc50386598)

[Beamline optimisation 12](#_Toc50386599)

[Partially coherent simulation 13](#_Toc50386600)

[Acknowledgements 14](#_Toc50386601)

# Flux and intensity simulation

These two simulations all start from the input file.

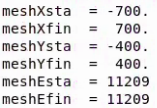
Choose the calculation type in Input\_example.input

**calc\_type =** : ‘multie’ is for the intensity simulation and ‘flux’ is for the flux calculations.

The undulator and **the first slit size** and **position** are defined in the input file.

The slit position is defined at ‘slitZ=’. ‘slitDX=’ and ‘slitDY=’ are used to define the slit size in horizontal (X) and vertical (Y). Photon energy used to calculate the flux is defined by ‘Ephot\_ini’ and ‘Ephot\_end’.

For the intensity simulation, a specific energy should be used and an initial window size should be defined as bellow (unit is μm),



The core file to calculate the intensity is : **e2s\_SRW/SRW\_intensity.py**

The core file to calculate the flux is : **e2s\_SRW/ SRW\_flux.py**

**Command to run: python E2S.py Input\_example.input**

The data will be saved in **e2s\_SRW/SRW\_I13d/**

To analyse the data:

Command for plot the intensity:

**python e2s\_SRW/ANALYSIS/ ana\_intensity\_new.py**

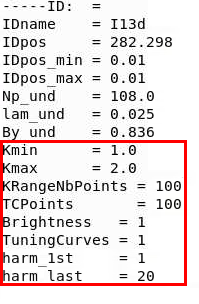
Command for plot the flux:

**python e2s\_SRW/ANALYSIS/ ana\_flux.py**

# Brightness and tuning curves calculation

This simulation is apart from SRW.

Command to run: **python E2S Input\_example.input**



The parameters in red rectangle are used to define the brightness calculation:

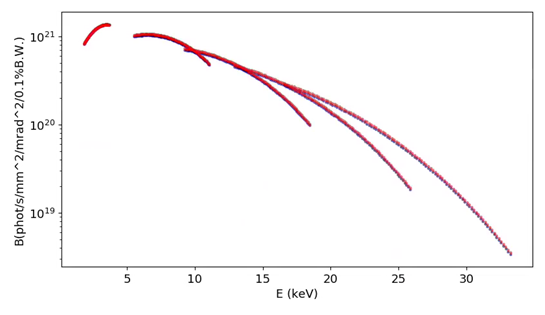
**kmin-kmax**: The range of scanning K

**Brightness, TuningCurves**: ‘1’ means calculate,’0’ means do not calculate

**harm\_1st, harm\_last**: number of harmonics chosen to show

To analyse the brightness data, or if you want to compare the brightness calculated from different lattice, use the file **E2S/plt\_brightness\_with\_ratio.py**

This script can not only present the brightness but also can show the relative variations of brightness.



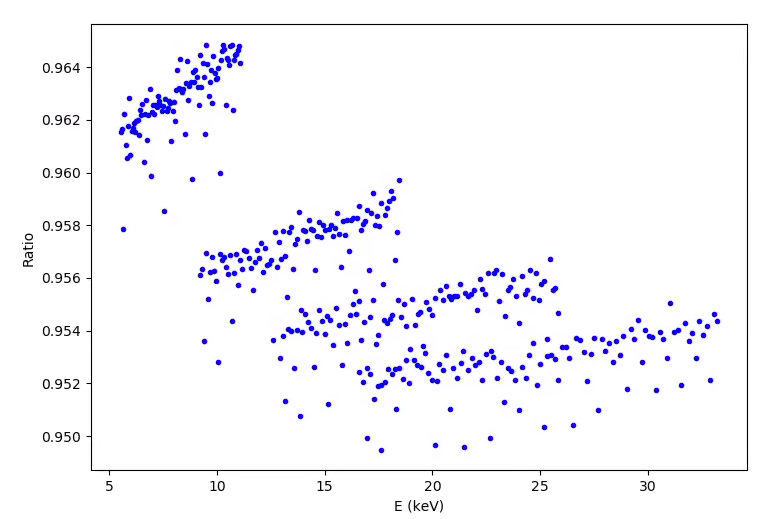


Figure 1 An example to show the comparison of brightness for two different lattices. Top picture shows the brightness at same position for two different lattices (red and blue dots represents two lattices). Bottom picture shows the ratio of brightness at different energies. 5 harmonic numbers are used.

# Power density simulation

## Power density simulation in free space propagation

The power density simulation for free space propagation has been implemented in E2S. Change the ‘calc\_type’ to ‘power’ in the input file.

Command to run: **python E2S XXX.input**

File to analyse data: **e2s\_SRW/ANALYSIS/ana\_powerdensity.py**

The total power is also calculated by integration in two planes.

## Power density simulation along beamline

First is to calculate the photon flux at slit position. (Be careful here, this step may take long time, remember to choose the window size as large as possible so that all the photons have been included) Fig. 2 shows an example of photon flux calculated through an 8mmx8mm slit (took more than 3 hours).

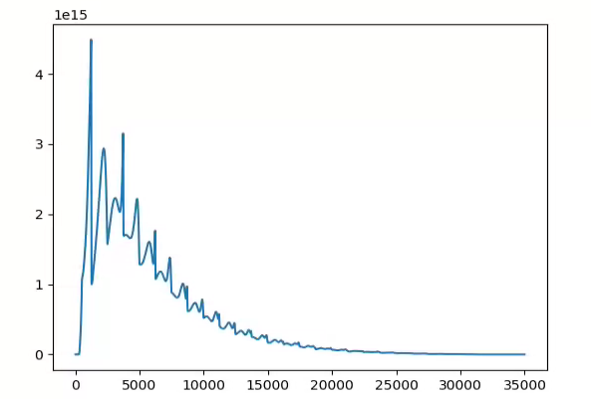


Figure 2 An example to show the photon flux through an 8mmx8mm slit.

Then generate the intensity calculation for every bin energy. In e2s\_SRW/ generate\_intensities.py, you can set the minimum photon flux value **( use min(F))** to read or set a range of energy to read in the generate\_intensities.py.

Command to run (example):

**python e2s\_SRW/ generate\_intensities.py SRW\_I13d/flux\_\_\***

All the intensities will be saved in one folder (folder name is defined in the generate\_intensities.py, then sum all the intensities using **e2s\_SRW/ read\_multiplef.py**

**All the steps which need modification or rename have been commented in the e2s\_SRW/generate\_intensities.py and e2s\_SRW/read\_multiplef.py, please follow the comments in the files before start a calculation.**

# E2S-SRW optimiser

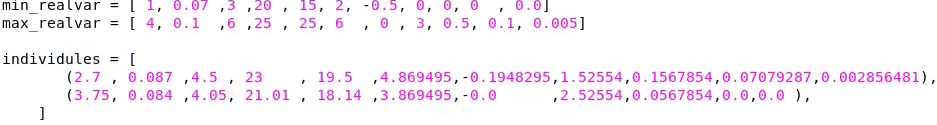
Here we use I20 scanning branch as an example to show how to use E2S-SRW optimiser to optimise beamline parameters or BL+twiss parameters. (An update to Hamilton cluster: module load python)

Command to run:

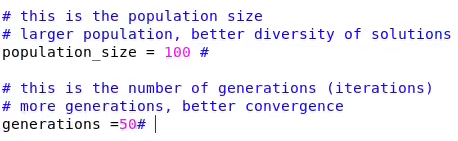
**Python nsga.py test\_kur\_par\_twiss\_and\_beamline.py**

## Set up an optimisation

test\_kur\_par\_twiss\_and\_beamline.py is wrote for twiss and beamline optimisation together, for beamline only optimisations and BL+twiss optimisations, first is to set the individual values and the range of the variables. If the number of variables changed, please be careful with other files which includes the variable dimensions as well.

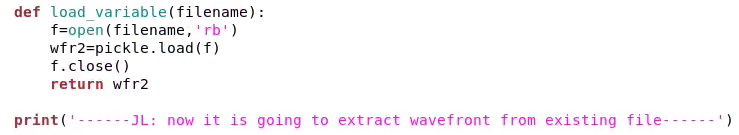


Number of generations and populations are also set here



## Beamline optimisation only

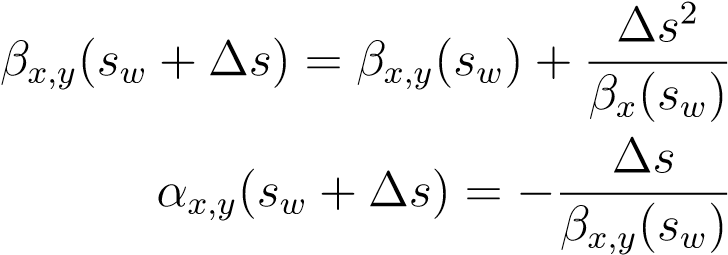
The insertion device for I20 scanning branch is wiggler, it will take more than 1 hour to simulate the emission of radiation, therefore the wavefront at the first slit was saved. For the beamline optimisation, the wavefront will be extracted directly to speed up the simulation. This method can be used to speed up the investigation for wiggler beamlines.



The main calculation function is called ‘CalcIntensity (OptBL, InputFile)’, for beamline only optimisation, this function is defined in **SRW\_intensity\_BLasParam\_Ji.py,** please insert this file to **InGenNoMain.py.**

## BL+twiss optimisations

This optimisation will optimise the βx, βy at the waist(sw) and calculate the Twiss parameters at insertion device position according to the formula



Hence the machine parameters to be optimised are *βxw*(m), *βyw*(m),*ηx* , *ηx’.*

The ‘CalcIntensity (OptBL, InputFile)’ function for BL+twiss optimisations is defined in **SRW\_intensity\_BLasParam\_I20withtwiss.py,** please insert this file to **InGenNoMain.py.**

## Results analyse

For each individual, some attached files will be produced during calculation to help you analyse data**.** All files are identified with the individual number.

**Example(XXX means individual number):**

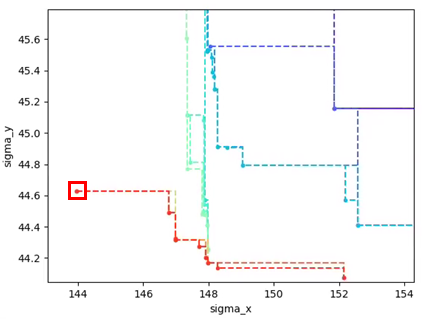
**nsgaTest-XXX.var** is used to check the variables in this individual

**nsgaTest-XXX-submit.sh.eXXX and nsgaTest-XXX-submit.sh.oXXX** are used to record the calculation, if the optimisation has errors, these two files may help you to track the calculation.

**nsgaTest-XXX.sol2 is used to list the solution for that individual.**

The command to plot the fronts is: **python plotor31.py**

An example to show how to find the desired individual number from fronts plot. First, choose one of the solutions in the figure (Fig. 3 top picture, the red rectangle ), then zoom-in, read the beam size sigma\_x or sigma\_y as more digits as possible (the red rectangles in Fig. 3 bottom picture), search this number in all the solutions, you will find the corresponding individual number.

****

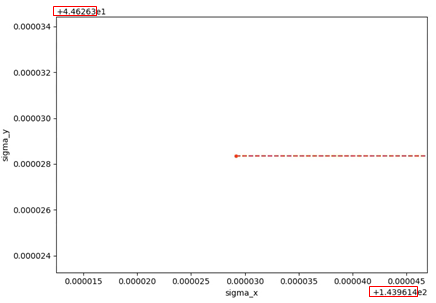
****

Figure 3. An example to show how to find the required individual number from fronts plot.

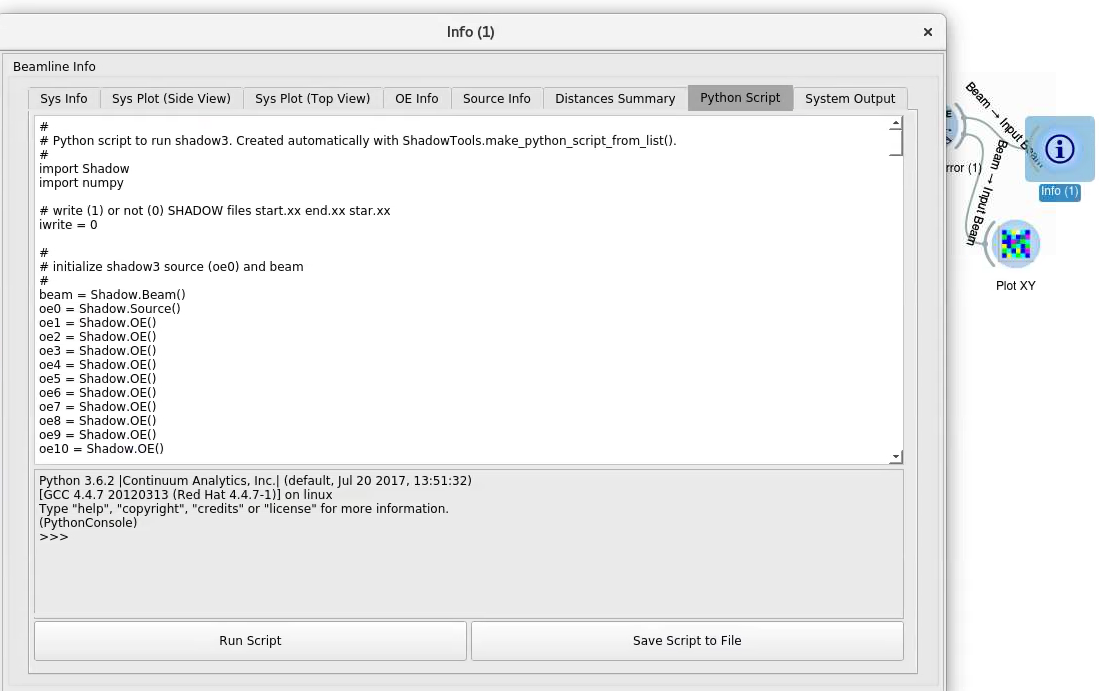
After you have successfully found the individual number, plot the corresponding intensity data:

**python** **ANALYSIS/ana\_intensity.py SRW\_I20/mult\*\*\*\*\*.dat**

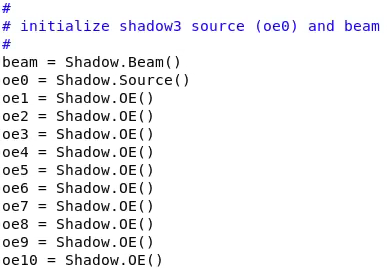
# E2S-SHADOW optimiser

## Create a beamline model

It is recommended to first create a beamline model in OASYS. This will ensure that all parameters are correctly referenced and will provide immediate visual feedback. The Python raw code can be saved from a complete model in OASYS using the info widget.



The raw python code from OASYS starts with the initialisation of all the elements in the beamline



For more information about the definition of the variables can be found:

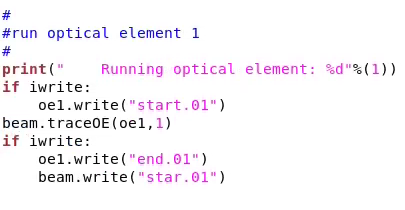
For Source:

<https://raw.githubusercontent.com/srio/shadow3/master/docs/source.nml>

For OE:

<https://raw.githubusercontent.com/srio/shadow3/master/docs/oe.nml>

Once all the parameters for each OE have been defined shadow then creates the source and runs the ray-tracing simulation for each OE specified.



## Build SHADOW3-python

Download shadow3-python from:

<https://github.com/oasys-kit/shadow3>

For building the necessary structure needed to run shadow3 in python, you need to "make python" which essentially creates the needed runtime libraries "**make lib**" and then runs "**python setup.py build**".The latter will compile the necessary C code for the binding (shadow\_bind\_python.c) and define the needed libraries shadow3c (C binding) and shadow3. Using the standard python building strategy, the results are placed in a directory called "build".

**Command to check the library:**

**$ ls build/lib.linux-x86\_64-3.7/Shadow/**

Because initialize shadow3-python via the \_\_init\_\_.py file, it is not possible to run SHADOW stuff from the same build directory. We must cd to another directory, and define two environment variables pointing to the directory contain the shadow3 stuff and the directory containing the required libraries.

**Command to define two environment :**

**export LD\_LIBRARY\_PATH='/users /Shadow\_Optimiser/shadow3/'**

**export PYTHONPATH=’/users/Shadow\_Optimiser/shadow3/build/lib.linux-x86\_64-3.7’**

**Then test if it works (very important! Make sure no errors here):**

**>>> import Shadow**

**>>>**

## **Beamline optimisation**

E2S-SHADOW optimiser is built by linking SHADOW3-python to NSGA II.

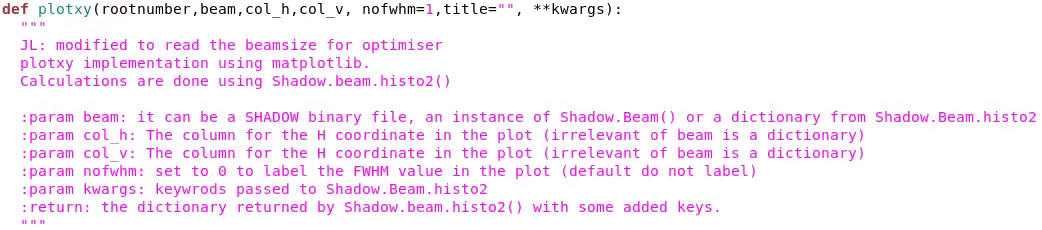
(module load python/3)

Command to run:

**Python nsga.py kur\_par\_test.py**

**kur\_par\_test.py** is used to set the individual values and the range of each variables in the optimisation. Some main function to update the beamline and calculate the rays distribution are **Calcrays,** **BLoptics, update\_BL in BL\_optics\_rays.py.** For each individual, a new **SHADOW-python file** will be created named with the individual number (shadow\_runXXX.py, XXX is the individual number).

**ShadowTools.py** is a modified file to read the beam size and plot histogram information. The main part is the function **plotxy.**



All the solutions will be saved in a folder called **Objects.** The fronts can be plotted by **plotor31.py** (the same analyse method as E2S-SRW optimiser). To check the corresponding rays distribution, command is: **python shadow\_runXXX.py.**

**(PS: E2S-SHADOW optimiser is interfaced with Python3)**

# Partially coherent simulation

The code starts from the Elegant calculation, partially coherent calculation has been implemented to the E2S.py.

**To start with: In the input file, write ‘calc\_type = partially\_coherent’**

**The number of macro-electrons to be calculated is also defined in input file, ‘Nelectr =’.**

The main file to submit the jobs into cluster is in partially coherent/e2s\_SRW:

**./qsub\_Individual\_hamilton.sh**

(the number of cores is defined in this file)

This is used to run **pan\_runbatch\_Individual\_hamilton.sh**, here is the core of this simulation:

**python SRW\_individualelectrons\_BLasParam.py SRW.input**

In SRW\_individualelectrons\_BLasParam.py, it is allowed to select the different beamline and Twiss settings.

‘**update\_accelerator\_optics=**’, ‘1’ is to change the twiss setting, ‘0’ do not change.

Example:

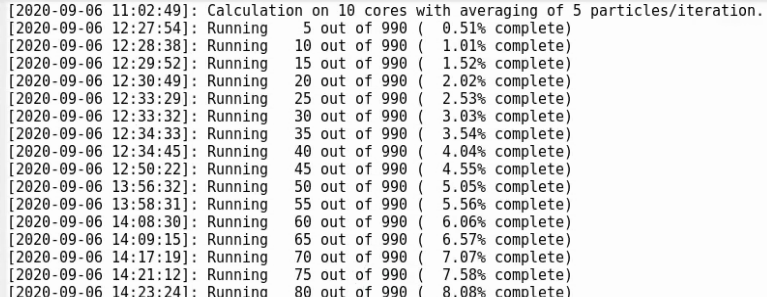
**‘modify\_beamline\_with\_mono=’,** ‘1’ is to change the beamline setting, ‘0’ do not change.

Example:



**‘nMacroElecAvgPerProc =’** defines the number of macro-electrons / wavefront to average on worker processes before sending data to master.

The degree of simulation can be checked in **srw\_mpi.log**



If the simulation has errors, one can check the **e2s\_SRW.e\*\*** file or **e2s\_SRW.o\*\*.**

The data is saved in **SRW\_I13d/individual\*\*\*.**

**To analyse the data, the command used is:**

**python ANALYSIS/ana\_intensity\_new.py SRW\_I13d/individual\*\*\*.**

# Acknowledgements

Thanks to Riccardo Bartolini, Marco Apollonio, Faissal Bakkali Taheri for all the supervising and contribution of the programme, as well as the rest of the Accelerator Physics Group in Diamond who have welcomed and supported through the code development.