Simulation of X-ray beamlines by Phase-Space Analysis

Benjamin MILLER
M1 - Nanotech Masters degree

October 3, 2017

Supervisors

Dr. Claudio FERRERO
ISDD - Data Analysis Unit
ESRF-The European Synchrotron
71, Avenue des Martyrs
CS40220
F-38043 Grenoble Cedex 9
Direct Line: +33 (0)4 76 88 23 70
Fax: +33 (0)4 76 88 25 42
Email: ferrero@esrf.eu

Dr. Manuel SANCHEZ DEL RIO
BP 220
ESRF-The European Synchrotron
71, Avenue des Martyrs
F-38043 Grenoble-Cedex 9
Direct line : +33-476 882513
Fax : +33-476 882542
Email: srio@esrf.eu

Contents

List of Figures

Glossary

2D Gaussian function of the form $f(x, y) = A \exp\left(-\left(a(x - x_o)^2 + 2b(x - x_o)(y - y_o) + c(y - y_o)^2\right)\right)$. It will always appear as an ellipse in a 2D representation. The parameters a,b and c will influence the size of the ellipse and the angle it makes with the x-axis.. 10

acceptance function Numerical aperture. 6

brilliance number of photons per second per mm^2 per rad^2 per $0.0014\frac{\Delta\lambda}{\lambda}$. 5

Gaussian function of the form $f(x) = a \exp(-\frac{(x-b)^2}{2\sigma^2})$ where a is the height of the peak, b the center of the peak and σ the standard deviation. 5

PSA Phase-Space Analysis. 5

1 A presentation of the ESRF

The ESRF, which stands for European Synchrotron Radiation Facility, is the result of an international cooperation between 22 nations to build a place for experimenting with the most intense and brilliant X-ray source worldwide. Thousands of scientists come here every here to experiment in one of the 43 available beamlines linked to the storage ring. This ring, the emblem of the center, it is the outer part of a particle accelerator in which high energy electrons turn and produce the X-ray. The ESRF employs around 600 people. Most are French, German or Italian, but a huge number of cultures are represented there. The experiments realized over there span over a huge range of fields, such as physics and health, but also archeology and geology.

2 Introduction

The purpose of the Grenoble synchrotron is to produce a continuous X-ray beam with the following properties : coherence and high energy.
Along the synchrotron are beamlines, where scientists use this X-ray beam for experiments. These beamlines have a similar global structure : the light comes out of the ring and goes through an optical system before reaching the sample. But to be able to use the results and know how to modify the optical system, one must know the size and the intensity of the X-ray beam that arrives on the sample. This is why the PSA software was designed.

The PSA software, named after the Phase Space Analysis formalization, can theoretically calculate, with the Mathematica program, the size, intensity and divergence of an X-ray beam that comes out of an optical system if the user supplies the size, intensity and divergence of the initial beam and the characteristics of the optical elements.

Now, the ESRF would like to release a more complete version of PSA equipped with a graphical interface, Oasys, but it will not work with Mathematica. Therefore, it is necessary to convert the software from the Wolfram language used in Mathematica to Python. This report's main topic will thus be to highlight the differences in behavior between Python and Wolfram when dealing with symbolic calculations and integrations and how to solve the problems caused by these differences.

First, the Phase Space analysis formalization on which the software relies shall be explained. Then, the following section will detail the way the data is processed in the program and finally how this was translated into Python, what went wrong in the process and how it was fixed.

3 Phase-Space analysis

3.1 Formalization

In Phase-Space Analysis (PSA), every distribution and acceptance function of an optical element is approximated by a Gaussian, this allows us to treat them analytically [3]. Naturally, the closer the real distribution is to a Gaussian, the better the approximation will be. Every system studied by the software will have this structure : an X-ray source and an optical system comprised of optical components such as mirrors or lenses

3.1.1 The source

The source is an X-ray source similar to the ones used in beamlines. The coordinate system used during the whole study is presented in Fig.1. The photon distributions in the horizontal and vertical direction are considered to be mutually independent [1], this leads us to use a set of 2 vectors to describe the characteristics of the beam at a given point :

$$\begin{bmatrix} x \\ x_p \\ \frac{\Delta\lambda}{\lambda} \end{bmatrix} \quad \begin{bmatrix} y \\ y_p \\ \frac{\Delta\lambda}{\lambda} \end{bmatrix} \tag{1}$$

where x is the position on the spacial distribution, x_p the angular divergence of the beam and $\frac{\Delta\lambda}{\lambda}$ is the relative wavelength band.

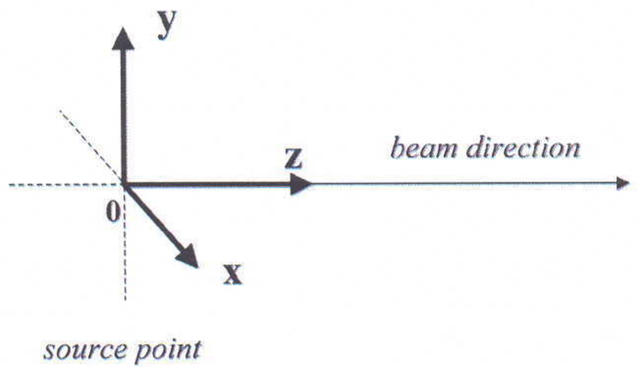


Figure 1: Coordinate system used to describe the beam propagation from the x-ray source through the optics to the sample

The source is characterized in terms of its brilliance by the following expression [3,5,6,8]:

$$I(x, x_p, y, y_p, \lambda) = I(0, 0, 0, 0, \lambda) I_x(x, x_p) I_y(y, y_p) I_\lambda(\frac{\Delta\lambda}{\lambda}) \tag{2}$$

where $I(0, 0, 0, 0, \lambda)$ is the brilliance at the center of the beam and $I(x, xp, y, yp, \lambda)$ is the brilliance of the extended source. The term extended source will refer to the source at any given place, where it has or has not traveled through the optics. Every function in (2) is approximated by a multiple Gaussian distribution :

$$I_x(x, x_p) = \exp(-\frac{\frac{x^2}{\sigma_x^2} + \frac{(x_p)^2}{\sigma_{x_p}^2}}{2}) \tag{3}$$

$$I_y(y, y_p) = \exp(-\frac{\frac{y^2}{\sigma_y^2} + \frac{(y_p - \Gamma_y)^2}{\sigma_{y_p}^2}}{2}) \tag{4}$$

$$I_\lambda(\frac{\Delta\lambda}{\lambda}) = \exp(-\frac{(\frac{\Delta\lambda}{\lambda})^2}{2\sigma_{s\lambda}^2}) \tag{5}$$

σ_x and σ_y are the standard deviations of the source's brilliance with respect to space, and σ_{x_p} and σ_{y_p} the standard deviations with respect to phase, so in a way they represent the size of the beam and the divergences along the two directions of our axis on Fig.1 perpendicular to the beam, and $\frac{\Delta\lambda}{\lambda}$ is the relative wavelength band. The emittance of the electron beam Γ will be equal to 0 unless the electron beam went through a bending magnet [2].

3.1.2 Optical Components

The effect of each optical element on the beam can be reduced to 2 transformation matrices and an acceptance function. The transformation matrices will take the transmitted ray from the position-angle-wavelength space and transform it into the incident ray. We need two transformation matrices, that both have a 3×3 dimension, per element since we use two sets of vectors to describe the source. The acceptance function is similar to the numerical aperture, it characterizes the range in which all 3 parameters can be accepted by the element. Here, every acceptance function will be a Gaussian. There are two functions, one that changes the brilliance according to the angular divergence and one according to the spacial parameters.

3.1.3 Coordinate transformation

We shall assume that we know the intensity distribution $f(u_1, u_{p1}, \frac{\Delta\lambda}{\lambda})$ at a certain position $z = z_1$. Both u_1 and u_{p1} are given in the coordinate system at $z = z_1$. Either propagation in the z-direction or going through an element will affect the coordinates and their transformation is usually given as :

$$\begin{bmatrix} x_1 \\ x_{p1} \\ \frac{\Delta\lambda}{\lambda} \end{bmatrix} = A \cdot \begin{bmatrix} x_2 \\ x_{p2} \\ \frac{\Delta\lambda}{\lambda} \end{bmatrix} \tag{6}$$

where A is a transformation matrix.

3.1.4 Propagation

Simply traveling thought air affects the state of the beam. One of the 3 considered parameters is divergence, which translates here as the angle in which the electrons will scatter at a certain distance. when the beam comes out of the source, the divergence will be low, but the further it travels, the larger the divergence angle will be because the probability we find electrons further from the center will be higher. It is like a cone, at the beginning the electrons are all close to the center but after 100 meters there is a chance that they all scattered. This effect is taken into account with a flight matrix [7]:

$$\begin{bmatrix} 1 & -L & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

where L is the length of propagation

3.2 Propagation from the source to the sample

When we apply the PSA method, what we want is to find, after propagation through the optical system, the source's brilliance at the sample, similar to the expression given at (2). We get it by multiplying the acceptance functions of each optical element to the source's brilliance.
To make this more clear, we shall rely on an example. Let us consider the following system comprised of the source, a lens and the sample :

There will be 3 transformations :

- between (1) and (2) there is a propagation of length L1
- between (2) and (3) the brilliance changes due to the acceptance of the lens
- between (3) and (4) there is a propagation of length L2

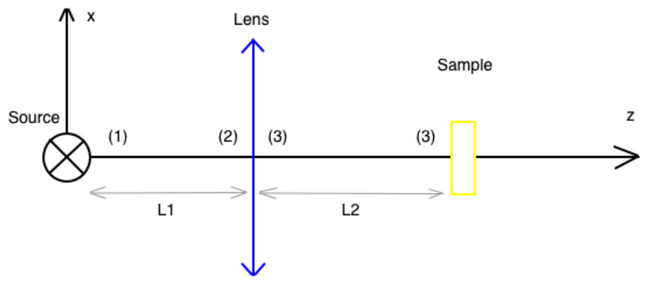


Figure 2: Considered system

Now here is the tricky part. Theory gives us the following transformation :

$$\begin{bmatrix} x_4 \\ x_{p4} \\ \frac{\Delta\lambda}{\lambda} \end{bmatrix} = A_3 \cdot A_2 \cdot A_1 \cdot \begin{bmatrix} x_1 \\ x_{p1} \\ \frac{\Delta\lambda}{\lambda} \end{bmatrix} \tag{8}$$

with A_1 and A_3 the flight matrices and A_2 the transformation matrix of the lens. The acceptance functions take into parameter the state of the source just before it goes through the lens, which is described by this vector :

$$\begin{bmatrix} x_2 \\ x_{p2} \\ \frac{\Delta\lambda}{\lambda} \end{bmatrix} = A_1 \cdot \begin{bmatrix} x_1 \\ x_{p1} \\ \frac{\Delta\lambda}{\lambda} \end{bmatrix} \tag{9}$$

Now, in the final result, variables in relation to the initial source must not appear in any way since they cannot be used. It is necessary to express the brilliance with the variables related to the final source since they will be integrated in order to obtain the distribution over phase and space. Hence :

$$\begin{bmatrix} x_2 \\ x_{p2} \\ \frac{\Delta\lambda}{\lambda} \end{bmatrix} = A_2^{-1} \cdot A_3^{-1} \cdot \begin{bmatrix} x_4 \\ x_{p4} \\ \frac{\Delta\lambda}{\lambda} \end{bmatrix} \tag{10}$$

So the transformations needed in this present case are the inverse of those usually given [6]. As of now, we shall use the transformation matrices T_i where $T_i = A_i^{-1}$. For this example, this gives us the following matrices :

$$T_1 = \begin{bmatrix} 1 & -L_1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 1/F & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad T_3 = \begin{bmatrix} 1 & -L_2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

3.3 A list of the considered optical components

This list is mainly to detail the behavior and parameters of each component, an extensive list of all matrices and acceptances is in the first annexe. Each component can be placed on the path of the ray in the x or the y axis.

3.3.1 Lenses

Here, parabolic X-ray lenses are used. It is shaped as a glass box with parabolic consecutive holes. Their behavior is similar to a regular lens, as it is characterized by its focal distance and is generally used to lower the divergence of the beam. The size and number of holes and the distance between the holes will greatly influence the acceptance.

3.3.2 Slits

They are either rectangular or circular. They are made of a metal, to reflect the unwanted x-ray. Their purpose is to lower the size of the beam, so the standard deviation of the Gaussian distribution of the brilliance relative to space.

3.3.3 Monochromators

A monochromator is a crystal used to select a narrow band of wavelengths, which will thus decrease the value of the relative wavelength : $\frac{\Delta\lambda}{\lambda}$.Their operating principle is related to Bragg diffraction, the rays impinge upon the crystal and are reflected or transmitted by the planes inside the crystal. Interferences then occur between the reflected rays coming from different crystalline planes. This means their Bragg angle is crucial in their filtering action. Silicon is the most commonly used.

3.3.4 Mirrors

They can be flat, curved or toroidal. They have no acceptance functions since their size is not taken into account. Like the lenses, their focus distance is their main characteristic.

3.3.5 Multilayers

A multilayer is a pile of successive layers of two elements with interference mirror properties. One layer usually transmits and one reflects rays. They rely on Bragg diffraction like the monochromator, the difference is mainly in their structure. An example of structure is Mo/Si.

4 Data processing through the PSA software

Let us now see how this process is implemented by the software and how to get, from the source parameters and the optical elements, the brilliance distribution of the source.

4.1 Parameters

The source is characterized by equation (2), which is a product of three Gaussians and a constant, so only the standard deviations of these Gaussians and the constant need to be supplied.
As for the optics, the parameters needed are the ones that will intervene in the expressions of the matrices and the acceptance functions. They are all detailed in Appendix 1.

4.2 Propagation

Let us re-use the example from 2.2.
To begin with, a list of all useful matrices and the initial source's vectors are created :

- $ListM_x = [T_{1x}, T_{2x}, T_{3x}]$
- $ListM_y = [T_{1y}, T_{2y}, T_{3y}]$
- $M_x = \begin{bmatrix} x_4 \\ x_{p4} \\ \frac{\Delta\lambda}{\lambda} \end{bmatrix}$
- $M_y = \begin{bmatrix} y_4 \\ y_{p4} \\ \frac{\Delta\lambda}{\lambda} \end{bmatrix}$

Then, the brilliance of the source is calculated using the final source's parameters :

$$\begin{bmatrix} x_1 \\ x_{p1} \\ \frac{\Delta\lambda}{\lambda} \end{bmatrix} = T_{1x}T_{2x}T_{3x}M_x \tag{12}$$

$$I_x(x_1, x_{p1}) = I_x(f(x_4, x_{p4})) \tag{13}$$

The same is done for y and y_p.
Now, let us calculate the acceptance of the lens :

$$A(x_2, \frac{\Delta\lambda}{\lambda}) = \exp(-\frac{x_2^2 + Rd}{2\Sigma(\lambda)^2}) \qquad where \qquad \begin{bmatrix} x_2 \\ x_{p2} \\ \frac{\Delta\lambda}{\lambda} \end{bmatrix} = T_{2x}T_{3x}M_x \tag{14}$$

The brilliance of the source reaching the sample is thus :

$$I_4(x_4, x_{4p}, \frac{\Delta\lambda}{\lambda}) = I_1(x_4, x_{p4}, \frac{\Delta\lambda}{\lambda}) * A(f(x_4), \frac{\Delta\lambda}{\lambda}) \tag{15}$$

4.3 Integration

The flux Φ at any arbitrary position along the beamline is calculated by integrating with respect to all variables :

$$\Phi = \int \iiiint_{\infty} I(x, x_p, y, y_p, \frac{\Delta\lambda}{\lambda})\, dx\, dx_p\, dy\, dy_p\, d\lambda \tag{16}$$

What is also interesting to know is the standard deviation of each Gaussian distribution regarding the 5 considered variables : $\sigma_x, \sigma_{xp}, \sigma_y, \sigma_{yp}, \sigma_\lambda$.

The procedure for getting each of these deviations is quite similar, so only the case of σ_x will be studied as an example. At this stage of the analysis, we have a brilliance function in the form of a product of negative exponentials, so integrable. A useful observation is that we do not have exactly a product of several Gaussians, so we cannot get the integral that easily. To get σ_x, we first place ourselves in the space coordinates, x and y. This means the brilliance is first integrated

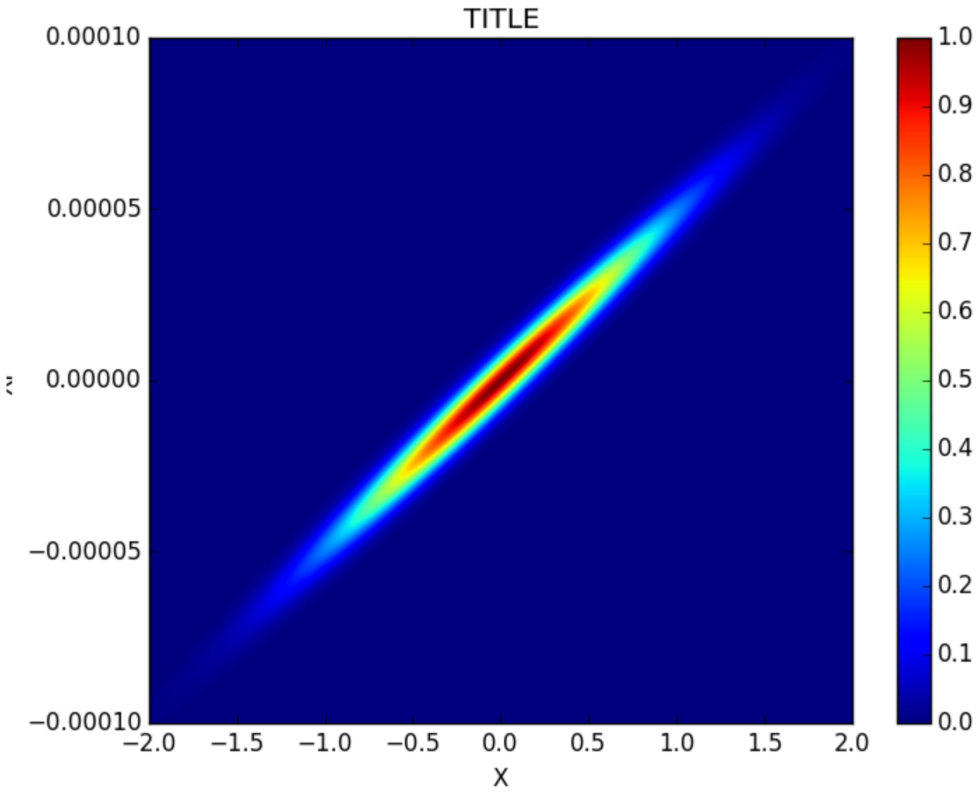


Figure 3: 2D Gaussian

with respect to x_p, y_p and $\frac{\Delta\lambda}{\lambda}$.By this way, a 2D Gaussian is obtained, showing the distribution of the brilliance in function of x and y like in Fig 3.

Hereby, σ_x will be the value of the standard deviation when $y = 0$, thus getting this function :

$$f(x, y = 0) = \iiint_{\infty} I(x, x_p, y, y_p, \frac{\Delta\lambda}{\lambda})\, dx_p\, dy_p\, d\lambda \tag{17}$$

This function f is a Gaussian, so all is left is to extract the standard deviation by evaluating the function in two points x and solving the equation. On Figure 4 is the Gaussian at the origin of Figure 3 :

This ends the analysis done by the PSA software. Now, lets see which are the modifications to bring so this algorithm works in Python.

5 Highlighting differences between Python and Mathematica

The main issue to deal with when converting from Wolfram to Python is that Wolfram works with an infinite precision and with symbolic calculation.

There is indeed a symbolic module in Python, but it does not work for functions that do not have a primitive, which is our case since the functions are similar to the Gauss error function. By way of mean, all integrations are numerical.

5.1 Length of operation

Converting from symbolic to numerical operations brings one complication, the integrations of a function over its variables cannot be done one after the other but all at the same time otherwise the software does not know what to do of the symbols during the process. Now, in several cases, this leads us to compute quadruple integrals. Tests led to conclude that adding a variable during

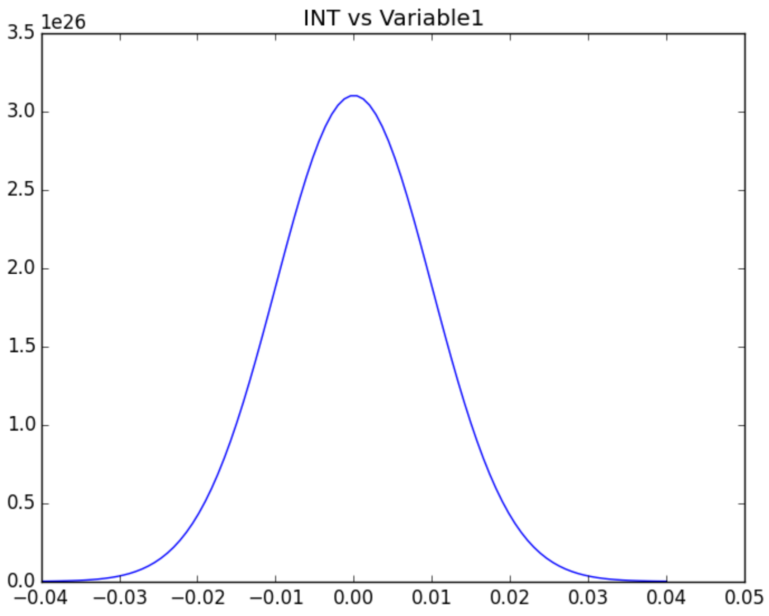


Figure 4: Distribution of $I_y(y, 0)$

a simultaneous integration will multiply the process' time by approximately 300, so only triple integrals were tolerated in the code. This means the integrals had to be calculated separately.

Now, as an example, here is the solution found for calculating the flux (16) :

The brilliance before any transformation can be written in the form of equation (2). After going through an optical system, it will have this form :

$$I_f(x, x_p, y, y_p, \frac{\Delta\lambda}{\lambda}) = I_f(0, 0, 0, 0, \lambda) I_{xf}(x, x_p, \frac{\Delta\lambda}{\lambda}) I_{yf}(y, yp, \frac{\Delta\lambda}{\lambda}) I_\lambda(\frac{\Delta\lambda}{\lambda}) \tag{18}$$

The only way to integrate this whole expression with respect to all variables is to separate the integrals, so we have :

$$\Phi = \int [I_f(0, 0, 0, 0, \lambda) \iint I_{xf}(x, x_p, \frac{\Delta\lambda}{\lambda})\, dx\, dx_p \iint I_{yf}(y, yp, \frac{\Delta\lambda}{\lambda})\, dy\, dy_p]\, d\lambda \tag{19}$$

So we test to see if either I_{xf} or I_{yf} depends on $\frac{\Delta\lambda}{\lambda}$. An observation of the transformation matrices and the acceptances leads us to conclude that this is the case only if there is a monochromator on the x or y path. So unless there is a monochromator on both the x and y path (a simulation that almost never occures), it will always be possible to separate the expression of the flux into a product of a double and a triple integral.

By separating integrals, the flux of every system that does not include a monochromator on both paths can be calculated using non-symbolic integration, with a reasonable duration. Sadly, numeric triple integrals still take a lot of time to compute, so the calculation takes around ten minutes. Furthermore, to enhance the precision, triple integrals are calculated by using a 3D array of values of the function taken with a step, summing every value and then multiplying them by the step.

5.2 Dealing with too low values

Getting the numerical value of the standard deviation implies evaluating the expression of the brilliance at a non-null value. The issue here is that Python will sometimes evaluate the integral at zero. Solving this only requires to lower the value used for the evaluation by incrementally dividing it by two until a non-null result is reached.

5.3 A new integration routine

For reasons such as extensive computation time and precision, it is not possible to keep infinite boundaries for integrations, and too high or too low boundaries will lead to false results. By way of mean, it is essential to use a reliable integration routine for 2D Gaussians.

5.3.1 First step : Boundaries at the origin

At any moment of the propagation, there is never the introduction of any other function than Gaussian error functions, so the 2D Gaussians studied here are centered around the origin.

Lets see the easiest way to get integration boundaries and one case where it works.

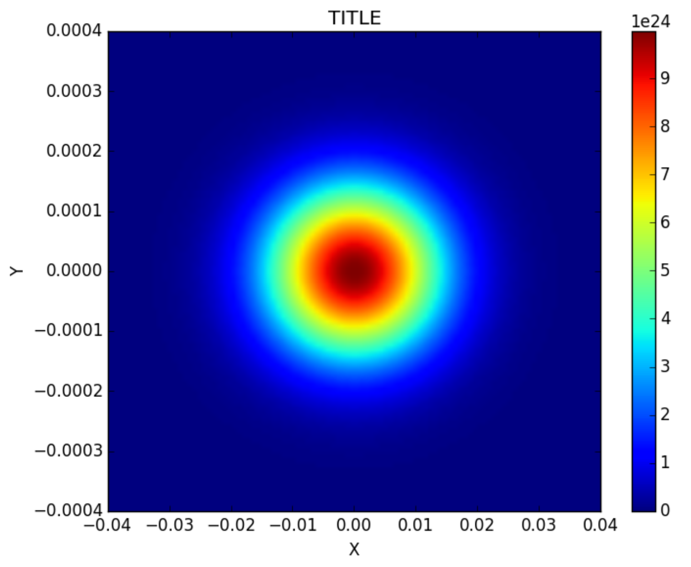


Figure 5: Brilliance as a function of y and y_p

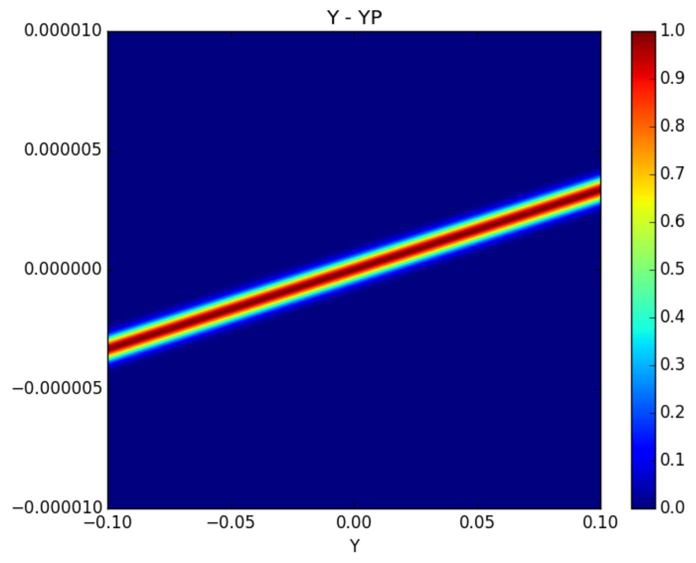


Figure 6: A consequence of too low boundaries for a thin Gaussian

One variable, x, is set to zero, while the other one, y, will vary along its axis. Lets call our Gaussian function f. The peak will thus be f(0,0). y is initially set at a very low value, lower than what can be measured. Here, $\epsilon = 10^{-14}$ was chosen. Then, as long as $f(0,\epsilon) < 10^-15 \cdot f(0,0)$, ϵ is multiplied by 2 incrementally. 10^{-15} was chosen because during tests, precision was lacking with higher values.

For 2D Gaussians with a 0 degree angle, these boundaries are sufficient since they trace a square around the whole figure. Fig 5 is an illustration of this.

Now, when encountering a thin 2D Gaussian with a thin width which makes a non-null angle with the x-axis, this routine does not work anymore, only a very small portion of the figure is contained inside the square, as seen in Figure 6 :

5.3.2 Second step : Enlarging the boundaries

The observation that comes to mind is that the boundaries need to be enlarged. The center of the Gaussian is enclosed in a square. We know by looking at the graphic that the process is not over because the square cuts the figure midway. To translate this in mathematical terms, the process is not over until the integral along each side of the square is lower than 10−15 of the center value. By applying this method, the thin Gaussian finally appears in Figure 7.

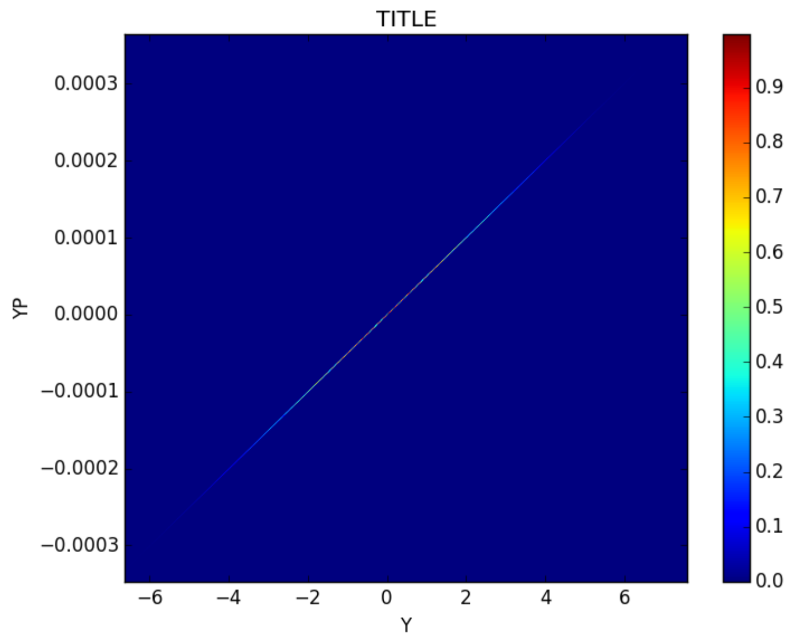


Figure 7: Thin Gaussian

5.3.3 Non-centered Gaussians

The process in 3.3 requires the calculation of a non-centered Gaussian, because the evaluation mentioned at the end of the paragraph is done before to avoid the use of symbols during the integration. Indeed, as shown in Fig.8, when one of the parameters is set to a non-null value, the peak is not centered in (0,0).

This issue is solved by multiplying the initial integration boundaries by 10 before starting to enlarge the boundaries. That way, we ensure that at least a part of the Gaussian is inside the boundaries. No values are given here because this may require a bit of tuning between the tenfold value and the evaluation chosen in 3.3. This is why the evaluation must always be chosen low enough so that once the program correcting lower values is done, we have two values that will allow a correct enlargement of the boundaries.

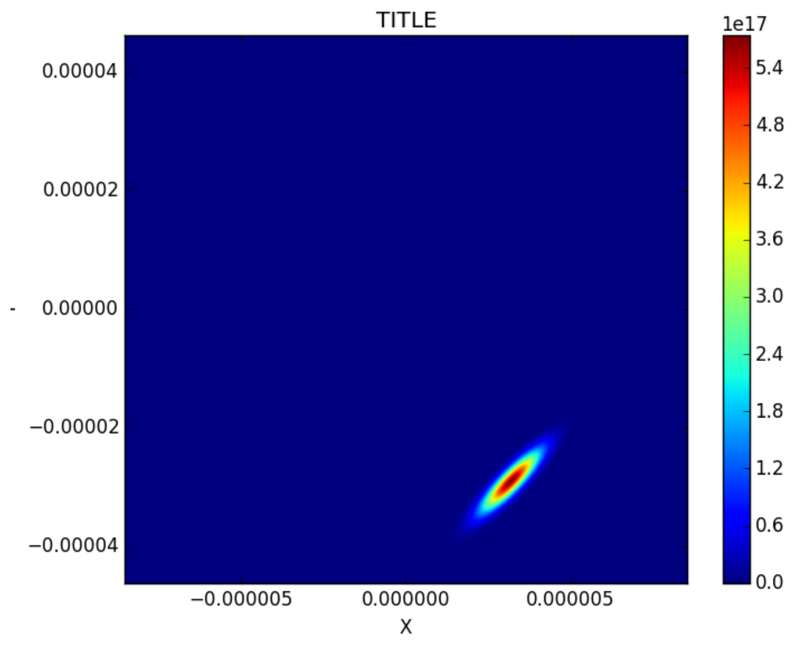


Figure 8: Non-centered Gaussian

6 Conclusion

The Python-Psa software is operational and returns all five standard deviations, the flux and graphs depicting the 2D Gaussians and the distributions at the origin. It is fully functional for slits, pinholes, horizontal parabolic lenses, simple propagation, mirrors, multilayers, but results still are not correct for mosaic and bent monochromators, 2D parabolic lenses and plane horizontal monochromators. As for vertical parabolic lenses and pinholes, there is no data to assess if the results are correct. Even if running the whole program takes around ten minutes, this can hardly be repaired since the calculation of the triple integral takes almost the whole time. To improve the process, it would be necessary to find a new integration routine or other integration functions, like the trapezium method. Anyway, once it fully works, the code will be linked to a graphical interface to better visualize and input the optical system, and it will be ready to integrate Oasys.

7 References

[1] C. Ferrero, D.-M. Smiglies, C. Riekel, G. Gatta and P. Daly, "*Extending the possibilities in phase space analysis of synchrotron radiation x-ray optics*", Applied optics/Vol.47, No.22/ 1 August 2008.
[2] G.K. Green, "Spectra and optics of synchrotron radiation", BNL Rep. 50522 (Brookhaven National Laboratory, New York, USA, 1976).
[3] J.S. Pedersen and C. Riekel, "*Resolution Function and Flux at the Sample for Small-Angle X-ray Scattering Calculated in Position-Angle-Wavelength Space*, Journal of Applied Crystallography 24, 893-909 (1991)
[4]"What is the ESRF" esrf.eu. ESRF, n.d. 29 Sept 2017
[5]G. K. Green, "Spectra and optics of synchrotron radiation", BNL REP: 50522 (Brokkhaven National Laboratory, New York USA, 1976).
[6]T.Matsuhita and U.Kaminaga, in *Handbook on Synchrotron Radiation*, E.E.Koch, ed. (North Holland, 1983°, Vol.1, pp.261-314.
[7]D.-M. Smilgies, "Compact matrix formalism for phase space analysis of complex optical systems", Appl. Opt. 47, E106-E115 (2008).
[8]S.Krinsky, M.G.Perlman, and R.E. Watson, in *Handbook on Synchrotron Radiation*, E.E. Koch, ed.(North Holland, 1983), Vol. 1 pp. 67-171.

Appendix

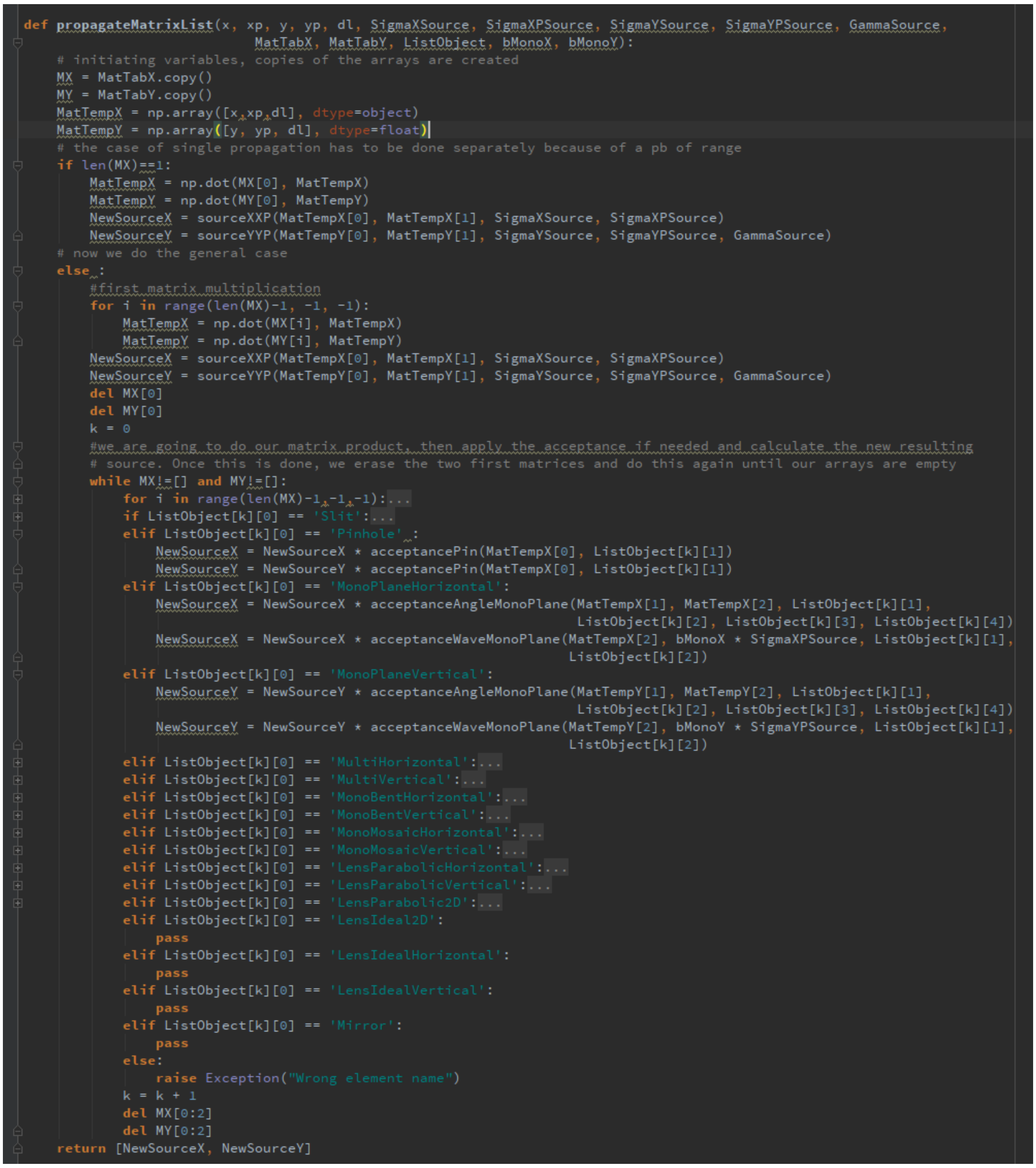
Appendix 1 : List of optical components and their matrices and acceptance functions

Matrices	Acceptance functions
Flight path of length L $\begin{bmatrix} 1 & -L & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	
Perfect flat crystal monochromator $\begin{bmatrix} b & 0 & 0 \\ 0 & \frac{1}{b} & (1-1/b)\tan(\theta_B) \\ 0 & 0 & 1 \end{bmatrix}$	$A(\lambda) = \frac{R_{Mono}R_{Int}}{W_d}\sqrt{\frac{6}{\pi}} * \exp(-(y_p - \frac{\Delta\lambda}{\lambda}\frac{12\tan(\theta_B))^2}{(2W_d^2)})$ $A(y_p, \frac{\Delta\lambda}{\lambda}) = \sqrt{\frac{6}{\pi}} * \exp(-\frac{(\frac{\Delta\lambda}{\lambda})^2\tan(\theta_B)^2}{2(\sigma_{yp}^2 + \frac{W_d}{12}))})$
Perfect curved crystal monochromator $\begin{bmatrix} b & 0 & 0 \\ \frac{1}{F_c} & \frac{1}{b} & (1-1/b)\tan(\theta_B) \\ 0 & 0 & 1 \end{bmatrix}$	$A(\lambda) = \frac{R_{Mono}R_{Int}}{W_d}\sqrt{(\frac{6}{\pi})} * \exp(-(y_p - \frac{y}{r\sin(\theta_B+\alpha)} - \frac{\Delta\lambda}{\lambda}\frac{12\tan(\theta_B))^2}{(2W_d^2)})$ $A(y, y_p, \frac{\Delta\lambda}{\lambda}) = \sqrt{(\frac{6}{\pi})} * \exp(-(\frac{\Delta\lambda}{\lambda})^2\frac{(\frac{\tan(\theta_B)}{2})^2}{(\frac{SigmaSource}{DistanceFromSource})^2 + \frac{W_d^2}{12}})$
Mosaic monochromator $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 2\tan(\theta_B) \\ 0 & 0 & 1 \end{bmatrix}$	$A(y_p, \lambda) = \frac{R_{Int}}{\eta}\cdot\sqrt{\frac{6}{\pi}} * \exp(-\frac{(y_p - \frac{\Delta\lambda}{\lambda}\tan(\theta_B))^2}{2\eta^2})$ $A(y, y_p, \frac{\Delta\lambda}{\lambda}) = \sqrt{\frac{6}{\pi}} * \exp(-\frac{(\frac{\Delta\lambda}{\lambda})^2\tan(\theta_B)^2}{2(\sigma_{yp}^2+\eta^2)})$
Curved and toroidal mirror $\exp(-\frac{4\pi\sin(\Theta_i)\sigma_T}{\lambda})\begin{bmatrix} 1 & 0 & 0 \\ \frac{1+F_m\Delta}{F_m} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	
Curved multilayer $\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{F_c} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$A(\lambda) = \sqrt{\frac{6}{\pi}} \cdot \exp(-\frac{(\frac{\Delta\lambda}{\lambda})^2}{2\cdot((\frac{SigmaSource}{DistanceFromSource})^2 + \frac{Ws^2}{8\cdot\ln(2)})\cdot\tan(\theta_{ML})^2})$ $A(y, y_p, \frac{\Delta\lambda}{\lambda}) = \frac{8}{3}Rml\cdot\sqrt{\frac{\ln 2}{\pi}}\cdot\exp(-\frac{8\ln(2)(-y_p - \frac{y}{R_c\cdot\sin(\theta_{ML})} - \frac{\Delta\lambda}{\lambda}\cdot\tan(\theta_{ML}))^2}{2W_s^2})$
Slit	$A(y) = \sqrt{\frac{\frac{6}{\pi}}{\frac{6\cdot\ln(2)}{\pi}}} * \exp(-\frac{12}{2}(\frac{y}{Aperture})^2)$
Circular pinhole of diameter D_{ph}	$A(y) = \sqrt{(\frac{8}{\pi})} * \exp(-\frac{16}{2}(\frac{y}{D_{ph}})^2)$
Parabolic lens $\begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{F} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ $\frac{1}{F} = \frac{2N\delta}{r}$	$A(y) = \exp(-\frac{y^2+Rd}{2\Sigma^2})$

The previous parameters being :
$b = \frac{\sin(\theta_B+\alpha)}{\sin(\theta_B-\alpha)}$
θ_B = Bragg angle
α = angle between the normal to the crystal surface and the normal to the diffracting planes. Its sign is chosen as positive when the x-ray incidence angle with respect to the crystal surface is greater than the respective than the respective exit angle. Very often, this value will be set at zero.
W_D = Darwin width of the crystal
$\frac{R_c}{2}$ = radius of the Rowland circle for a Johansson type curved monochromator
$p = R_c \sin(\theta_B + \alpha)$
$q = R_c \sin(\theta_B - \alpha)$
$\frac{1}{F_c} = \frac{1}{p} + \frac{1}{q}$
p_m = source to mirror distance
q_m = mirror to focus distance
$\frac{1}{F_m} = \frac{1}{p_m} + \frac{1}{q_m}$
θ_{ML} = Multilayer Bragg angle
R_{ML} = peak reflectivity of the multilayer
W_{ML} = multilayer reflectivity angular width for monochromatic rays
W_y = slit width
Σ = standard deviation
δ = thickness of the lens
d = distance between the holes
R = radius of the holes
r = curvature of the lens
N = refractive index
Θ_i = inclination angle
R_{mono} = maximum of the acceptance of the monochromator
R_{int} = value of the integral of the acceptance

Appendix 2 : Important functions from the program’s code

Propagation function

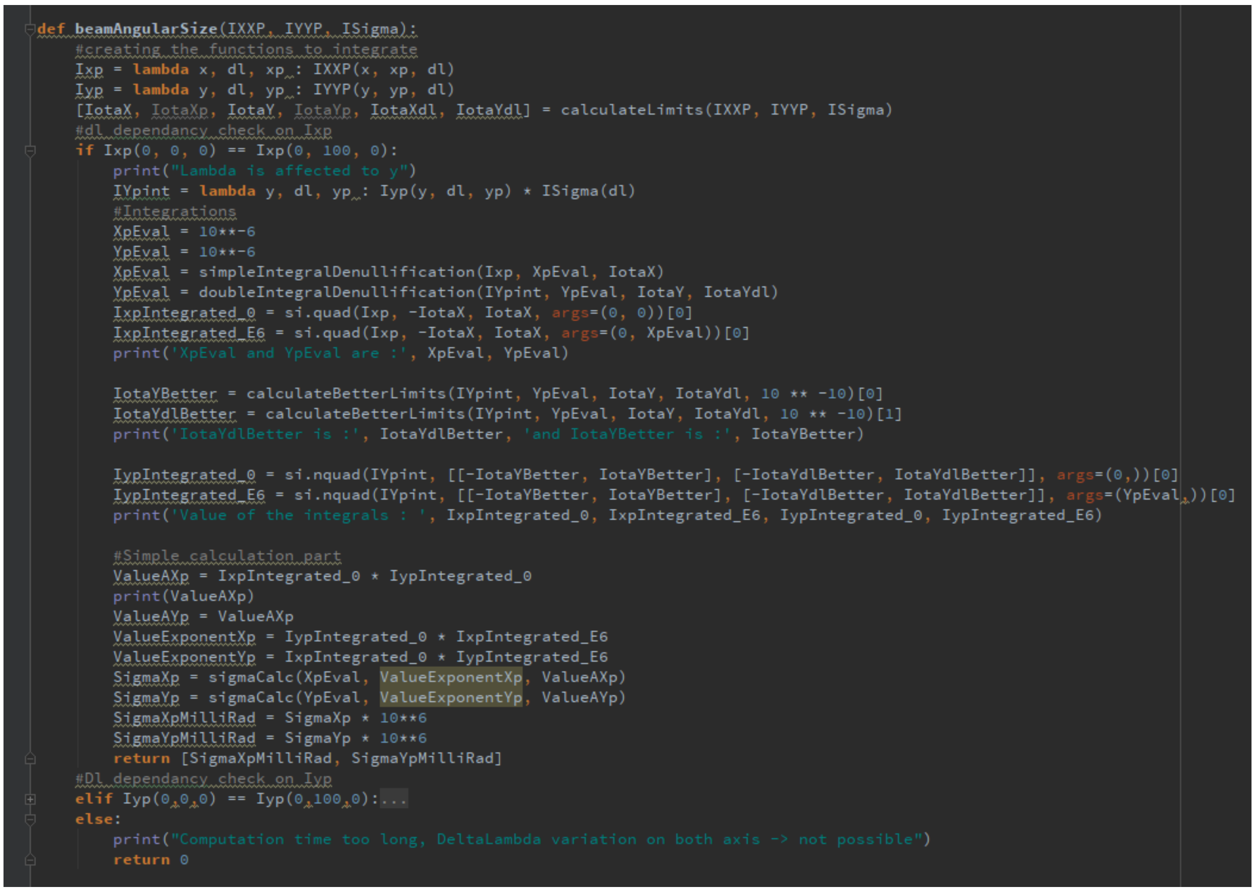


```python
def propagateMatrixList(x, xp, y, yp, dl, SigmaXSource, SigmaXPSource, SigmaYSource, SigmaYPSource, GammaSource,
                        MatTabX, MatTabY, ListObject, bMonoX, bMonoY):
    # initiating variables, copies of the arrays are created
    MX = MatTabX.copy()
    MY = MatTabY.copy()
    MatTempX = np.array([x,xp,dl], dtype=object)
    MatTempY = np.array([y, yp, dl], dtype=float)
    # the case of single propagation has to be done separately because of a pb of range
    if len(MX)==1:
        MatTempX = np.dot(MX[0], MatTempX)
        MatTempY = np.dot(MY[0], MatTempY)
        NewSourceX = sourceXXP(MatTempX[0], MatTempX[1], SigmaXSource, SigmaXPSource)
        NewSourceY = sourceYYP(MatTempY[0], MatTempY[1], SigmaYSource, SigmaYPSource, GammaSource)
    # now we do the general case
    else :
        #first matrix multiplication
        for i in range(len(MX)-1, -1, -1):
            MatTempX = np.dot(MX[i], MatTempX)
            MatTempY = np.dot(MY[i], MatTempY)
        NewSourceX = sourceXXP(MatTempX[0], MatTempX[1], SigmaXSource, SigmaXPSource)
        NewSourceY = sourceYYP(MatTempY[0], MatTempY[1], SigmaYSource, SigmaYPSource, GammaSource)
        del MX[0]
        del MY[0]
        k = 0
        #we are going to do our matrix product, then apply the acceptance if needed and calculate the new resulting
        # source. Once this is done, we erase the two first matrices and do this again until our arrays are empty
        while MX!=[] and MY!=[]:
            for i in range(len(MX)-1,-1,-1):...
            if ListObject[k][0] == 'Slit':...
            elif ListObject[k][0] == 'Pinhole' :
                NewSourceX = NewSourceX * acceptancePin(MatTempX[0], ListObject[k][1])
                NewSourceY = NewSourceY * acceptancePin(MatTempX[0], ListObject[k][1])
            elif ListObject[k][0] == 'MonoPlaneHorizontal':
                NewSourceX = NewSourceX * acceptanceAngleMonoPlane(MatTempX[1], MatTempX[2], ListObject[k][1],
                                                                   ListObject[k][2], ListObject[k][3], ListObject[k][4])
                NewSourceX = NewSourceX * acceptanceWaveMonoPlane(MatTempX[2], bMonoX * SigmaXPSource, ListObject[k][1],
                                                                  ListObject[k][2])
            elif ListObject[k][0] == 'MonoPlaneVertical':
                NewSourceY = NewSourceY * acceptanceAngleMonoPlane(MatTempY[1], MatTempY[2], ListObject[k][1],
                                                                   ListObject[k][2], ListObject[k][3], ListObject[k][4])
                NewSourceY = NewSourceY * acceptanceWaveMonoPlane(MatTempY[2], bMonoY * SigmaYPSource, ListObject[k][1],
                                                                  ListObject[k][2])
            elif ListObject[k][0] == 'MultiHorizontal':...
            elif ListObject[k][0] == 'MultiVertical':...
            elif ListObject[k][0] == 'MonoBentHorizontal':...
            elif ListObject[k][0] == 'MonoBentVertical':...
            elif ListObject[k][0] == 'MonoMosaicHorizontal':...
            elif ListObject[k][0] == 'MonoMosaicVertical':...
            elif ListObject[k][0] == 'LensParabolicHorizontal':...
            elif ListObject[k][0] == 'LensParabolicVertical':...
            elif ListObject[k][0] == 'LensParabolic2D':...
            elif ListObject[k][0] == 'LensIdeal2D':
                pass
            elif ListObject[k][0] == 'LensIdealHorizontal':
                pass
            elif ListObject[k][0] == 'LensIdealVertical':
                pass
            elif ListObject[k][0] == 'Mirror':
                pass
            else:
                raise Exception("Wrong element name")
            k = k + 1
            del MX[0:2]
            del MY[0:2]
    return [NewSourceX, NewSourceY]
```

Angular integration function

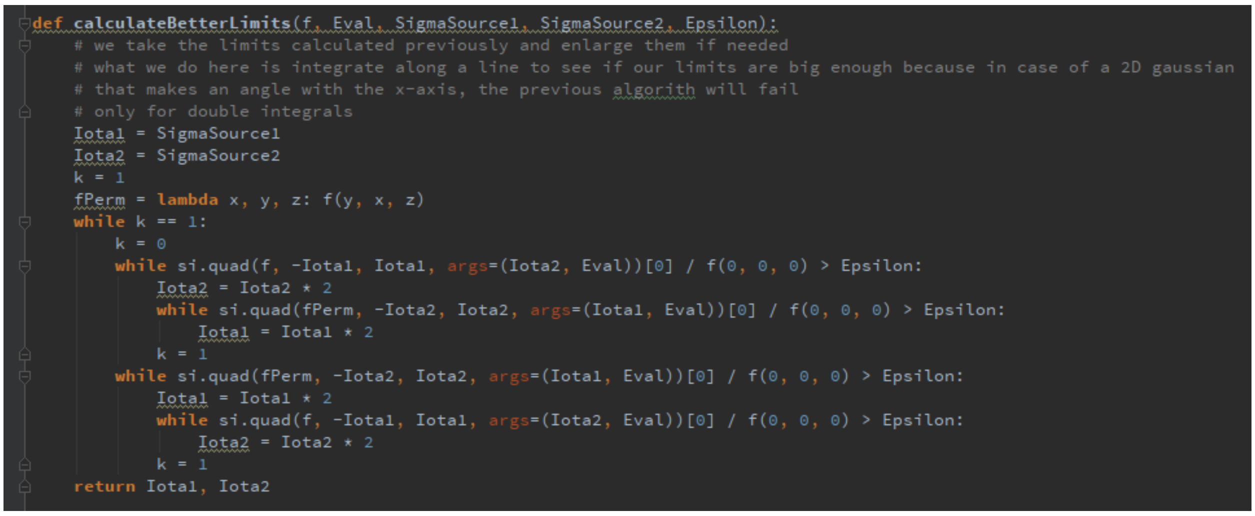
There are three integration function, one for the distribution of the brilliance according to the space, to the phase and to the relative wavelength, and their structure is very similar.

Here is the integration function relative to the phase :



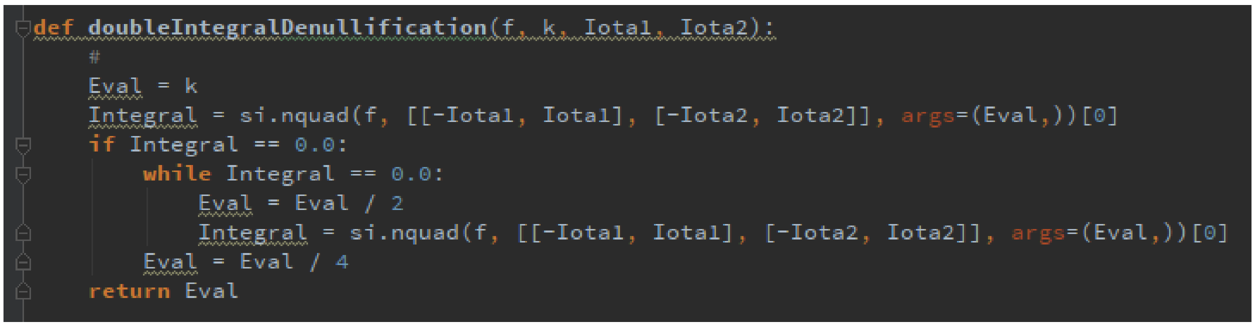
```python
def beamAngularSize(IXXP, IYYP, ISigma):
    #creating the functions to integrate
    Ixp = lambda x, dl, xp : IXXP(x, xp, dl)
    Iyp = lambda y, dl, yp : IYYP(y, yp, dl)
    [IotaX, IotaXp, IotaY, IotaYp, IotaXdl, IotaYdl] = calculateLimits(IXXP, IYYP, ISigma)
    #dl dependancy check on Ixp
    if Ixp(0, 0, 0) == Ixp(0, 100, 0):
        print("Lambda is affected to y")
        IYpint = lambda y, dl, yp : Iyp(y, dl, yp) * ISigma(dl)
        #Integrations
        XpEval = 10**-6
        YpEval = 10**-6
        XpEval = simpleIntegralDenullification(Ixp, XpEval, IotaX)
        YpEval = doubleIntegralDenullification(IYpint, YpEval, IotaY, IotaYdl)
        IxpIntegrated_0 = si.quad(Ixp, -IotaX, IotaX, args=(0, 0))[0]
        IxpIntegrated_E6 = si.quad(Ixp, -IotaX, IotaX, args=(0, XpEval))[0]
        print('XpEval and YpEval are :', XpEval, YpEval)

        IotaYBetter = calculateBetterLimits(IYpint, YpEval, IotaY, IotaYdl, 10 ** -10)[0]
        IotaYdlBetter = calculateBetterLimits(IYpint, YpEval, IotaY, IotaYdl, 10 ** -10)[1]
        print('IotaYdlBetter is :', IotaYdlBetter, 'and IotaYBetter is :', IotaYBetter)

        IypIntegrated_0 = si.nquad(IYpint, [[-IotaYBetter, IotaYBetter], [-IotaYdlBetter, IotaYdlBetter]], args=(0,))[0]
        IypIntegrated_E6 = si.nquad(IYpint, [[-IotaYBetter, IotaYBetter], [-IotaYdlBetter, IotaYdlBetter]], args=(YpEval,))[0]
        print('Value of the integrals : ', IxpIntegrated_0, IxpIntegrated_E6, IypIntegrated_0, IypIntegrated_E6)

        #Simple calculation part
        ValueAXp = IxpIntegrated_0 * IypIntegrated_0
        print(ValueAXp)
        ValueAYp = ValueAXp
        ValueExponentXp = IypIntegrated_0 * IxpIntegrated_E6
        ValueExponentYp = IxpIntegrated_0 * IypIntegrated_E6
        SigmaXp = sigmaCalc(XpEval, ValueExponentXp, ValueAXp)
        SigmaYp = sigmaCalc(YpEval, ValueExponentYp, ValueAYp)
        SigmaXpMilliRad = SigmaXp * 10**6
        SigmaYpMilliRad = SigmaYp * 10**6
        return [SigmaXpMilliRad, SigmaYpMilliRad]
    #Dl dependancy check on Iyp
    elif Iyp(0,0,0) == Iyp(0,100,0):...
    else:
        print("Computation time too long, DeltaLambda variation on both axis -> not possible")
        return 0
```

Algorithm to enhance limit calculation



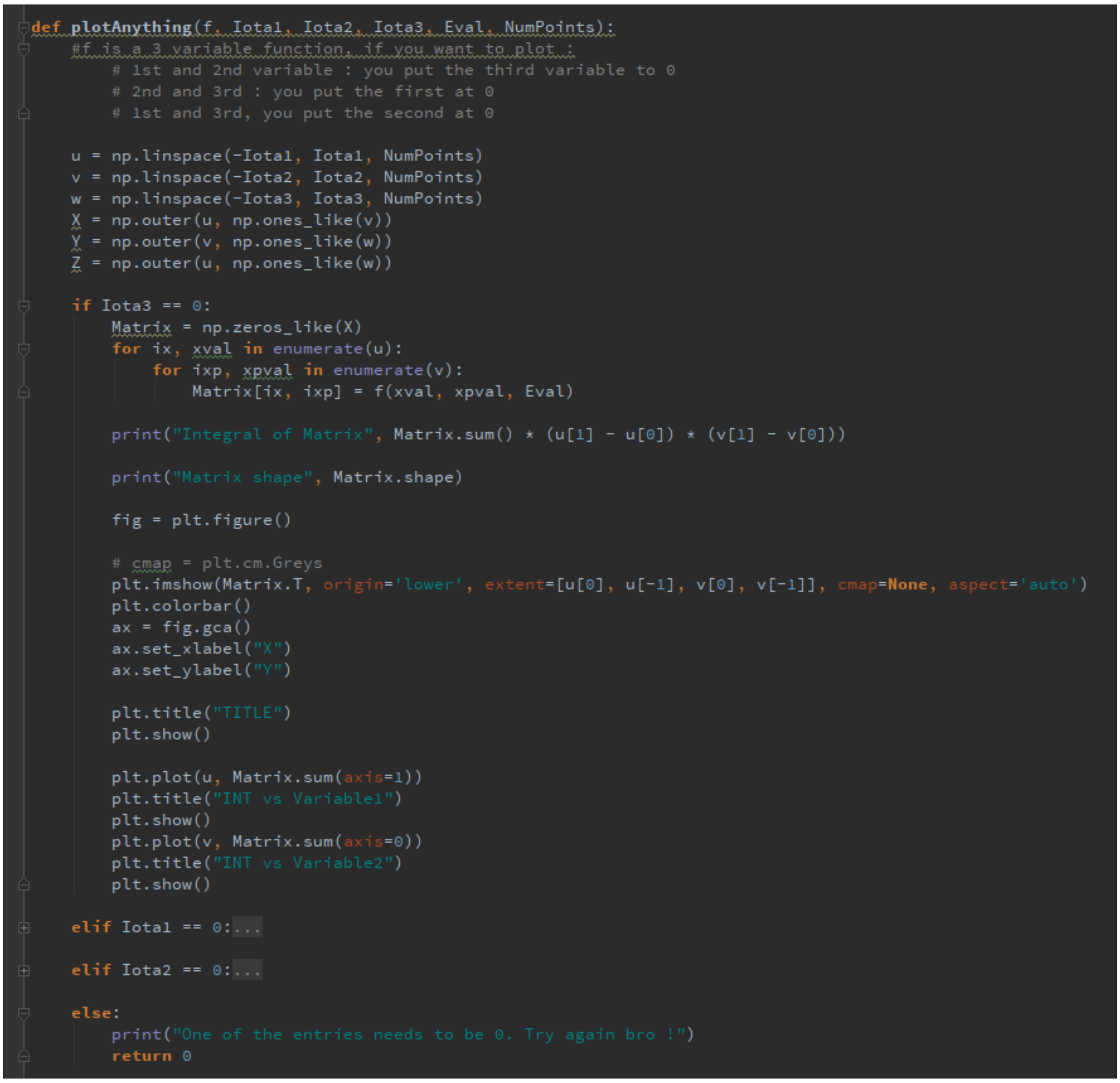
```python
def calculateBetterLimits(f, Eval, SigmaSource1, SigmaSource2, Epsilon):
    # we take the limits calculated previously and enlarge them if needed
    # what we do here is integrate along a line to see if our limits are big enough because in case of a 2D gaussian
    # that makes an angle with the x-axis, the previous algorith will fail
    # only for double integrals
    Iota1 = SigmaSource1
    Iota2 = SigmaSource2
    k = 1
    fPerm = lambda x, y, z: f(y, x, z)
    while k == 1:
        k = 0
        while si.quad(f, -Iota1, Iota1, args=(Iota2, Eval))[0] / f(0, 0, 0) > Epsilon:
            Iota2 = Iota2 * 2
            while si.quad(fPerm, -Iota2, Iota2, args=(Iota1, Eval))[0] / f(0, 0, 0) > Epsilon:
                Iota1 = Iota1 * 2
            k = 1
        while si.quad(fPerm, -Iota2, Iota2, args=(Iota1, Eval))[0] / f(0, 0, 0) > Epsilon:
            Iota1 = Iota1 * 2
            while si.quad(f, -Iota1, Iota1, args=(Iota2, Eval))[0] / f(0, 0, 0) > Epsilon:
                Iota2 = Iota2 * 2
            k = 1
    return Iota1, Iota2
```

Function to avoid the integrals being equal to 0



```python
def doubleIntegralDenullification(f, k, Iota1, Iota2):
    #
    Eval = k
    Integral = si.nquad(f, [[-Iota1, Iota1], [-Iota2, Iota2]], args=(Eval,))[0]
    if Integral == 0.0:
        while Integral == 0.0:
            Eval = Eval / 2
            Integral = si.nquad(f, [[-Iota1, Iota1], [-Iota2, Iota2]], args=(Eval,))[0]
        Eval = Eval / 4
    return Eval
```

Plotting function



```python
def plotAnything(f, Iota1, Iota2, Iota3, Eval, NumPoints):
    #f is a 3 variable function, if you want to plot :
        # 1st and 2nd variable : you put the third variable to 0
        # 2nd and 3rd : you put the first at 0
        # 1st and 3rd, you put the second at 0

    u = np.linspace(-Iota1, Iota1, NumPoints)
    v = np.linspace(-Iota2, Iota2, NumPoints)
    w = np.linspace(-Iota3, Iota3, NumPoints)
    X = np.outer(u, np.ones_like(v))
    Y = np.outer(v, np.ones_like(w))
    Z = np.outer(u, np.ones_like(w))

    if Iota3 == 0:
        Matrix = np.zeros_like(X)
        for ix, xval in enumerate(u):
            for ixp, xpval in enumerate(v):
                Matrix[ix, ixp] = f(xval, xpval, Eval)

        print("Integral of Matrix", Matrix.sum() * (u[1] - u[0]) * (v[1] - v[0]))

        print("Matrix shape", Matrix.shape)

        fig = plt.figure()

        # cmap = plt.cm.Greys
        plt.imshow(Matrix.T, origin='lower', extent=[u[0], u[-1], v[0], v[-1]], cmap=None, aspect='auto')
        plt.colorbar()
        ax = fig.gca()
        ax.set_xlabel("X")
        ax.set_ylabel("Y")

        plt.title("TITLE")
        plt.show()

        plt.plot(u, Matrix.sum(axis=1))
        plt.title("INT vs Variable1")
        plt.show()
        plt.plot(v, Matrix.sum(axis=0))
        plt.title("INT vs Variable2")
        plt.show()

    elif Iota1 == 0:...

    elif Iota2 == 0:...

    else:
        print("One of the entries needs to be 0. Try again bro !")
        return 0
```

Abstracts

Abstract

PSA is a formalization used in X-ray optics, that allows the user to consider that every distribution is Gaussian and that reduces an optical element to a matrix and an acceptance function. The PSA-software uses this method to calculate the brilliance of an X-ray beam according to phase, space and relative wavelength with the distribution of the initial beam and the characteristics of the elements of the optical system that the beam crosses.
But this software runs on Mathematica, which is outdated, and a new version on Python needed to be implemented. This generated several issues due to the fact that Mathematica works with an infinite precision whereas Python does not and forced a change in the integration procedure. Finally, the software works for most optical components, renders figures of the distributions but sadly takes several minutes to calculate the flux.

Résumé

PSA est une approximation, utilisée en optique des rayons X, qui permet de considérer que toutes les distributions sont des gaussiennes et d'approximer tout élément optique par une matrice et une acceptance. Le logiciel PSA utilise cette méthode pour calculer la distribution de la brillance d'un faisceau de rayon X en fonction de paramètres spaciaux, angulaires et de la longueur d'onde relative, en connaissant les paramètres du faisceau initiales et les caractéristiques de chaque élément du système optique.
Le souci est que ce logiciel fonctionne avec Mathematica, qui n'est plus mis à jour, et il doit être converti en Python. A cause de la différence de précision entre Mathematica et Python ainsi que le passage du symbolique à l'analytique, une nouvelle routine d'intégration a dû être implémenté. Le logiciel fournit des résultats corrects pour la plupart des composants optiques, rend les tracés des distributions mais malheureusement prend plusieurs minutes pour rendre les résultats.