PyWolf: A PyOpenCL implementation for simulating the propagation of partially coherent light

Tiago E. C. Magalhães\textsuperscript{a,b,c,}, José M. Rebordão\textsuperscript{a,b}

\textsuperscript{a}Departamento de Física, Faculdade de Ciências, Universidade de Lisboa, Edifício C8, Campo Grande, PT1749-016 Lisboa, Portugal
\textsuperscript{b}Instituto de Astrofísica e Ciências do Espaço, Edifício C8, Campo Grande, PT1749-016 Lisboa, Portugal
\textsuperscript{c}Present address: Instituto de Física de Materiais Avançados, Nanotecnologia e Fotônica (IFIMUP), Departamento de Física e Astronomia, Faculdade de Ciências, Universidade do Porto, Rua do Campo Alegre s/n, 4169-007 Porto, Portugal

Abstract
We present PyWolf, an open-source software capable of performing numerical simulations of partially coherent light propagation from two-dimensional light sources. PyWolf computes the evolution of a user-defined cross-spectral density function in the Fresnel and far field approximations, which enables the retrieval of second-order optical quantities of interest such as the spectral degree of coherence and spectral density for a given frequency. The open-source tool kit PyOpenCL is used to increase the computation speed. We present examples of propagation of different source models and optical systems to validate our implementation. Performance results for the computation speed when using parallel computation through PyOpenCL is shown. Source models and propagation systems can be easily added to PyWolf, which has a graphical user interface built with PyQt5. This software can be of great utility for partially coherent light simulation problems that are difficult to treat analytically.

Keywords: Coherence theory; PyOpenCL; Partially coherent light propagation; Parallel computation; Cross-spectral density.

PROGRAM SUMMARY
Program Title: PyWolf
Licensing provisions: GPLv3
Programming language: Python
Repository: https://github.com/tiagoeccmagalhaes/PyWolf
External routines: NumPy, SciPy, Matplotlib, PyOpenCL, PyQt5

Nature of problem: Numerical simulations of the propagation of partially coherent light from planar sources in the Fresnel or far field approximations. Computation time can be improved by using optimized versions of the Fast Fourier Transform (FFT) algorithm, but other calculations can only be increased through parallel computation.

Solution method: We use the open-source toolkit PyOpenCL to perform parallel computation on time-consuming calculations. The user can easily modify and add more features to the software, such as source models and propagation methods. A graphical user interface is also built using PyQt5 which enables the user to set input parameters, including the user’s custom models, view the simulation results, export data, and save and load sessions.

1. Introduction

In second-order coherence theory, partially coherent light is described in the space-time domain by the mutual coherence function \([1]\) or in the space-frequency domain by the cross-spectral density function \([2]\). In free space, these two quantities satisfy a pair of wave equations \([3, 4, 5]\) (known as the Wolf equations). For propagation analysis purposes, it is usual to use the space-frequency approach, since the response of matter to light fields is, in most linear regime cases, frequency-dependent \([2]\). For planar sources, the integral solution of the Wolf equations yields a four-dimensional (4D) integral. In the Fresnel approximation \([6, 7, 8]\), this integral reduces to two two-dimensional (2D) Fourier transforms, aside from the multiplication by the so-called propagators or kernels.

If the cross-spectral density function is known, we can extract optical quantities such as the spectral density and the correlation between pairs of points (i.e., the spectral degree of coherence). In general, the computational array describing the cross-spectral density function at a given angular frequency is 4D \([9]\), and complex values must be stored for each pair of 2D points. As the size of this 4D array increases, the amount of system memory increases as well as the computation time. For instance, consider two 4D arrays with sizes \(N^4\) and \((N + 1)^4\). The difference between the number of elements in both arrays is \(4N^3 + 6N^2 + 4N + 1\). If \(N = 100\), the difference between both arrays is approximately \(4 \times 10^6\) elements. Although highly optimized algorithms for solving 2D Fourier transforms exist, such as the Fast Fourier Transform (FFT) algorithm, we are still left with other computational tasks that require a large computation time, such as building the initial source model into an array and multiplying by the propagators.

Several techniques and approximations have been developed in the past, not only to decrease the computation time but also to simplify the analytical calculations. For example, the coherence-mode decomposition method introduced by Emil...
Wolff\cite{10} reduces the calculation to two-dimensional (2D) integrals. It consists of writing the cross-spectral density as a superposition of contributions from completely coherent modes (for a detailed description, see reference [11]). For instance, perfectly coherent light is characterized by one mode and as the spectral degree of coherence “decreases”, the number of modes increases. To find these modes, we need to find solutions to a 2D equation rather than a 4D one. Burvall et al.\cite{12} developed a method, known as the elementary function method, which is easier to handle numerically when compared to the coherent-mode decomposition. However, the source cross-spectral density cannot be arbitrary and must satisfy several assumptions (e.g., cross-spectral density must be real). Several other methods that involve approximations on the source model were proposed (see reference [12] and references therein). In this work, we are interested in a general approach since we want to use arbitrary source models, where the propagation is limited only in terms of the Fresnel approximation and single-frequency description. To deal with the huge computation time required to perform operations on 4D arrays, we implemented an algorithm that uses parallel computing devices to decrease the computation time, the results of which have already been published [9, 13]. Although our initial problems are related to light propagation and imaging, its formal mathematical structure can be found elsewhere, for example in [14] [15].

In this work, we present an open-source software written in the Python language capable of propagating the cross-spectral density function using parallel computing devices in very time-consuming tasks. To achieve this goal, we used PyOpenCL\cite{16}, a tool kit that gives access through Python to the Open Computing Language (OpenCL), a programming language developed by the Khronos Group to support heterogeneous computing [17]. To enable a user-friendly environment and easy customization of source models and propagation functions, we added a graphical user interface using the PyQt5 package. PyWolf aims at providing researchers a framework for faster simulations of partially coherent light propagation from user-defined source models. Researchers in different fields (e.g., coherence theory, optical instrumentation, and astrophysics) may find this tool important when dealing with the propagation of partially coherent light in optical systems of interest.

This article is structured as follows. In Section 2 we will briefly review the theory of the propagation of partially coherent light from planar sources in the space-frequency domain using the cross-spectral density function. In Section 3 we will present the data and numerical model for the propagation of the cross-spectral density function and discuss the PyOpenCL implementation. In Section 4 we discuss PyWolf's framework and graphical user interface. In Section 5 we present simulation examples to validate our implementation. In Section 6 we discuss performances in terms of computation time with and without the use of PyOpenCL in the computational tasks. Finally, in Section 7 we conclude and discuss future improvements.

2. Theory

In this section, we will summarize the theory underlying the propagation of the cross-spectral density function from the source plane to the observation plane in free space. For a more detailed analysis of coherence theory and propagation of partially coherent light, we refer the reader to references [14] [18] [15].

2.1. Propagation in a homogeneous and isotropic medium

With reference to Fig. 1 let $W_\mathcal{A}(\xi_1, \xi_2, \omega)$ be the cross-spectral density function of plane $\mathcal{A}$, containing a source $\rho$, given by

\[
W_\mathcal{A}(\xi_1, \xi_2, \omega) = \sqrt{S_\mathcal{A}(\xi_1, \omega)} \sqrt{S_\mathcal{A}(\xi_2, \omega)} \mu_\mathcal{A}(\xi_1, \xi_2, \omega),
\]

(1)

where $\xi_1 = (\xi_1, \eta_1)$ and $\xi_2 = (\xi_2, \eta_2)$ are a pair of source points, $S_\mathcal{A}(\xi, \omega)$ is the spectral density at point $\xi$, and $\mu_\mathcal{A}(\xi_1, \xi_2, \omega)$ is the spectral degree of coherence. Both quantities can be extracted from the cross-spectral density function $W_\mathcal{A}(\xi_1, \xi_2, \omega)$ using their definitions [13]:

\[
S_\mathcal{A}(\xi, \omega) = W_\mathcal{A}(\xi, \xi, \omega),
\]

(2)

\[
\mu_\mathcal{A}(\xi_1, \xi_2, \omega) = \frac{W_\mathcal{A}(\xi_1, \xi_2, \omega)}{\sqrt{S_\mathcal{A}(\xi_1, \omega)} \sqrt{S_\mathcal{A}(\xi_2, \omega)}}.
\]

(3)

While the spectral density of the source $\rho$ may be rather arbitrary, the spectral degree of coherence is limited to the non-negative definiteness constraint of the cross-spectral density function [19] [20]:

\[
\int \int \int \int f^*(\xi_1) f(\xi_2) W(\xi_1, \xi_2, \omega) \, d\xi_1 \, d\eta_1 \, d\xi_2 \, d\eta_2 \geq 0.
\]

(4)

Let $\mathcal{B}$ be an observation plane parallel to plane $\mathcal{A}$ and distanced by $R$. In the framework of second-order coherence theory, the propagation of the cross-spectral density function $W(\xi_1, \xi_2, \omega)$ to plane $\mathcal{B}$ is given by [21] [27]:

\[
W_\mathcal{B}(x_1, x_2, \omega) = \int \int h^*(x_1, \xi_1, \omega) W_\mathcal{A}(\xi_1, \xi_2, \omega) \times h(x_2, \xi_2, \omega) \, d\xi_1 \, d\eta_1 \, d\xi_2 \, d\eta_2,
\]

(5)
where the integration is taken over plane $\mathcal{A}$, the asterisk $*$ denotes the complex conjugate and $h(x, \xi, \omega)$ is the response function, i.e., the propagation kernel from point $x$ to point $\xi$. In free space, $h(x, \xi, \omega)$ is given by

$$h(x, \xi, \omega) = \frac{\Lambda}{r} \exp(ikr), \quad (6)$$

where $i$ is the imaginary unit, $k$ is the wavenumber, $r = |r|$ is the distance between points $\xi$ and $x$ (see Fig. 1), and $\Lambda$ is the obliquity factor [22], which is defined depending on the type of source in plane $\mathcal{A}$. If source $\rho$ is primary (e.g., the Sun), the obliquity factor assumes the simple form of $\Lambda = 1/4\pi$. If the source is secondary (e.g., an aperture), it can be shown that $\Lambda$ is approximately given by [7]

$$\Lambda \approx \frac{k}{4\pi} (\cos \theta_1 + \cos \theta_2), \quad (7)$$

where

$$\cos \theta_1 = \frac{k(0)}{|k(0)|}, \quad (8)$$

$$\cos \theta_2 = \frac{(r-\xi) \cdot (r-\xi)}{|r-\xi|}, \quad (9)$$

$k^{(0)} = k^{(0)} \cdot \rho^{(0)}$ is the incident wave vector on the secondary source and $k^{(0)}$ denotes the z-component of the vector $k^{(0)} = (k^{(0)}_x, k^{(0)}_y, k^{(0)}_z)$, in the same direction of the optical axis $z$ (see reference [7] for more details). Let $g \rho$ be a finite region in plane $\mathcal{B}$ such that the distance $R$ is larger than the characteristic scale of this region. For instance, if $g \rho$ is a circular region with radius $\rho_0$, we assume that $R \gg \rho_0$. If we consider that points $x_1$ and $x_2$ are within region $g \rho$, we may approximate the term $\Lambda/r$ of equation (6) as

$$\frac{\Lambda}{r} \approx \frac{\Lambda}{R}. \quad (10)$$

Henceforth, we will always use this approximation. Substituting equation (6) into (5) and using the Fresnel approximation, the cross-spectral density at plane $\mathcal{B}$ can be written as

$$W_g(x_1, x_2, \omega) = C q'(R, x_1, \omega) q(R, x_2, \omega) \int q'(R, \xi_1, \omega)$$

$$\times \exp\left(-2\pi i K_1 \cdot \xi_1 \right) d\xi_1 \int W_g(\xi_1, \xi_2, \omega)$$

$$\times q(R, \xi_2, \omega) \exp\left(-2\pi i K_2 \cdot \xi_2 \right) d\xi_2, \quad (11)$$

where $C$ is a constant defined as

$$C = \frac{\Lambda_1 \Lambda_2}{R^2}, \quad (12)$$

$K$ is the spatial frequency of the 2D Fourier transform given by

$$K = \frac{\omega}{2\pi c R} x, \quad (13)$$

and $q(R, \xi, \omega)$ is a function defined as

$$q(R, \xi, \omega) = \exp\left(\frac{2\pi i R^2 \xi^2}{\Lambda} \right) \exp\left(\frac{\omega \xi^2}{\Lambda R} \right), \quad (14)$$

where $\xi^2 = \xi_1 \cdot \xi$ and we used the fact that, in free space, $\omega = 2\pi c / \Lambda$. The following properties can be easily derived from equation (14):

$$q(R, \xi, \omega) = q(\beta^2 R, \xi, \omega), \quad (15)$$

$$q(R, \xi_1, \omega) q'(R, \xi_1, \omega) = q(R, \sqrt{\xi_1^2 - \beta^2}, \omega), \quad (16)$$

$$q(R_1, \xi_1, \omega) q'(R_2, \xi_2, \omega) = q\left(\frac{R_1 R_2}{R_2 - R_1}, \xi, \omega\right), \quad (17)$$

$$q(R, \xi, \omega) q'(R, \xi, \omega) = 1, \quad (18)$$

where $R_1$ and $R_2$ are arbitrary distances and $\beta$ is a constant (e.g., a transverse magnification).

If the distance $R$ is large enough compared with the source $\rho$ dimensions, we can use the far field approximation [23], i.e., set $q(R, \xi, \omega) \approx 1$, and equation (11) reduces to

$$W_g(x_1, x_2, \omega) = C q(R, \sqrt{x_1^2 - x_1^\beta}, \omega) \int \exp\left(-\frac{\omega x_1 \cdot \xi_1}{c R} \right)$$

$$\times d\xi_1 d\eta_1 \int W_g(\xi_1, \xi_2, \omega)$$

$$\times \exp\left(-\frac{\omega x_2 \cdot \xi_2}{c R} \right) d\xi_2 d\eta_2, \quad (19)$$

where we used equation (16) to simplify the product of the two $q$ functions. For the sake of simplicity, we will hereafter refer to these functions as propagators. Notice that, apart from the propagators, the far field cross-spectral density is the 2D Fourier transform of the cross-spectral density in plane $\mathcal{A}$, which is expected from the van Cittert-Zernike theorem (see, for example, Chapter 5 of reference [14]).

After calculating $W_g(x_1, x_2, \omega)$ in equation (11) [or equation (19), depending on the approximation], we can use it to extract second-order optical quantities of interest in plane $\mathcal{B}$, such as the spectral density $S_g(x, \omega)$ and the spectral degree of coherence $\mu g(x_1, x_2, \omega)$, using equations (2) and (3), respectively.

2.2 Imaging System

We have just reviewed the propagation of the cross-spectral density function from the source plane to an observation plane, in free space. We can use the same procedure in more complex systems. We will briefly address the case of an ideal imaging system, which will be used to validate our numerical model. A more detailed analysis can be found, for example, in reference [8].

Let $\mathcal{A}$ be once again the source plane, as represented in Fig. 2, containing a primary source $\rho$. A positive thins lens, L, with focal length $f$ is placed in plane $\mathcal{B}$, distant by $R_1$. Since we are dealing with an ideal imaging system with one thin lens, the entrance and exit pupil will lie in the same plane, $\mathcal{B}$. The cross-spectral density $W_{gB}(x_1, x_2, \omega)$ immediately before plane $\mathcal{B}$ is calculated using the Fresnel approximation of
equation (11), and we can write it as

\[ W_B^{(+)}(x_1, x_2, \omega) = C_1 \mathcal{F}_{\xi_1} \left\{ q(R_1, \sqrt{\xi_1^2 - \xi_2^2}, \omega) \right\} \times W_{\mathcal{A}}(\xi_1, \xi_2, \omega) |_{x_1} q(R_1, \sqrt{x_1^2 - x_2^2}, \omega) , \]

where \( C_1 = (\Lambda_1^2 \Lambda_2^2)/R_1^2 \). The symbols \( \mathcal{F}_{\xi_1} \) and \( \mathcal{F}_{\xi_1}^{-1} \) denote the direct and inverse 2D Fourier transform, respectively, with respect to the spatial variable \( \xi \), and the subscript \( x \) represents the new spatial domain, i.e.,

\[ \tilde{f}(x) = \mathcal{F}_{\xi} \{ f(\xi) \}_x = \int_{-\infty}^{+\infty} f(\xi) \exp \left( -\frac{i\omega}{c} \cdot \xi \right) d\xi d\eta , \]

where \( f(\xi) \) is the function to be Fourier transformed and \( \tilde{f}(x) \) is its Fourier transform in the new domain \( x \). Immediately after plane \( B \), the cross-spectral density \( W_B^{(+)}(x_1, x_2, \omega) \) is given by [7]

\[ W_B^{(+)}(x_1, x_2, \omega) = T^*(x_1, \omega) W_B^{(-)}(x_1, x_2, \omega) T(x_2, \omega) , \]

where \( T(x, \omega) \) is the transmission function of \( L \), given by

\[ T(x, \omega) = P(x) q^*(x, \omega) , \]

where \( q^*(f, x, \omega) \) is defined in equation (14) and \( P(x) \) is the pupil function. The latter can be written as [8],

\[ P(x) = |P(x)| e^{i\phi(x)} , \]

where \( \Phi(x) \) is the aberration function (see, for example, reference [9]).

Let \( C \) be an imaging plane, distanced by \( R_2 \) from plane \( B \). The cross-spectral density \( W_C(\xi_1', \xi_2', \omega) \) at plane \( C \) is calculated, once again, using the Fresnel approximation of equation (11). Using the lens law

\[ \frac{1}{R_1} + \frac{1}{R_2} - \frac{1}{f} = 0 , \]

and using the properties of the \( q \) function of equations (16), we obtain

\[ W_C(\xi_1', \xi_2', \omega) = C_1 C_2 q(R_2, \sqrt{\xi_1'^2 - \xi_1^2}, \omega) \times \mathcal{F}_{\xi_1}^{-1} \left\{ \mathcal{F}_{\xi_1} \left\{ q(R_1, \sqrt{\xi_2^2 - \xi_1^2}, \omega) \right\}_x \right\} \times W_{\mathcal{A}}(\xi_1, \xi_2, \omega) |_{x_1} P'(x_1, x_2) \left\{ \xi_1' \right\}_1 , \]

where \( C_2 = (\Lambda_1^2 \Lambda_2^2)/R_2^2 \) and \( P'(x_1, x_2) \) is defined as

\[ P'(x_1, x_2) = P(x_1) P(x_2) . \]

If we consider an aberration free imaging system \( (\Phi = 0) \) and neglect the finite extent of the lens \( L \), we can set \( P(x_1, x_2) = 1 \), and accordingly, see, for example, reference [22]. Furthermore, if the lens \( L \) is placed at a distance \( R_1 \) such that

\[ q(R_1, \sqrt{\xi_2^2 - \xi_1^2}, \omega) \approx 1 , \]

the cross-spectral density at plane \( C \) will be simply

\[ W_C(\xi_1', \xi_2', \omega) = C' q(R_2, \sqrt{\xi_1'^2 - \xi_1^2}, \omega) W_{\mathcal{A}}(\xi_1, \xi_2, \omega) , \]

where \( C' = C_1 C_2 \). The spectral density \( S_C \) will thus be

\[ S_C(\xi', \omega) = C' S_{\mathcal{A}}(\xi', \omega) , \]

an expected result since planes \( \mathcal{A} \) and \( C \) are conjugated. However, we must keep in mind that we have neglected the finite extent of the lens \( L \) and aberrations, otherwise, we would not have a perfect mapping from plane \( \mathcal{A} \) to the image plane.

3. Numerical Propagation of the Cross-Spectral Density using PyOpenCL

We will now discuss the data representation for the cross-spectral density function for a single frequency. We will then describe the numerical implementation of the propagation of the cross-spectral density in free space.

3.1. Data representation

Let \( W_{\mathcal{A}}(\xi_1, \xi_2, \omega_0) \) be the cross-spectral density function, at a given angular frequency \( \omega_0 \), of plane \( \mathcal{A} \), which contains a light source \( \rho \), as illustrated in Fig. 1. Let also \( W^{(0)} \) be a \( N \times N \times N \times N \) array, representing a sampling of \( W_{\mathcal{A}}(\xi_1, \xi_2, \omega_0) \) in a finite squared-size portion of plane \( \mathcal{A} \). The superscript \( (0) \) denotes the initial plane from which we want to propagate the cross-spectral density function. We will denote \( W^{(0)} \) as the cross-spectral density array (CSDA) instead of the cross-spectral density matrix to avoid confusion from the electric cross-spectral density matrix found in the unified theory of polarization and coherence (e.g., see Chapter 9 of reference [18]).

Computationally, the position vectors \( \xi_1 \) and \( \xi_2 \) are defined as

\[ \xi_{1,2} = (\xi_{1,2}, \eta_{1,2}) = \Delta \xi (n_{1,2} - n_0) , \]
where $\Delta \xi$ is the spatial resolution of the CSDA (spatial sampling) and $n_{i,2}$ and $n_0$ are two adimensional vectors, respectively defined as

$$n_{i,2} = (i_{1,2}, -j_{1,2}) \ (i_{1,2}, j_{1,2} = 0, 1, \ldots, N - 1),$$

$$n_0 = \left( N \frac{1}{2}, N \frac{1}{2} \right),$$

where $i$ and $j$ are indexes of the CSDA, which correspond to the indexes of a typical numeric array. Figure 3 illustrates an example of a CSDA for $N = 5$, which has a total size of 625, i.e., $N^4$.

Let us define, for the sake of brevity, $M = N/2$. Since the CSDA $W^{(0)}$ is a 4D array, it can be described as

$$W^{(0)} = \begin{bmatrix} W_{0,0}^{(0)} & \cdots & W_{0,M}^{(0)} \\ \vdots & \ddots & \vdots \\ W_{M,0}^{(0)} & \cdots & W_{M,M}^{(0)} \end{bmatrix},$$

where $W_{i,j}^{(0)} (i, j = 0, 1, \ldots, N - 1)$ is a $N \times N$ array for a given point $\xi_l = \Delta \xi (n_{1,1} - n_0)$. Therefore, we can write $W_{i,j}^{(0)}$ as

$$W_{i,j}^{(0)} = \begin{bmatrix} W_{i,j,0,0}^{(0)} & \cdots & W_{i,j,0,M}^{(0)} \\ \vdots & \ddots & \vdots \\ W_{i,j,M,0}^{(0)} & \cdots & W_{i,j,M,M}^{(0)} \end{bmatrix},$$

where $W_{i,j,k,l}^{(0)}$ is a complex number which can be sampled from the cross-spectral density function, i.e.,

$$W_{i,j,k,l}^{(0)} = W [ (n_{1} - n_{0}) \Delta \xi, (n_{2} - n_{0}) \Delta \xi, \omega_0 ].$$

Thus, for a given source model, which is described by the spectral density and the spectral degree of coherence [see equation (1)], we must construct a given CSDA $W^{(0)}$, where each value $W_{i,j,k,l}^{(0)}$ will be sampled from the cross-spectral density function of such model.

### 3.2. Creation of the CSDA $W^{(0)}$ with PyOpenCL

To start the simulation, PyWolf requires an initial CSDA $W^{(0)}$. To define this and other arrays, PyWolf uses the NumPy Python library, namely, the $\text{npy}$ object. The initial CSDA $W^{(0)}$ can be created according to the selected model in PyWolf or uploaded to PyWolf through the $\text{npy}$ format. In the latter case, $W^{(0)}$ can be custom-built by the user or obtained by a previous simulation of PyWolf. If not uploaded, PyWolf will create this CSDA according to what the user-defined in the input parameters of PyWolf (see Section 4), which will next describe.

The first step is to define an "empty" $N \times N \times N \times N$ array and initialize it with zeros. Then, depending on the user input, we can add a geometrical model (e.g., a circle) by filling elements with ones (see, for example, Algorithm 6 in Appendix A). For instance, if at a given point source $\xi = \Delta \xi (i, j)$ light is not present, then we have that $W_{i,j}^{(0)} = 0 + 0 i$. Otherwise, if light is present, we have that $W_{i,j}^{(0)} = 1 + 0 i$. If we do not wish to apply any geometry, the initial CSDA is initialized with ones, rather than zeros.

After modeling the geometry, we must apply a given source model to the CSDA. In general, this involves the computation of $N^4$ elements in $W^{(0)}$, a heavy time-consuming task as we increase $N$. To decrease the computing time, PyOpenCL is used to compute each $N \times N$ matrix $W^{(0)}$. Four loops (for cycles) are used to go through $N^4$ elements. To increase the speed, we can use PyOpenCL to parallelize the last two loops. We will use the same procedure whenever we have to deal with four imbricated loops.

Mathematically, the source model in plane $\mathcal{A}$ can be described as

$$W_{\mathcal{A}}(\xi_1, \xi_2, \omega_0) = Y_{\mathcal{A}}(\xi_1) Y_{\mathcal{A}}(\xi_2) W_{\mathcal{A}}(\xi_1, \xi_2, \omega_0),$$

where $Y_{\mathcal{A}}(\xi)$ is a function defining the geometry in plane $\mathcal{A}$, consisting of only ones and zeros, and $W_{\mathcal{A}}(\xi_1, \xi_2, \omega_0)$ is the cross-spectral density function of a given model. An example of the creation of the CSDA $W^{(0)}$ for a Gaussian Schell-model source is described in Appendix A.

### 3.3. Spatial sampling and resolution

Computationally, the spatial resolution $\Delta x$ in the observation plane $\mathcal{B}$ will be determined by the 2D Fourier transforms of equation (11) or, in the far field approximation, of equation (19). In either case, the spatial frequency vector $\mathbf{K}$ of each 2D Fourier transform is given by equation (13). If we want to sample the observation plane with a spatial step $\Delta x$, the spatial frequency step $\Delta \mathbf{K}$ is, according to equation (13), given by

$$\Delta K = \frac{\omega}{2 \pi R} \Delta x.$$  

(38)

If we use the FFT algorithm to solve each Fourier transform, the spatial frequency resolution $\Delta K_{\text{FFT}}$ is given by

$$\Delta K_{\text{FFT}} = (N \Delta \xi)^{-1}.$$  

(39)
Thus, equalizing equations (38) and (39), we find that
\[
\Delta x = \frac{2\pi c R}{\omega N \Delta \xi}. \tag{40}
\]

Note that if we use zero padding \(24\) to perform the FFT, we must consider the total size of the 2D matrix in the term \(N\), even if, later, we decide to crop the matrix to its original size. The relation between the spatial resolutions \(\Delta \xi\) and \(\Delta x\) in equation (40) is used whenever we propagate the CSDA from one plane (with spatial resolution \(\Delta \xi\)) to another (\(\Delta x\)), separated by \(R\).

3.4. Propagation

We will now discuss the numerical implementation of the propagation of an arbitrary CSDA \(W^{(0)}\), for a given single frequency \(\omega_0\), according to the theory described in Section 2.1 in terms of the propagation between the source plane and the observation plane. Propagation through multiple planes, such as in imaging, is straightforward and follows the same steps, as long as the Fresnel approximation is valid. PyOpenCL is used to parallelize computational tasks that are time-consuming. Algorithm 1 describes all the steps taken during the propagation, starting in the creation of the source CSDA \(W^{(0)}\) and ending with the computation of the final CSDA \(W^{(6)}\) at the observation plane.

After creating the source CSDA \(W^{(0)}\) (lines 1-3 of Algorithm 2), we use equation (11) to propagate it to the observation plane, which is described in Algorithm 2 by the function \texttt{freespaceprop}. If the far field approximation is not used, the CSDA \(W^{(0)}\) will be multiplied by the propagators \((q\text{ functions})\) and we will obtain the CSDA \(W^{(1)}\). For this procedure, function \texttt{qfunction} is used (see Algorithm 3) and the computation time can be decreased if we use PyOpenCL, which parallelizes two of the four loops used to compute all elements of the CSDA. If the far field approximation is used, this step is skipped and, therefore, \(W^{(1)} = W^{(0)}\).

Algorithm 1 All steps used for propagation in free space of for a given model.

\textbf{Input:} \(N \times N \times N \times N\) matrix \(W^{(0)}\), array size integer \(N\), spatial sampling \(\Delta \xi\), central frequency \(\omega_0\), distance \(R\), speed of light \(c\), Boolean value usePyOpenCL, geometry parameters GeoPars, source model parameters, SourcePars

\textbf{Output:} \(N \times N \times N \times N\) matrix \(W^{(6)}\)

1: \(W^{(0)} \leftarrow \text{zeros}(N, N, N, N)\) \(\rightarrow\) Creates a 4D complex array filled with zeros.
2: \(W^{(0)} \leftarrow \text{geometry}(W, N, \text{GeoPars}, \text{usePyOpenCL})\) \(\rightarrow\) Builds the geometry of the source plane (e.g., function \texttt{Circle} of Algorithm 6).
3: \(W^{(0)} \leftarrow \text{model}(W, N, \text{SourcePars}, \text{usePyOpenCL})\) \(\rightarrow\) Builds the coherence model for the CSDA (e.g., function \texttt{GSMsource} of Algorithm 7).
4: \(W^{(0)} \leftarrow \text{freespaceprop}(W^{(0)}, N, \Delta \xi, \omega_0, R, c, \text{field, usePyOpenCL})\)

The next step is to compute \(2N^2\) 2D Fourier transforms. For this task, we will use the Fast Fourier Transform (FFT) algorithm. First, \(N^2\) 2D direct Fourier transforms in the \(\xi_1\) domain (lines 6-8 of Algorithm 2) are computed. When completed, we end up with the CSDA \(W^{(2)}\). Then, we need to swap axes 2 and 3 (corresponding to the \(\xi_2\) domain of the CSDA \(W^{(2)}\) with axes 0 a 1 (corresponding to the \(\xi_1\) domain). This transposed CSDA is now denoted as \(W^{(3)}\). Then, we perform \(N^2\) inverse 2D Fourier transforms in the \(\xi_2\) domain (lines 10-12 of Algorithm 2), obtaining the CSDA \(W^{(4)}\). After that, we perform the same transpose as before in the CSDA \(W^{(4)}\), switching axes 2 and 3 with 0 and 1 (lines 13 of Algorithm 2) and we obtain the new CSDA \(W^{(5)}\). Note that there is no need to perform this transpose since the cross-spectral density function is Hermitian \(13\), i.e.,
\[
W(x_2, x_1, \omega) = W^*(x_1, x_2, \omega). \tag{41}
\]

It is not a time-consuming task and we leave it for the sake of clarity. After computation of all 2D Fourier transforms, we need to calculate the spatial resolution \(\Delta x\) of equation (40). We are left to multiply the CSDA \(W^{(5)}\) by the \(q\) function again (line 15 of Algorithm 2). The final CSDA is denoted by \(W^{(6)}\).

3.5. Retrieval of the spectral density and spectral degree of coherence

After computing the CSDA \(W^{(6)}\), we can extract optical quantities for a given frequency \(\omega_0\) such as the spectral density \(S(x, \omega_0)\) and the spectral degree of coherence \(\mu(x_1, x_2, \omega_0)\). In the former case, since we are only dealing with a single frequency, the spectral density will be equivalent to the intensity distribution since, in arbitrary units, we have that
\[
I(x) = \int_{-\infty}^{\infty} S(x, \omega) \, d\omega. \tag{42}
\]
**Algorithm 3** Propagator $q$ function.

1. \textbf{function} $qfunction(W, N, \Delta \xi, \omega_0, \Delta \xi, c, \text{usePyOpenCL})$
2. \hspace{1em} for $i_1 = 0$ to $N - 1$ step 1 do
3. \hspace{2em} for $j_1 = 0$ to $N - 1$ step 1 do
4. \hspace{3em} $\xi_1 = (i_1 - N / 2, i_1 - j_1) \Delta \xi$
5. \hspace{2em} if usePyOpenCL = True then
6. \hspace{3em} for $i_2, j_2 = 0$ to $N - 1$ step 1 do in parallel
7. \hspace{4em} $\xi_2 = (i_2 - M, M - j_2) \Delta \xi$
8. \hspace{4em} $W_{i_1, j_1, i_2, j_2} \leftarrow W_{i_1, j_1, i_2, j_2}$
9. \hspace{4em} $\times \exp \left[ i \frac{\Delta \xi}{2 \pi} \left( \xi_2^2 - \xi_1^2 \right) \right]$
10. \hspace{2em} else
11. \hspace{3em} for $i_2 = 0$ to $N - 1$ step 1 do
12. \hspace{4em} for $j_2 = 0$ to $N - 1$ step 1 do
13. \hspace{5em} $\xi_2 = (i_2 - N / 2, N / 2 - j_2) \Delta \xi$
14. \hspace{5em} $W_{i_1, j_1, i_2, j_2} \leftarrow W_{i_1, j_1, i_2, j_2}$
15. \hspace{5em} $\times \exp \left[ i \frac{\Delta \xi}{2 \pi} \left( \xi_2^2 - \xi_1^2 \right) \right]$
16. \hspace{2em} return $W'$

**Algorithm 4** Retrieval of the spectral density array $S$ for a given CSDA $W$.

1. \textbf{function} $\text{specdensity}(W, N)$
2. \hspace{1em} $S \leftarrow \text{zeros}(N, N)$ \hspace{0.5em} \( \Rightarrow \) Creates a $N \times N$ array with zeros.
3. \hspace{1em} for $i = 0$ to $N - 1$ step 1 do
4. \hspace{2em} for $j = 0$ to $N - 1$ step 1 do
5. \hspace{3em} $S_{i, j} \leftarrow W_{i, j}$
6. \hspace{1em} return $S$

Therefore, for a single frequency $\omega_0$, we obtain $I(x) = S(x, \omega_0)$. The procedure for computing the spectral density through $W^{(0)}$ is described in Algorithm 4.

To compute the spectral degree of coherence, we need a 4D array since it depends on pairs of points $x_1$ and $x_2$. For the sake of simplicity, we will represent the spectral degree of coherence array as a 2D array by assuming that $x_1$ is a user-defined point $P_1 = (P_{x_1}, P_{y_1})$, which according to equation (31) can be represented as

$$P_1 = (P_{x_1}, P_{y_1}) = \Delta x \left( i_p - \frac{N}{2}, \frac{N}{2} - j_p \right),$$

where $\Delta x$ is the spatial resolution at the plane of the CSDA in analysis, $i_p$ and $j_p$ are constant indexes of the CSDA $W$, i.e., $i_p, j_p \in [0, 1, \ldots, N - 1]$. The procedure for computing the spectral degree of coherence array from a given CSDA $W$ is described in the function $\text{coherence}$ of Algorithm 5.

**Algorithm 5** Retrieval of the spectral degree of coherence array $dc$ for a given CSDA $W$.

1. \textbf{function} $\text{coherence}(W, N, i_p, j_p)$
2. \hspace{1em} $c \leftarrow \text{zeros}(N, N)$ \hspace{0.5em} \( \Rightarrow \) Creates an $N \times N$ complex array with zeros.
3. \hspace{1em} $S_0 \leftarrow W_{i_p, j_p}$ \hspace{0.5em} \( \Rightarrow \) Spectral density at point $P_1$
4. \hspace{1em} for $i = 0$ to $N - 1$ step 1 do
5. \hspace{2em} for $j = 0$ to $N - 1$ step 1 do
6. \hspace{3em} $S' \leftarrow W_{i,j}$ \hspace{0.5em} \( \Rightarrow \) Spectral density at point $x_2$
7. \hspace{3em} $c_{i,j} \leftarrow W_{i_p, j_p} / \left( \sqrt{S'} \sqrt{S_0} \right)$
8. \hspace{2em} return $c$

An overview of PyWolf’s GUI, summarizing the inputs, the packages used, and main tasks. An advantage of using a user-friendly GUI is that the user can add custom python codes in specific cases (e.g., source and propagation models) that are recognized by PyWolf’s GUI and can be readily selected, along with custom input parameters. We will next describe the main features of PyWolf.

**PyOpenCL.** PyWolf detects the available OpenCL platforms and their devices to be used through the PyOpenCL package. The options are then displayed so that the user can choose one. The user can also choose not to use PyOpenCL, in which case sequential computing will be used in the propagation algorithm or in creating the source CSDA $W^{(0)}$.

**CSDA size.** The user chooses an input integer value $N$. It will be used to create the CSDA, which is a NumPy complex array of shape $N \times N \times N \times N$.

**Source model.** PyWolf enables users to add custom source models by placing Python codes in the source model folder. During initiation, PyWolf scans for source models in that specific folder and adds that option in the GUI. When the user chooses a given source model, the input parameters are requested from the user. For example, if a Python code containing the Gaussian Schell-model source is detected, this option will appear. If it is selected, entries for the effective correlation length $\sigma_e$ and the spectral density standard deviation $\sigma_s$ [see equation (57)] will be displayed and the user can fill in their values. Additionally, the user can add two extra options for a given source model: (i) add a custom geometric form, which acts as an optical mask, and (ii) add a custom spectral density. In both cases, the user can add custom-made Python scripts to a specific folder that PyWolf scans and adds as an option in the GUI, along with the respective entries for the input parameters. Lastly, the user can choose to upload a custom CSDA in a NumPy’s npy format, which may have either been exported from a previous simulation or built by the user.

**Propagation planes.** After defining the source model, the user must define the propagation system characteristics. First, the user selects the number of propagation planes, located after the source plane. In the current version, the maximum number is 3. The propagation between two planes obeys equation (11).
each propagation plane, the user defines the distance $R_i (i = 1, 2, 3)$ relative to the previous plane and an option for using the far-field approximation of equation (19), which decreases the simulation time. Moreover, the user can decide to add a custom pupil (e.g., a circular aperture) and an optical device (e.g., a thin lens). In both cases, the user can add custom-made Python scripts to a specific folder that is detected by PyWolf during initialization.

**CSDA and propagation algorithm.** After all required input parameters and options are selected, the user starts the simulation by pressing a specific button. PyWolf starts by creating the class CSDA which contains a $N \times N \times N \times N$ complex array. It then builds the source model according to the user input. The computation time is decreased if the user chooses to use PyOpenCL. Then, PyWolf will propagate the source CSDA to the final observation plane. For instance, if the number of propagation planes is 3, PyWolf implements a loop with three cycles: propagation from the source plane $A$ to plane $B$, from plane $B$ to plane $C$, from plane $C$ to plane $D$. Likewise, if the user decides to use PyOpenCL, the propagation algorithm (Algorithm 2) will be executed at the chosen parallel computing device to decrease the computation time. Note that, in this version, NumPy’s FFT is the predefined algorithm to perform the 2D Fourier transforms.

**Plot and data export.** After the simulation, the main results are plotted in PyWolf’s GUI using the matplotlib repository. For both the source and propagated CSDAs, PyWolf plots three figures: (i) the image (i.e., spectral density for the chosen angular frequency), (ii) the spectral degree of coherence $\mu(P_1, x_2, \omega_0)$ (3D graph), where $P_1 = (P_{1x}, P_{1y})$ is a user-defined point, and (iii) the spectral degree of coherence $\mu(P_1, P_{2x}, y_2, \omega_0)$ (2D graph), where $P_1$ is once again a user-defined point and $P_{2x}$ a user-defined coordinate for vector $x_2$. Finally, the user can export all data, including the source and propagated CSDAs in NumPy’s npy format.

5. Simulation examples

In what follows, we will present four simulations performed in PyWolf to validate and illustrate our numerical implementation. We will also show how different types of studies can take advantage of PyWolf’s PyOpenCL implementation and what are the typical inputs needed. The main inputs for the four simulations are illustrated in Table 1. Note that the current version of PyWolf is based on the approximation of equation (19) and, therefore, PyWolf’s simulations and the accuracy of its results must take into account its validity.

5.1. Fresnel propagation of a perfectly coherent squared source

Let $\rho_1$ be a homogeneous squared source with length $a$, perfectly coherent, quasi-monochromatic with central frequency $\omega_0$ and in plane $A$. The cross-spectral density function $W_{\rho_1}$ of $\rho_1$ can be written as

$$W_{\rho_1}(\xi_1, \xi_2, \omega_0) = A_1 \text{rect} \left( \frac{\xi_1}{2a} \right) \text{rect} \left( \frac{\eta_1}{2a} \right) \times \text{rect} \left( \frac{\xi_2}{2a} \right) \text{rect} \left( \frac{\eta_2}{2a} \right),$$

where $A_1$ is a positive constant. Let $B$ be an observation plane, distance from plane $A$ by $R$, where the Fresnel approximation is valid. The intensity $I(x)$ at plane $B$, using both the Fresnel approximation and that of equation (19), is found to be [22]

$$I(x) = \frac{1}{4} \left\{ [C_F(X_0) - C_F(X)]^2 + [S_F(X_0) - S_F(X)]^2 \right\}$$

$$\times \left\{ [C_F(Y_0) - C_F(Y)]^2 + [S_F(Y_0) - S_F(Y)]^2 \right\}^2,$$

where $X$ and $Y$ are modified spatial coordinates defined as

$$X_x = \pm \frac{k}{cR} \left( \frac{a}{2} \pm x \right),$$

$$Y_y = \pm \frac{k}{cR} \left( \frac{a}{2} \pm y \right),$$

Figure 4: (Color online) Illustration of PyWolf’s GUI inputs and main tasks and packages used.
and $C_F(x)$ and $S_F(x)$ are the Fresnel integrals, given by [25]

$$C_F(X) = \int_0^X \cos \left( \frac{\pi r^2}{2} \right) dr, \quad (48)$$

$$S_F(X) = \int_0^X \sin \left( \frac{\pi r^2}{2} \right) dr. \quad (49)$$

If the approximation of equation [10] is not valid, the current version of PyWolf will not retrieve valid results. In this case, the intensity $I(x)$ can be calculated through the equivalent of equation (5) for field propagation in free space from a primary source [7]:

$$I(x) = \left| \frac{1}{4\pi} \int Q(\xi, \omega_0) \exp \left( i \omega_0 t/c \right) \frac{\xi}{r} d\xi d\eta \right|^2, \quad (50)$$

where $Q(\xi, \omega_0)$ is the source density for a primary source and we used the fact that $I(x) = S(x, \omega_0)$.

For illustrating purposes, we wish to observe the so-called dark-spot in the Fresnel regime at a specific distance from the coherent source. According to the numerical simulations performed by Le Bolloc’h et al. [26], the dark spot is located at a distance $R$, given by

$$R \approx \frac{a^2}{2\pi \lambda_0}. \quad (51)$$

We will next show how to perform the simulation for this propagation system (plane $\mathcal{A}$ to $\mathcal{B}$) using PyWolf. We are mainly interested in comparing the result for the spectral density $S(x, \omega_0)$, at an angular frequency $\omega_0$, with the intensity distribution described by equation (45).

To construct the source CSDA $W^{(0)}$ with a given spatial resolution step of $\Delta \xi$, we can write the size of the square $a$ as a function of $\Delta \xi$ and the number of elements in each axis, i.e.,

$$a = N_a \Delta \xi, \quad (52)$$

where $N_a$ is an adimensional positive integer, defining the number of “pixels” on the square’s side, which is an input to PyWolf.

The dark spot distance [equation (51)] is therefore given by

$$R = \frac{(N_a \Delta \xi)^2}{2\pi \lambda_0}. \quad (53)$$

For convenience, we will impose that the resulting minimum spatial sampling (i.e., spatial resolution) $\Delta x$ in plane $\mathcal{B}$ is equal to that of the source plane $\mathcal{A}$, i.e., $\Delta x = \Delta \xi$. Thus, substituting $\Delta x = \Delta \xi$ in equation (40), substituting equation (51) into (40), and solving for $N_a$, we find that

$$N_a = \sqrt{2\pi N}. \quad (54)$$

Thus, the values $R$ and $N_a$ are now functions of $N$ (or $N_z$ when a zero-padding is used), $\lambda$, and $\Delta \xi$.

To perform the simulation in PyWolf, we start by defining the CSDA size parameter $N = 200$. This choice is driven by the amount of system memory. Additionally, we will use a zero padding procedure in the FFT of $N_z = 512$ and, therefore, according to us that $I(x) = S(x, \omega_0)$.

For the spatial resolution we set $\Delta \xi = 0.0351 \mu m$ so that $a = 2 \mu m$. For the central frequency we set $\omega_0 = 1.2566 \times 10^{19}$ rad/s (in free space, $\lambda_0 \approx 1.5 \AA$). The distance between the source and observation planes is set as the dark spot distance of equation (53), which gives $R = 4.244$ nm. Notice that, in this case, the paraxial approximation of equation (10) is valid. The Fresnel number $N_F$ in this case (see, for example, Chapter 4.4 of reference [22]) is:

$$N_F = \frac{a^2}{\lambda_0 R} = \frac{(N_a \Delta \xi)^2}{\lambda_0 R} \approx 6.29, \quad (55)$$

and, therefore, the Fresnel and far field approximations will have significant differences. Then, we chose perfectly coherent light for the cross-spectral density model. Table II summarizes the input values in PyWolf for this simulation (Simulation example 1). Results for the spectral density are shown in Fig. 5 and compared with the theoretical ones which are calculated using equation (45). The absolute error is computed with an independent simulation using equation (50).
inner and outer radius $a$ and $o$, respectively, and central angular frequency $\omega_0$. We can write the cross-spectral density function of $\rho_2$ at the source plane $\mathcal{A}$ as

\[
W_\rho(x_1, x_2, \omega_0) = A_2 \left[ \text{circ} \left( \frac{x_1}{a_0} \right) \text{circ} \left( \frac{x_2}{a_0} \right) - \text{circ} \left( \frac{x_1}{a_i} \right) \text{circ} \left( \frac{x_2}{a_i} \right) \right] \times \delta(x_2 - x_1, \omega_0),
\]

where $A_2$ is a positive constant and $\text{circ}(\xi)$ is the circle function defined as

\[
\text{circ}(\xi) = \begin{cases} 
1 & , |\xi| < 1 \\
\frac{1}{\pi} & , |\xi| = 1 \\
0 & , |\xi| > 1
\end{cases}.
\] (57)

Let $B$ be the observation plane located in the far-field at a distance $R_1$ from the source plane $\mathcal{A}$. For the sake of brevity, let us define $b = |x_2 - x_1|$ and $b' = x_2^2 - x_1^2$. The spectral degree of coherence at $B$ can be calculated using the van Cittert-Zernike theorem [14]:

\[
\mu(b, b', \omega_0) = \left\{ \frac{2J_1(ka_0b/R_1)}{ka_0b/R_1} \right\} - \frac{a_i}{a_o} \left\{ \frac{2J_1(ka_0b/R_1)}{ka_0b/R_1} \right\}
\times \exp \left( -\frac{i\omega_0 b'}{c R} \right),
\] (58)

where $J_1$ is the first-order Bessel function of the first kind. We seek to simulate the propagation of the cross-spectral density function of source $\rho_2$, extract the spectral degree of coherence and compare it with the theoretical values of equation (58).

5.2 Far-field propagation of an incoherent annular source

Let us now consider the case of a homogeneous, perfectly incoherent, and quasi-monochromatic annular source $\rho_2$ with inner and outer radius $a_i$ and $a_o$, respectively, and central angular frequency $\omega_0$. We can write the cross-spectral density function of $\rho_2$ at the source plane $\mathcal{A}$ as

\[
W_\rho(\xi_1, \xi_2, \omega_0) = A_2 \left[ \text{circ} \left( \frac{\xi_1}{a_o} \right) \text{circ} \left( \frac{\xi_2}{a_o} \right) - \text{circ} \left( \frac{\xi_1}{a_i} \right) \text{circ} \left( \frac{\xi_2}{a_i} \right) \right] \times \delta(\xi_2 - \xi_1, \omega_0),
\] (56)

To perform this simulation, we chose the array size parameter $N = 200$ and did a zero padding for the FFT with a total size of $N_z = 512$, similar to what was used in the examples of Section 5.4. For the central wavelength, we used $\omega_0 = 3.7699 \times 10^{15}$ rad/s (yellow color), while for the spatial resolution we chose $\Delta \xi = 1 \times 10^{-2}$ m. For the dimensional parameters of the source geometric model, we chose $N_i = 5$ and $N_o = 11$. The distance between the source and observation planes is set to $R_1 = 5$ m, which is sufficient for a valid far-field approximation. Table 1 summarizes the input values (Simulation example 2). Results for the magnitude and phase of the simulated spectral degree of coherence are illustrated in Fig. 6 as well as the theoretical values for comparison using equation (58). Notice that the exponential term in equation (58) adds an additional phase term to the spectral degree of coherence.

5.3 Aperture effects in an imaging system with partially coherent light

We will now use PyWolf to simulate the propagation of light in thin lens imaging, as illustrated in Fig. 2. The goal is to validate PyWolf’s simulation results in the case where partially coherent light impinging on an aperture, such as that of a lens, produces changes in the image of an object. This type of problem can be found, for instance, in transfer functions of astronomical telescopes or microscopies, although, in this case, we tackle it with the cross-spectral density function.

Let $\rho_1$ be a perfectly incoherent, homogeneous, and quasi-monochromatic light source with central frequency $\omega_0$, located in plane $\mathcal{A}$. Suppose that a mask is placed at the source plane
so that it changes its shape. The cross-spectral density can be written as

$$W_{\mathcal{A}}(\xi_1, \xi_2, \omega_0) = Y(\xi_1)^* W_{\mathcal{P}}(\xi_1, \xi_2, \omega_0) Y(\xi_2), \tag{61}$$

where $Y(\xi)$ is a function that defines the geometry of the source with counter domain $[0, 1]$ and $W_{\mathcal{P}}(\xi_1, \xi_2, \omega_0)$ is the cross-spectral density for a perfectly incoherent source, given by

$$W_{\mathcal{P}}(\xi_1, \xi_2, \omega_0) = A_3 \delta(\xi_1 - \xi_2, \omega_0), \tag{62}$$

where $A_3$ is a positive constant. In this case, the source follows the letters “ia” [see Fig. 7(a)].

A positive and circular lens, $L$, with focal length $f$ and radius $a_{L}$, is placed at a distance $R_1$ from the source, in plane $\mathcal{B}$. Similar to previous examples, we define an adimensional parameter $N_L$ for the aperture radius $a_L$, such that

$$a_L = N_L \Delta \xi'. \tag{63}$$

Light from plane $\mathcal{A}$ reaching the lens plane, $\mathcal{B}$, will be partially coherent and modeled by the van Cittert-Zernike theorem. An image will be formed at plane $\mathcal{C}$, distanced by $R_2$. For an ideal imaging system, $R_1$ and $R_2$ satisfy equation (25). The quality of the image of an incoherent object will depend on the spatial coherence area of light reaching plane $\mathcal{B}$ and on the aperture size of the lens. On the other hand, the spatial coherence area of light reaching the lens will depend on the wavelength of light and the distance $R_1$. In terms of coherence theory, the cross-spectral density of $W_C(\xi'_1, \xi'_2, \omega_0)$ at the image plane will be given by equation (26) and the spectral density (or the intensity) is directly extracted from $W_C(\xi'_1, \xi'_2, \omega_0)$. However, since the source is perfectly incoherent, instead of using coherence theory to describe the image in plane $\mathcal{C}$, we can use a well-known result from Fourier optics which states that for incoherent objects, the intensity $I_\mathcal{C} (\xi', \eta')$ of the image is given by (see, for example, Section 6.3 of reference [22]):

$$I_\mathcal{C} (\xi', \eta') = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} p(\xi - \xi', \eta - \eta') I_\mathcal{A}(\xi, \eta) \, d\xi \, d\eta, \tag{64}$$

where $p(x, y)$ is the impulse response given by:

$$p(x, y) = \frac{\lambda_0 R_2}{\pi a \sqrt{x^2 + y^2}} I_1 \left( \frac{2\pi a \sqrt{x^2 + y^2}}{\lambda_0 R_2} \right)^2. \tag{65}$$

To validate our results, we will compare our results with that of equation (64).

To perform this simulation, we set the size parameter $N = 200$, without zero padding. Then, we choose the central angular frequency to $\omega_0 = 2.8560 \times 10^{12}$ rad/s (red color) and a spatial...
Figure 7: (Color online) Imaging partially coherent light. (a) Spectral density $S(\xi, \omega_0)$ at the source plane $A$ using the CSDA $W^{(0)}$. (b) Simulated spectral density $S(\xi', \omega_0)$ at the image plane $C$ using the CSDA $W^{(6)}$. (c) Simulated image using equation (64) and the image in (a).

Figure 8: (Color online) Gaussian Schell-model beam. (a) Spectral density $S(\xi, \omega_0)$ at the source plane $A$ using the CSDA $W^{(0)}$. (b) Simulated magnitude of the spectral degree of coherence, $|\mu(0, x_2, \omega_0)|$ for $0 = (0, 0)$. (c) Simulated phase of the spectral degree of coherence, $\beta(0, x_2, \omega_0)$. (d) Simulated spectral density $S(\xi', \omega_0)$ at the observation plane $B$ using the CSDA $W^{(6)}$. (e) Cross-section of (b) at $y_2 = 0$ mm with the theoretical values using equation (65). (f) Cross-section of (c) at $y_2 = 0$ mm with the theoretical values using equation (65). (g) Absolute error of the simulated spectral density represented in (d) calculated using the expected values. (h), (i) absolute errors for the magnitude and phase of the simulated spectral degree of coherence, respectively. The insets show the absolute error in a smaller interval of $x$, namely, $[-6, 6]$ in (h) and $[-5, 5]$ in (i).
resolution of $\Delta \xi = 1 \mu m$. To define the source geometry, we uploaded a binary image of size 200×200 containing the letters “ia”. Then, we selected two propagation planes, $B$ (lens plane) and $C$ (image plane). At plane $B$, we set the circular aperture parameter to $N_1 = 11$. The spatial resolution $\Delta \omega$ at plane $B$ is calculated using equation (40), which gives $\Delta x = 1.65 mm$. Therefore, the lens aperture radius is $a = 1.815 cm$. For the distances between planes $A - B$ and $B - C$ we chose $R_1 = 0.5 m$ and $R_2 = 0.5 m$, respectively. To comply with equation (25), the focal length of the lens was set to $f = 0.25 m$. With these values, the lateral magnification is -1 and the spatial resolution in plane $C$ is the same as that of the source plane, i.e., $\Delta \xi' = \Delta \xi = 1 \mu m$. Table 1 summarizes the input values in PyWolf for this simulation (Simulation example 3). The source spectral density is illustrated in Fig. 7(a) and the results for the simulated source spectral density in the image plane are shown in Fig. 7(b). To compare the results obtained by PyWolf, Fig. 7(c) shows the result for the same simulation using equation (64), i.e., using Fourier optics.

5.4. Beam condition for a Gaussian Schell-model source

Let $\rho_4$ be a planar, secondary, quasi-monochromatic Gaussian Schell-model source with central frequency $\omega_0$ and contained in plane $A$. Its cross-spectral density is given by [13]

$$W_{\rho_4}(\xi_1, \xi_2, \omega_0) = A_4 \exp \left( \frac{\xi_1^2 + \xi_2^2}{4\sigma_s^2} \right) \exp \left( \frac{\xi_1^2 - \xi_2^2}{2\sigma_s^2} \right), \quad (66)$$

where $\sigma_s$ is the effective correlation length and $\sigma_s$ is the standard deviation of the spectral density, respectively. It can be shown (see Chapter 5 of reference [14]) that this source model can generate a beam if the following necessary and sufficient condition is fulfilled:

$$\frac{1}{4\sigma_s^2} + \frac{1}{2\sigma_s^2} \approx \frac{2\pi^2}{\lambda^2}. \quad (67)$$

If such is the case, the cross-spectral density at a given observation plane $\beta$ can be shown to be [14]

$$W(x_1, x_2, \omega) = \exp \left[ -\frac{1}{4(\psi^* - \beta^2)} \left( \psi^* x_1^2 + \psi x_2^2 - 2\beta x_1 \cdot x_2 \right) \right] \times \frac{\lambda^2(\omega)}{16(a^2 - b^2)(\psi^* - \beta^2)}, \quad (68)$$

where $a, b, \beta$, and $\psi$ are parameters defined as

$$a = -\frac{1}{4\sigma_s^2} + \frac{1}{2\sigma_s^2}, \quad (69)$$

$$b = \frac{1}{2\sigma_s^2}, \quad (70)$$

$$\beta = \frac{b}{4(a^2 + b^2)}, \quad (71)$$

$$\psi = \alpha - \frac{\lambda R}{2\omega}. \quad (72)$$

The spectral density and spectral degree of coherence can be readily extracted from equation (68) by using definitions in equations (2) and (3). We will therefore use PyWolf to simulate the propagation of this source model and compare the numerical results with that of equation (68).

To perform this simulation in PyWolf, we set the size parameter to $N = 200$, without zero padding. The angular frequency and spatial resolution chosen were $\omega_0 = 2.9920 \times 10^{15} \text{ rad/s}$ ($\Delta \omega = 630 \text{ nm}$) and $\Delta \xi = 0.1 \text{ mm}$, respectively. We then chose the Gaussian Schell-model source option in PyWolf. To construct the CSDA for this model, we need two adimensional parameters, $N_s$ and $N_\mu$, defined as

$$N_s = \frac{\sigma_s}{\Delta \xi}, \quad (73)$$

$$N_\mu = \frac{\sigma_\mu}{\Delta \xi}. \quad (74)$$

For these parameters, we chose $N_s = 21$ and $N_\mu = 41$. The distance between the source plane and observation plane was set to $R_1 = 3.1746 m$ so that $\Delta \xi = 1 \mu m$. The far-field approximation is not used. Table 1 summarizes the input values in PyWolf for this simulation (Simulation example 4) and the simulation results are represented in Fig. 8.

6. Performance

The first significant time-consuming process is the creation of an empty array of size $N^4$, which depends on the available RAM. Python’s NumPy package is used for such a task. Then, PyWolf builds the CDSA $W^{(0)}$ according to the user-defined model. In general, PyWolf will have to compute $N^4$ values, which is a heavy time-consuming task. If the user selects the option of using PyOpenCL, PyWolf will use the selected OpenCL’s platform and device to parallelize this task.

To evaluate the advantage of using parallel computing by means of PyOpenCL, we performed the computation of the CDSA $W^{(0)}$ for a Gaussian Schell-model source to compare the time computation when using PyOpenCL and when using only sequential computing (see Algorithm 7 in Appendix A). We chose three computers with the same operating system, Windows 10, to execute PyWolf and perform the same task, namely, the creation of a CDSA of a Gaussian Schell-model source. Table 2 summarizes computers’ specifications. Besides the difference in the central processing units (CPUs) and the graphical processing units (GPUs), the random access memory (RAM) is also different, which may limit the maximum value for the input size parameter $N$. For each computer, we varied the size parameter $N$ to analyze how the computation time evolves. Figures 9(a-c) shows results for the computation of the CDSA $W^{(0)}$ of a Gaussian Schell-model source. Clearly, using PyOpenCL is a significant advantage in terms of computation time.

For the purpose of evaluating the advantage of using PyWolf’s PyOpenCL implementation in a full simulation for the propagation of partially coherent light, i.e. from the creation of the source CDSA $W^{(0)}$ to the final computation of the CDSA $W^{(0)}$, we performed simulation example 2 (see Section 5.4).
Table 2: Characteristics of each computer system (CS) used for the performance analysis.

| CS  | CPU            | RAM  | GPU                  | OpenCL          |
|-----|----------------|------|----------------------|-----------------|
|     |                |      |                      | Platform        |
|     |                |      |                      | Device          |
| 1   | Intel i7-6950X | 128 GB | Nvidia GTX 1050 (4 GB) | Nvidia CUDA GTX 1050 |
|     |                |      |                      | Intel OpenCL i7-6950X |
| 2   | Intel i5-4690  | 32 GB  | AMD Radeon R9 380 (4 GB) | AMD APP R9 380 |
|     |                |      |                      | AMD APP i5-4690 |
| 3   | Intel i5-3230M | 8 GB   | Nvidia GT 740M (2 GB) | Nvidia CUDA 740M |
|     |                |      |                      | Intel OpenCL i5-3230M |

Figure 9: (Color online) Computation time of the CSDA $W^{(0)}$ versus the matrix size parameter size $N$ (a-c) for three computer systems (see Table 1) and total computation time of the simulation example 2 of Section 5.2 as a function of the matrix size parameter $N$ (d-e). (a) Computer system 1. The triangular markers represent the computation without using PyOpenCL, while the other markers use PyOpenCL with a different platform and device. (b) Computer system 2. The triangular markers represent the computation without using PyOpenCL, while the other marker uses PyOpenCL with the graphics card AMD Radeon 380. (c) Computer system 3. The triangular markers represent the computation without using PyOpenCL, while the other two types of markers represent the use of PyOpenCL with two different platforms and devices. (d) Computer system 2. The triangular markers represent the computation without using PyOpenCL, while the other marker uses PyOpenCL with the graphics card AMD Radeon 380. (e) Computer system 3. The triangular markers represent the computation without using PyOpenCL, while the other two types of markers represent the use of PyOpenCL with two different devices.

with and without using PyOpenCL, in two different computer systems. Figures 9(e-f) shows the computation time results for the computation of the final CSDA $W^{(0)}$ as a function of the array size parameter $N$. In terms of computation time, the advantage of using PyOpenCL is easily observed as $N$ increases.

7. Conclusion

We introduced PyWolf, an open-source Python software that performs simulations of propagation of partially coherent light in diffraction and imaging. This program takes advantage of the open-source toolkit PyOpenCL to perform parallel computation in order to reduce the computation time. It allows the user to easily add new options in terms of source or propagation models. PyWolf has a graphical user interface built with PyQt5 which not only enables the user to choose simulation optics easily but also recognizes new custom codes added to PyWolf, such as source models, geometries, spectral densities, and pupil functions. After each simulation, the user can retrieve optical quantities such as the spectral degree of coherence and spectral density. This allows users, for instance, to evaluate the impact of each source model in a given optical system. For a given simulation, the numerical error will stem mainly from the spatial sampling used for a given source model, which will affect the outcome of the Fourier transform algorithm - which in this first version of PyWolf is the NumPy’s FFT algorithm - and, therefore, any simulation must take into account the errors introduced by the sampling.

Future improvements of PyWolf can include the use of different transform algorithms, including other approaches to propagation from radially symmetric source models (e.g., the Hankel transform). In the current version, PyWolf only simulates the propagation of the cross-spectral density for a given frequency. Future versions will also include the option of using...
polychromatic light. Simulations of correlation-induced spectral changes can also be added, as done in reference [27]. Computationally, the main bottleneck of PyWolf is the system memory usage, i.e., RAM. As the size of the CSDA increases, the memory increases by a factor of $N^4$. PyWolf is intended to be a community-oriented open-source software, on behalf of the optical coherence community and we welcome and will acknowledge contributions for other computational methods, algorithms, and custom models.

Acknowledgment

Tiago E. C. Magalhães acknowledges the support from Fundação para a Ciência e a Tecnologia (FCT, Portugal) through the grant PD/BD/105952/2015, under the FCT PD Program PhD:SPACE (PD/00040/2012). This work was supported by FCT/MCTES through national funds (PIDDAC) by this grant UID/FIS/04434/2019. The authors are grateful to the referees for their suggestions.

Appendix A. Example of a CSDA $W^{(0)}$ for a Gaussian Schell-model source in a circular aperture

Suppose we want to create a Gaussian Schell-model source with angular frequency $\omega_0$ bounded by a circle of radius $a$. This could represent, for instance, a quasi-monochromatic laser beam passing through a circular aperture of radius $a$.

Let $W_{\text{CS}}(\xi_1, \xi_2, \omega_0)$ be the cross-spectral density of a Gaussian Schell-model source given by equation (37) and $Y_c(\xi)$ a circular aperture function defined as

$$Y_c(\xi) = \begin{cases} 1 & \text{if } |\xi| \leq a \\ 0 & \text{otherwise} \end{cases} \quad (A.1)$$

The cross-spectral density at plane $\mathcal{A}$ will be the product of $W_{\text{CS}}(\xi_1, \xi_2, \omega_0)$ with $Y_c(\xi_1)Y_c(\xi_2)$. To build the source CSDA $W^{(0)}$, we create a complex $N \times N \times N \times N$ array full of zeros (i.e., $0 + 0i$). We then apply the desired geometric shape. This task can be parallelized using PyOpenCL. Algorithm 6 illustrates the procedure for both sequential and parallel computation. It can be accomplished in different ways. For instance, we can simply upload a binary image to construct the source CSDA geometry.

We can now construct the final source CSDA by sampling the Gaussian Schell source model of equation (37). Once again, we have to compute all non-zero elements and can parallelize this task using PyOpenCL. Algorithm 7 illustrates an example of how we can construct the cross-spectral density of a Gaussian Schell-model-source.

Algorithm A.3 (continued)

1: for $i_1 = 0$ to $N - 1$ step 1 do
2: for $j_1 = 0$ to $N - 1$ step 1 do
3: $r_1 \leftarrow \sqrt{(i_1 - N/2)^2 + (j_1 - N/2)^2}$
4: if $r_1 \leq N_a$ then
5: $W_{i_1,j_1} \leftarrow \text{ones}(N,N)$
6: else
7: if usePyOpenCL = True then
8: for $i_2, j_2 = 0$ to $N - 1$ step 1 do in parallel
9: $r_2 \leftarrow \sqrt{(i_2 - N/2)^2 + (j_2 - N/2)^2}$
10: if $r_2 \leq a$ then
11: $W_{i_1,j_1,i_2,j_2} \leftarrow \text{ones}(N,N)$
12: else
13: for $j_2 = 0$ to $N - 1$ step 1 do
14: $r_2 \leftarrow \sqrt{(i_2 - N/2)^2 + (j_2 - N/2)^2}$
15: if $r_2 \leq N_a$ then
16: $W_{i_1,j_1,i_2,j_2} \leftarrow \text{ones}(N,N)$
17: return $W$

Algorithm 6 Creation of the source CSDA with a circular geometry using sequential or parallel computing.

1: function CircleC(W, N, N_a, usePyOpenCL)
2: for $i_1 = 0$ to $N - 1$ step 1 do
3: for $j_1 = 0$ to $N - 1$ step 1 do
4: $r_1 \leftarrow \sqrt{(i_1 - N/2)^2 + (j_1 - N/2)^2}$
5: if $r_1 \leq N_a$ then
6: $W_{i_1,j_1} \leftarrow \text{ones}(N,N)$
7: else
8: if usePyOpenCL = True then
9: for $i_2, j_2 = 0$ to $N - 1$ step 1 do in parallel
10: $r_2 \leftarrow \sqrt{(i_2 - N/2)^2 + (j_2 - N/2)^2}$
11: if $r_2 \leq a$ then
12: $W_{i_1,j_1,i_2,j_2} \leftarrow \text{ones}(N,N)$
13: else
14: for $j_2 = 0$ to $N - 1$ step 1 do
15: $r_2 \leftarrow \sqrt{(i_2 - N/2)^2 + (j_2 - N/2)^2}$
16: if $r_2 \leq N_a$ then
17: $W_{i_1,j_1,i_2,j_2} \leftarrow \text{ones}(N,N)$
18: return $W$

Algorithm 7 Creation of a Gaussian Schell-model (GMS) source using sequential or parallel computation.

1: function GSMSource(W, N, N_a, N_\mu, usePyOpenCL)
2: for $i_1 = 0$ to $N - 1$ step 1 do
3: for $j_1 = 0$ to $N - 1$ step 1 do
4: $r_1 \leftarrow \sqrt{(i_1 - N/2)^2 + (j_1 - N/2)^2}$
5: if usePyOpenCL = True then
6: for $i_2, j_2 = 0$ to $N - 1$ step 1 do in parallel
7: $r_2 \leftarrow \sqrt{(i_2 - N/2)^2 + (j_2 - N/2)^2}$
8: if $r_2 \leq a$ then
9: $W_{i_1,j_1,i_2,j_2} \leftarrow \text{ones}(N,N)$
10: else
11: for $j_2 = 0$ to $N - 1$ step 1 do
12: $r_2 \leftarrow \sqrt{(i_2 - N/2)^2 + (j_2 - N/2)^2}$
13: if $r_2 \leq N_a$ then
14: $W_{i_1,j_1,i_2,j_2} \leftarrow \text{ones}(N,N)$
15: return $W$

References

[1] E. Wolf, A macroscopic theory of interference and diffraction of light from finite sources, i.e. fields with a narrow spectral range. Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences 225 (1160) (1954) 96–111.

[2] E. Wolf, New theory of partial coherence in the space-frequency domain. part i: spectra and cross spectra of steady-state sources, J. Opt. Soc. Am. A 72 (3) (1982) 343–351.

[3] E. Wolf, A macroscopic theory of interference and diffraction of light from finite sources ii. fields with a spectral range of arbitrary width, Proc. R. Soc. London, Ser. A 230 (1181) (1955) 246–265.
[4] G. B. Parrent, On the propagation of mutual coherence, JOSA 49 (8) (1959) 787–793.
[5] E. Wolf, New theory of partial coherence in the space-frequency domain. part ii: Steady-state fields and higher-order correlations, JOSA A 3 (1) (1986) 76–85.
[6] M. Born, E. Wolf, Principles of Optics, 6th Edition, Cambridge University Press, 1998.
[7] K. Yoshimori, A. Hirai, T. Isoue, K. Itoh, Y. Ichioka, Effects of a linear edge on optical imaging, J. Opt. Soc. Am. A 12 (5) (1995) 981–990.
[8] R. Castañeda, F. Medina, Partially coherent imaging with schell-model beams, Optics & Laser Technology 29 (4) (1997) 165–170.
[9] T. C. Magalhães, J. M. Rebor Dhabi, Simulation of partially coherent light propagation using parallel computing devices, Proc. SPIE 10453 (2017) 104531U.
[10] E. Wolf, New spectral representation of random sources and of the partially coherent fields that they generate, Opt. Commun. 38 (1) (1981) 3–6.
[11] A. S. Ostrovsky, Coherent-mode representations in optics, Vol. 164, SPIE Press, 2006.
[12] A. Burvall, A. Smith, C. Dainty, Elementary functions: propagation of partially coherent light, J. Opt. Soc. Am. A 26 (7) (2009) 1721–1729.
[13] T. E. C. Magalhães, Spatial coherence mapping of structured astrophysical sources, Ph.D. thesis, Faculdade de Ciências da Universidade de Lisboa (2019).
[14] L. Mandel, E. Wolf, Optical Coherence and Quantum Optics, Cambridge University Press, 1995.
[15] J. W. Goodman, Statistical optics, John Wiley & Sons, 2015.
[16] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, A. Fasih, Pycuda and pyopencl: A scripting-based approach to gpu run-time code generation, Parallel Computing 38 (3) (2012) 157–174.
[17] D. R. Kacli, P. Mistry, D. Schaa, D. P. Zhang, Heterogeneous computing with OpenCL 2.0, Morgan Kaufmann, 2015.
[18] E. Wolf, Introduction to the theory of coherence and polarization of light, Cambridge University Press, 2007.
[19] F. Gori, M. Santarsiero, Devising genuine spatial correlation functions, Opt. Lett. 32 (24) (2007) 3531–3533.
[20] Z. Mei, O. Korotkova, Cosine-gaussian schell-model sources, Optics Letters 38 (14) (2013) 2578–2580.
[21] W. H. C. E. Wolf, Coherence and radiant intensity in planar wave fields generated by fluctuating primary planar sources, J. Opt. Soc. Am. 68 (7) (July 1978).
[22] J. W. Goodman, Introduction to Fourier optics, Roberts and Company Publishers, 2005.
[23] F. Gori, Far-zone approximation for partially coherent sources, Optics letters 30 (21) (2005) 2840–2842.
[24] K. R. Rao, D. N. Kim, J. J. Hwang, Fast Fourier transform-algorithms and applications, Springer Science & Business Media, 2011.
[25] J. W. Goodman, Introduction to fourier optics, Roberts & Company Publishers 3 (1968) 63–96.
[26] D. Le Bolloc’h, J.-F. Sadoc, X-ray coherent diffraction interpreted through the fractional fourier transform, The European Physical Journal B 81 (4) (2011) 481–487.
[27] T. E. C. Magalhães, J. M. Rebor Dhabi, Numerical simulations of spectral shifts in the far-field spectrum of light due to source correlations, J. Opt. Soc. Am. A 35 (3) (2018) 423–430.