PYPARAX - PYTHON MODULE FOR OPTICAL SETUP SIMULATION IN PARAXIAL APPROXIMATION

Victor-Cristian Palea[1] and Liliana Preda[2]

This module is intended as an optical system simulation tool for propagation of optical beams through optical systems using the paraxial wave equation. The module uses three types of optical elements (free space, phase mask, and amplitude mask) based on which various procedures are defined. Out of the implemented procedures we name forward and backward propagation, phase mask retrieval, alignment analysis, and plotting functions.

Keywords: Python module, paraxial optics, optical system design, phase modulation, Monte-Carlo

1. Introduction

Numerical solvers and methods for the propagation of optical beams cover a wide range of regimes, with commercial solutions from ray tracing software such as OpticStudio[1] to FDTD based solvers such as OptiFDTD[2]. This however does not exclude the potential need in some contexts for in-house software solutions that are optimized strictly for the type of problem that is investigated and the computing system in use, while also being cost-effective.

We have developed the PyParax[3] module emerged from our necessity to study the propagation related properties of specific optical profiles[4, 5, 6], and methods for synthesizing them using phase modulation. The topic of characterizing the propagation related properties from Palea et al.[4, 5, 6] required that two steps are to be satisfied, namely the construction of optical profiles from the theoretical results, and the propagation of these profiles in order to investigate their properties. The synthesis problem follows as a second topic because it consists of identifying the methods with which these optical beams can be obtained in the lab using phase modulation. This required the integration of the numerical solver[7] into a method through which basic optical systems can be considered when propagating the optical beam. The implementation of such a propagator function that can handle optical systems has enabled the use of the software for more than just solving the propagation

[1]Asistent, Faculty of Applied Sciences, University POLITEHNICA of Bucharest, Romania, e-mail: victor.palea@physics.pub.ro

[2]Lecturer, Faculty of Applied Sciences, University POLITEHNICA of Bucharest, Romania, e-mail: liliana.preda@physics.pub.ro

equation and computing phase masks. Since the user defines the optical system, the possibility of optimizing it is evident.

Another aspect that is related to the synthesis of optical profiles is the constraints that have to be satisfied. Some common algorithms that handle the phase retrieval problem and are used in phase modulation[8, 9] impose that the output desired profile has only the amplitude function defined by the user, the phase being of no real interest. One exception here is the Yang-Gu algorithm[10] which can be applied by defining the phase function instead of the amplitude, but still one cannot choose both. A solution to having both amplitude and phase to be defined by the user is to use two phase masks[11] but this is more costly since two spatial light modulators are required for experimental implementations.

This state of affairs, and the need of optical profiles with specific amplitude and phase functions, implied a different approach to the problem of profile synthesis. Since PyParax can handle the propagation through optical system, the solution to the phase mask computation problem consisted in optimizing the optical system in order to have a good reconstruction of the desired optical profile. Also, since optical systems are expected to be implemented in the lab and not just simulated, an option for including alignment tolerances has been included in order to check qualitatively how the output profile is affected by the positioning of the optical elements.

2. Software description

PyParax can be used as a Python 3.X module that can be accessed on Github[3] under GNU license. The installation is done by calling the **setup.py** file using

```
>> python3 setup.py install
```

in Terminal for Ubuntu, or CMD for Windows 10, from the directory where the module is placed. After installment it can be used as any other Python module, since its inherent structure relies on submodules and classes in order to split the different sections of the main module as it is mentioned below.

The numerical solver is based on a Fourier Transform method[7] and the optical elements are of three types: free space, phase masks and amplitude masks. Due to their frequent use lenses have been implemented as a standalone optical element that can be defined through its focal length, although technically it is still a phase mask. From our point of view this classification of the optical elements, although having only three classes of elements, is quite general for image formation systems, thus permitting a variety of optical systems to be simulated.

For more information regarding the implementation check the specific submodule based on the following subsection and the documentation.

3. **Module structure**

The structure of the PyParax module is given in figure 1. There are three submodules that contain computational parts within PyParax, the Function Generator (FG), the Numerical Solver (NS), and the Experimental Simulator (ES). The FG is a collection of frequently used functions e.g. common initial condition (Gaussian, Airy), lens phase masks, and amplitude masks for circular apertures. The NS handles the computation of the solution for free space propagation. The ES is the main submodule since it contains all the functions that are related to the propagation through optical systems, optimization based on some parameters of the lenses (if they exist within the optical system), plots, Monte-Carlo calculations, and system RAM requirements estimations.

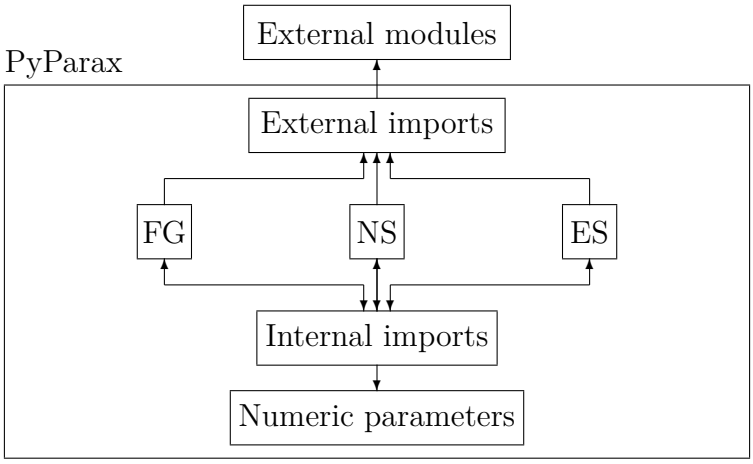


Fig. 1. The structure of the PyParax module. Arrows indicate the import logic between the submodules and with the external modules.

All the above mentioned submodules communicate via an internal imports submodule. They also import from a Numeric Parameters submodule the default values of various parameters such as the scale of the spatial domain, the number of points and step values on each axis, the wavelength, the refractive index of the medium and values for shifting the transverse axes. These values can be changed according to the requirements of the user.

In order to make use of existing optimized modules for linear algebra operations and data ploting, the three submodules FG, NS, and ES can import via an external imports submodule the required modules.

4. **Illustrative examples of functionality**

Based on the features of the module, several examples are presented, namely the propagation of a Gaussian beam through a misaligned telescope,

the computation of a phase mask for optical modulation of the output beam profile and its validation, and some Monte-Carlo of both previous cases.

4.1. Gaussian beam through an optical system

A basic optical system that can easily be implemented in PyParax consists of a telescope made of two lenses that are separated by a given distance. The propagation model used here is the 1-dimensional transverse space version, where the transverse axis is designated by an axis x and propagation along the z axis. The optical system consists of 5 optical elements presented in figure 2. The input Gaussian beam is propagated through free space of distance 100 units. The first lens in the system is shifted along the x axis by 0.5 units so before the lens phase mask is applied on the profile it is shifted for the corresponding distance. After this the resulting profile is propagated for another 150 units where it reaches the second lens which is shifted along the other transverse axis y by 1 unit. Since the propagation is done only with a 1-dimensional transverse space that considered only the x axis, this shift is not taken into account, so the phase mask of an aligned lens is applied. Next the profile is propagated for another 100 units where it reaches the end of the optical system marked as the output.

It should be noted that the implementation of the module sets as the length unit of measurement the *mm*, which has been used in all the examples that are to be presented in this article.

All the following code snippets start with the import of the FG and ES submodules done using

```
from pyparax import function_generator as FG
from pyparax import experimental_simulator as ES
```

so we will not repeat this step at each example.

The implementation of the optical system from figure 2 is given in the following code snippet:

```
system = [100, ["l", 50, 0.5,0], 150, ["l", 80, 0, 1],
    ↪ 100]
f0 = FG.standard_initial_conditions.generate_gauss_1d(0,
    ↪  0.2)
f = ES.experimental_simulator_1d.propagate(f0, system,
    ↪ print_output = True)
```

First we import the FG and ES submodules. An optical system is defined with a structure given by the list attributed to the variable **system**, which can be interpreted as having:

(1) 100 - free space of 100 units length;
(2) ["*l*", 50, 0.5, 0] - a lens with focal length of 50 units, and offsets of 0.5 units on the x axis and 0 units on the y axis;

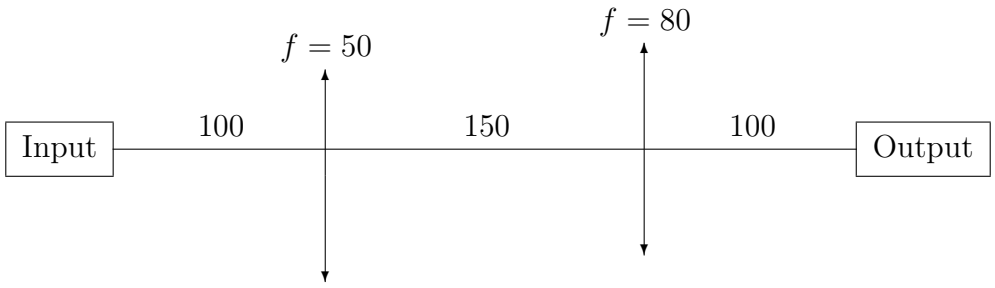


FIG. 2. Optical system schematic for a misaligned telescope.

(3) 150 - free space of 150 units
(4) ["l", 80, 0, 1] - a lens with focal length of 80 units, and offsets of 0 units on the x axis and 1 unit on the y axis;
(5) 100 - free space of 100 units.

Next the initial condition is assigned to the variable **f0**. We have used a Gaussian function of mean 0 units and standard deviation of 0.2 units. Finally the solution is computed and stored in the **f** variable. If the **print_output** variable is *True*, then a plot of the beam's amplitude profile is made at the end of the computation which is given in figure 3. The 2 vertical lines mark the existence of the lenses at the corresponding position in the optical system.

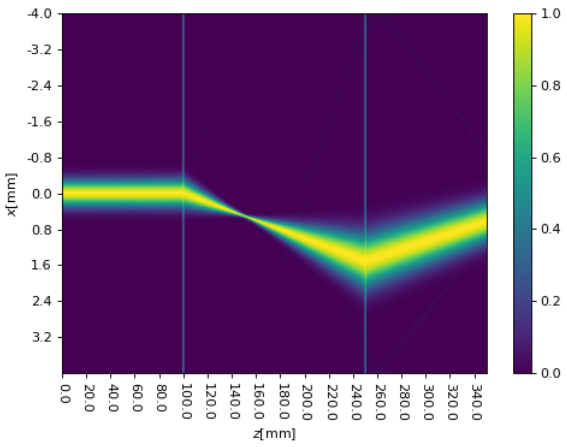


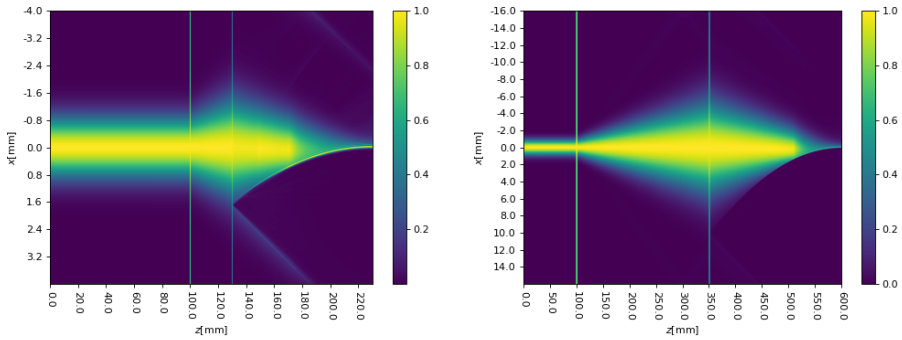
FIG. 3. Gaussian beam propagated through a misaligned telescope. The cyan vertical lines indicate the position of the lenses along the propagation axis.

An observation is required at this point. The amplitude functions from figure 3 and the following ones have been rescaled at each point along the propagatin axis such that the maximum amplitude is equal to 1. We consider this to help better visualize the beam when it is focused due to the difference in amplitude. This option can be toggled on or off via parameter **norm** which can be initialized directly from the **propagate** function's input parameters.

4.2. Single phase mask retrieval

The second example consists of an optical system that generates an Airy profile from a Gaussian using a phase mask. The method consists of using an initial system which is then optimized manually, which gives the cases from figure (4). For this task we use the following code snippet:

```
system1 = [100, ['l', -50, 0,0], 30]
system2 = [100]
f_in = FG.standard_initial_conditions.generate_gauss_1d
    ↪ (0, 0.6)
f_out = FG.standard_initial_conditions.generate_airy_1d
    ↪ (40, 0, 1)
mask = ES.mask_generator_1d.compute_mask_dual_system(
    ↪ system1, system2, f_in, f_out, check_mask = True,
    ↪ print_output = True)
```



(A) Initial (B) Optimized

FIG. 4. Phase modulated Gaussian beam to generate an Airy beam using an initial system which is then optimized.

In order to compute the phase mask the optical system is split in two parts, namely **system1** which contains all the elements upto the phase mask to be computed, and **system2** which contains the elements after the phase mask. The syntax here is similar as in the previous example with the observation that a negative focal length corresponds to a divergent lens. The input beam profile is given by **f_in**, initialized as a Gaussian beam, and the desired beam profile is **f_out** which in this case is an Airy function. The parameters of the Airy function generator are, in this order, **scale** along the transverse axis, **offset** of the main lobe, and **decay** by considering an exponential function that is multiplied with the original function in order to truncate its amplitude profile.

The phase mask computation is handled by a built in function implemented in **mask_generator_1d**, namely **compute_mask_dual_system**.

This is done by a forward-backward propagation method[12] with the resulting phase mask being saved in the **mask** variable. Since not all optical systems are optimal for a given input and desired output profiles, the quality of the retrieval is given by the maximum of the convolution between the desired output profile and the one generated with the retrieved phase mask. This analysis is toggled on via parameter **check_mask** which gives the similarity of approximately 91%. The similarity value can be improved to approximately 98.5% by extending the transverse domain from the default 2000 to 8000 points and changing the subsystems to

```
system1 = [100, ['l', -50, 0,0], 250]
system2 = [250]
```

4.3. Monte-Carlo tolerance test

Another PyParax functionality is related to alignment tolerance analysis. The investigated case here is given by an optical system consisting of two lenses out of which one has a possible alignment error of ± 0.5 units perpendicular to the propagation axis.

```
system = [100, ['l', 50, 0.5,0], 150, ['l', 30, 0,1],
    ↪ 100]
system_errors = [0,['l', 0, 0,0], 0, ['l', 0, 0.5, 0],
    ↪ 0]

f0 = FG.standard_initial_conditions.generate_gauss_1d(0,
    ↪  0.2)
f = ES.experimental_simulator_1d.
    ↪ monte_carlo_precision_test(f0, system,
    ↪ system_errors, 20)
```

Unlike the previous examples a variable **system_errors** with identical structure as **system** is initialized. It contains the tolerance for the numerical parameters that are used to define each element in **system**, which implies that the second lens's position can be sampled from the interval $[-0.5, 0.5]$ units.

Based on this approach, the function **monte_carlo_precision_test** propagates the initial condition **f0** for 20 iterations, each time generating an optical system with numerical values sampled from their corresponding tolerance intervals. The amplitude profile of the resulting beams are added to a variable that is returned to initialize **f**.

The plot of **f** is given in figure 5, where as expected, the shift of the second lens changes the position of the spot where the beam focuses on the x-axis, thus showcasing the Monte-Carlo function of PyParax.

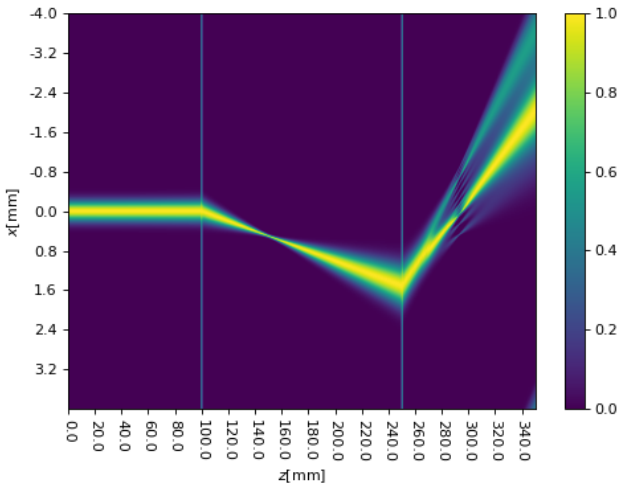


FIG. 5. Monte-Carlo test of the telescope system.

4.4. **Spiral phase plate**

For this last example we consider the case of investigating the synthesis of optical vortices using a spiral wave plate where we are interested in how the optical vortex propagates when the system is aligned perfectly (tabel 1), how does a misalignment affect the shape of the optical vortex amplitude profile (tabel 2), and how does the standard deviation of the input Gaussian beam account for a misalignment (tabel 3).

z[unit]		20	40	60	80	100	
Amp.							$m = 2$
ϕ							
Amp.							$m = 3$
ϕ							
Amp.							$m = 4$
ϕ							

TABLE 1. Amplitude and phase functions of orbital vortices of orders 2, 3 and 4 during propagation. The phase function is given as a product between the phase and the amplitude functions.

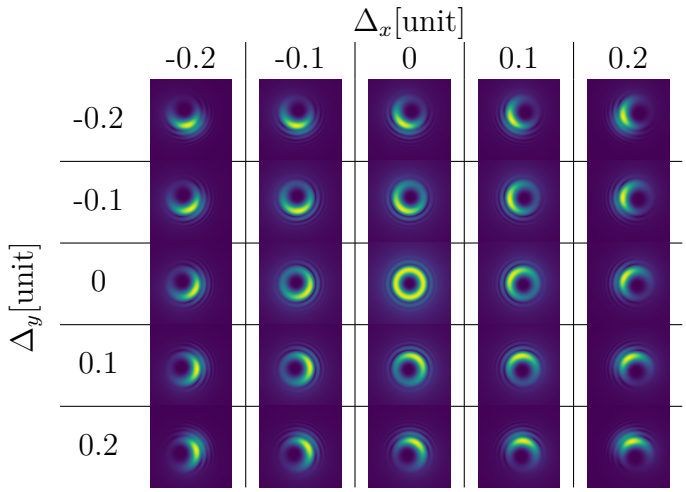


TABLE 2. Amplitude analysis of misalignment of the spiral phase mask for $m = 4$ along both transverse axes.

σ[unit]	0.1	0.2	0.3	0.4	0.5
f_0					
$\Delta_x = \Delta_y = 0[unit]$					
$\Delta_x = \Delta_y = 0.1[unit]$					
$\Delta_x = \Delta_y = 0.2[unit]$					

TABLE 3. Amplitude of optical vortices for different phase mask alignments and spreads of the input Gaussian.

The results suggest that although a perfect alignment is desirable, having a wider input Gaussian can compensate the existence of misalignments.

Optical vortices[13] can be generated by phase modulating a Gaussian beam using a phase mask ϕ defined analytically by

$$\phi = \exp(i2\pi m\theta) \tag{1}$$

where θ is the angle in the polar representation and m is the order of optical vortex. The optical system for synthesis implemented with PyParax is

```
system = [10, ['mp', mask, 0 ,0], z]
```

where **z** is a variable that is varied and **mask** is the phase computed with equation (1). The cases that we have computed are shown in table 1. Additionally for **z** = 100 units and $m = 4$ other simulations have been carried out

in table 2 in order to check the result in amplitude for having the phase plate misaligned on the transverse axes by upto ± 0.2 units, and how the spread of the input Gaussian beam can compensate the misalignment in table 3.

5. Conclusion

We have developed the PyParax module as a basic tool for simulating the propagation of optical profiles through systems made of amplitude and phase masks separated by free space. This allows for prototyping optical setups in the context of beam or image synthesis, testing its endurance to misalignment, and compute masks for phase modulation.

REFERENCES

[1] https://www.zemax.com/pages/opticstudio Retrieved: July 2021
[2] https://optiwave.com/optifdtd-overview/ Retrieved: July 2021
[3] https://github.com/victorcristianpalea/PyParax
[4] *V.-C. Palea, L.A. Preda*, The control of intensity peak dynamics for paraxial waves. Journal of Engineering Mathematics, 2019 Apr;115(1):89-98.
[5] *V.-C. Palea, L.A. Preda*, Construction of finite non-diffractive and self-accelerating laser beams, Mathematical Methods in the Applied Sciences, 2021 May 12.
[6] *V.-C. Palea, L.A. Preda*, Iisotimic curves for description and control of intensity profile dynamics of solutions to the paraxial wave equation, University POLITEHNICA of Bucharest, Scientific Bulletin-Series A-Applied Mathematics and Physics, 2019 Jan 1;81(2):287-296.
[7] *A. Couairon, E. Brambilla, T. Corti, D. Majus, O.D. Ramírez-Góngora, M. Kolesik*, Practitioner's guide to laser pulse propagation models and simulation, The European Physical Journal Special Topics, 2011 Nov;199(1):5-76.
[8] *R.W. Gerchberg, W.O. Saxton*, A practical algorithm for the determination of the phase from image and diffraction plane pictures, Optik, 1972, 35 237–46
[9] *R.G. Dorsch, A.W. Lohmann, S. Sinzinger*, Fresnel ping-pong algorithm for two-plane computer-generated hologram display, Applied optics, 1994 Feb 10;33(5):869-75.
[10] *G.Z. Yang, B.Z. Dong, B.Y. Gu, J.Y. Zhuang, O.K. Ersoy*, Gerchberg–Saxton and Yang–Gu algorithms for phase retrieval in a nonunitary transform system: a comparison, Applied optics. 1994 Jan 10;33(2):209-18.
[11] *L. Zhu, J. Wang*, Arbitrary manipulation of spatial amplitude and phase using phase-only spatial light modulators. Scientific reports. 2014 Dec 11;4(1):1-7.
[12] *P.R. Stepanishen, K.C. Benjamin*, Forward and backward projection of acoustic fields using FFT methods, The Journal of the Acoustical Society of America, 1982 Apr;71(4):803-12.
[13] *L. Allen, M.W. Beijersbergen, R.J. Spreeuw, J.P. Woerdman*, Orbital angular momen-tum of light and the transformation of Laguerre-Gaussian laser modes, Physical review 1992 Jun 1;45(11):8185.