UNIVERSITÉ DE LORRAINE | UFR SCIENCES FONDAMENTALES ET APPLIQUÉES

Beampy

User guide

Online documentation

Jonathan Peltier

PAIP Master, Lorraine University

January 14, 2020

Acknowledge

I want to thanks Pr. Nicolas Fressengeas - responsible of the master, director of the Laboratory MOPS and teacher in the course where this program were developed - for proposing interesting projects in his classes and to let us freely pursue our goals.

I also want to thanks Marcel Soubkovsky for initializing the interface in its early stage and consequently decreases the needed time to fully developed it.

Abstract

This software is based on Python 3.7.4. and has been tested on Spyder 3.7. It uses Beam Propagation Method (BPM) to compute beams propagation into a linear or non-linear medium.

It allows to choose different shape for the guides and beams: flat-top, Gaussian or even mode-based for the beam.

The beams can be propagate into free space, an array of guides or into a system of two curved guides and a straight central one.

In addition, a kerr effect can be introduce to take into account the refractive index variation with the beam intensity.

It is also possible to add an area where the beam is absorbed through looses.

This software is composed of 3 main python files: *bpm.py* that contain the Bpm class and thus, all the needed methods to compute the propagation, *interface.py* which is created using *Qt Designer* and *user_interface.py* that connect the Bpm methods to the interface and plot the outputs such as the power through the propagation.

Contents

1 bpm module

This section describe the essentials methods of the class Bpm used in Beampy and their relation to the interface.

1.1 Bpm class summary

```
1  bpm = Bpm(width, no, delta_no,
2            length_z, dist_z, nbr_z_disp,
3            length_x, dist_x)
4  [length_z, nbr_z, nbr_z_disp, length_x, nbr_x, x] = bpm.create_x_z()
5
6  shape = bpm.squared_guide()
7  shape = bpm.gauss_guide()
8  [peaks, dn] = bpm.create_guides(shape, nbr_p, p, offset_guide=offset_guide)
9  [peaks, dn] = bpm.create_curved_guides(shape, curve, half_delay, distance_factor
       , offset_guide=offset_guide)
10
11 field_i = bpm.gauss_light(fwhm_i, offset_light=offset_light_i)
12 field_j = bpm.squared_light(fwhm_j, offset_light=offset_light_j)
13 [field_k, h, gamma, beta] = bpm.mode_light(mode, lo, offset_light=offset_beam_k)
14 [field_l, h, gamma, beta] = bpm.all_modes(lo, offset_light=offset_light_l)
15
16 field = [field_i, field_j, field_k, field_l]
17 [progress_pow] = bpm.init_field(field, theta_ext, irrad, lo)
18
19 [lost_beg, lost_end] = bpm.losses_position(guide_lost, width_lost)
20
21 [progress_pow] = bpm.main_compute(chi3=chi3, kerr=kerr, kerr_loop=kerr_loop,
       variance_check=variance_check, alpha=alpha, lost_beg=lost_beg, lost_end=
       lost_end)
```

Code Listing 1: Bpm class called by the interface

The *user_interface.py* file call the *Bpm* class from *bpm.py* to compute the beams propagation and follows the above code listing 1.

The list bellow summarizes those methods but, a thorough explanation can be found right afterwards.

- (lines 1-3) The first action consists of initializing the *Bpm* class.
- (line 4) The windows variables are created with the *create_x_z* method, which return the corrected values of the variables: *length_z, nbr_z_disp, length_x* due to the sampling and return the new variables: *nbr_z, nbr_x, x*.
- The guide shape are defined by the *shape* variable, which is a lambda function.

  (line 6) The *bpm.squared_guide()* method return a squared modulation of the refractive index from -$\frac{1}{2}$*width* to $\frac{1}{2}$*width* center on 0.

  (line 7) The *bpm.gauss_guide(P)* method return a P order super-Gaussian modulation of the refractive index center on 0 with a waist=$\frac{1}{2}$*width* at 1/e.
- The guides are defined by one of the following methods:

  (line 8) *bpm.create_guides* which consist of an array of guides.

  (line 9) *bpm.create_curved_guides* which consist of two curved guides and a linear one.

- The beam can be:

  (line 11) Gaussian with *bpm.gauss_light*

  (line 12) flat-top with *bpm.squared_light*

  (line 13) based on a possible mode for a squared guide with *bpm.mode_light*.

  (line 14) based on all possibles modes with *bpm.all_modes*.

- (line 18) The *bpm.init_field* combines the fields if several were given (line 17). It put the V/m unity to the fields using the *irrad* intensity in W/m^2. It also multiply the field by the initial phase, depending on the exterior angle, define from the normal to the interface air/cladding. It create the free space matrix *phase_mat* describing the propagation of the beam without refractive index modulation. It also create the refractive index modulation *nl_mat*. Finally, it return the initial power that will be used inside *Bpm* to compute the propagation.

- (line 20) *bpm.losses_position* can be used to define an area or areas (by changing the code, not from the interface) where looses occurs.

- (line 22) *bpm.main_compute* is the final method that realized the propagation and return the power over x and z. It calls the *bpm_compute* for the BPM method, calls *absorption* to add the looses and calls *kerr_effect* to take non-linearity into account.

1.2 Windows variables

The x variable is defined as the array: [-$\frac{1}{2}$*length_x*,.,0,.,$\frac{1}{2}$*length_x* - *dist_x*] with *nbr_x* a even number of points and *dist_x* step between each points. This choice decrease the computation time by having a even number of points but introduce a asymmetry between negative and positive positions and the *length_z* variable correspond actually to *length_x* - *dist_x*. Note that this method changes the *length_x* variables to match a even number of point for the exact *dist_x* value.

Inside the *Bpm* class, the *nbr_z_disp* correspond to the number of points over z, including the first point define by the initial field. However, in the *MainApp* class of the *user_interface.py* file, *nbr_z_disp* is the number of point compute, without the initial point. So, in this class, one is added to its value when it's necessary to manipulate the real number of points over z. The reason behind this difference is that the *MainApp* class interact with the interface. And, the input value in the interface correspond to the number of point compute and not the real number of points over z. So, if Bpm returns the value+1, the interface will add 1 indefinitely for each computation.

To recap, the *dist_x*, *dist_z* variables are uses to redefine the lengths and number of points variables. Then, the new values are displayed on the interface.

1.3 Guides

The *bpm.gauss_guide(P)(x)* method return the normalized refractive index at the position x centered on zero for a P order super-Gaussian modulation. The guide width is defined such as $\Delta n = 1/e$ for $w_0 = \frac{1}{2}width$ (figure 1).

$$E(0) = exp\left(-(\frac{x}{w_0})^{2P}\right) \tag{1}$$

With P the order of the super-Gaussian beam, 1 correspond to a regular gaussian beam and 4 correspond to the usual super-Gaussian beam.

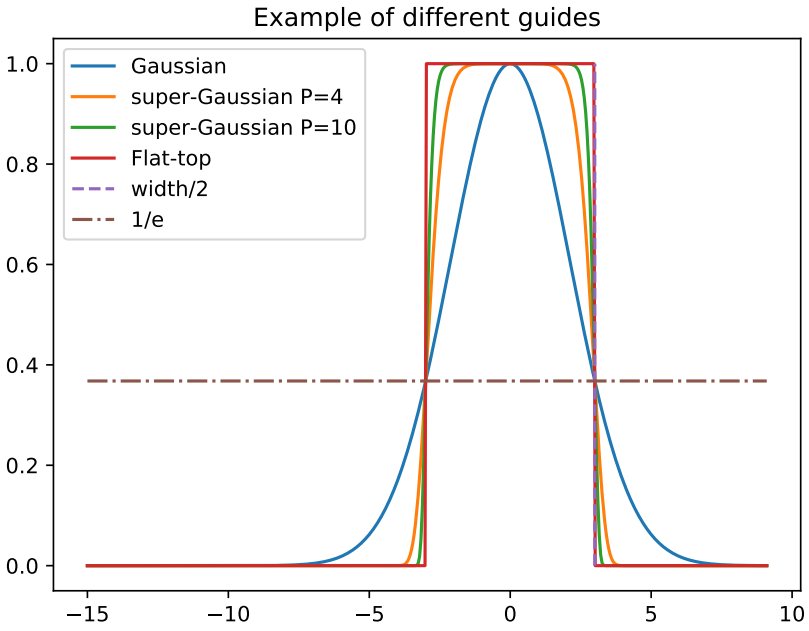


Figure 1: Waist of the guide defined as the width/2 at 1/e. From *examples.example_guides_x()*

Exemple: *bpm.gauss_guide(4)(0)*=1 and *bpm.gauss_guide(4)(width/2)*=e^{-1}

The 1/e convention has been chosen, for the guides, because this value allows to have the same width definition for squared, Gaussian or super-Gaussian guides.

The super-Gaussian guide at the order P=4 is preferred when using BPM because its shape is closed to the shapes obtained in laboratory with lithography and also because this shape don't have the numerical problems caused by the flat-top guide.

Thus, *bpm.squared_guide()(x)* must be used carefully because the BPM approximations implied smooth index variations to be valid.

Once the shape has been chosen, the user must define the z dependency of the guides between the two available methods: *bpm.create_guides* and *bpm.create_curved_guides* methods.

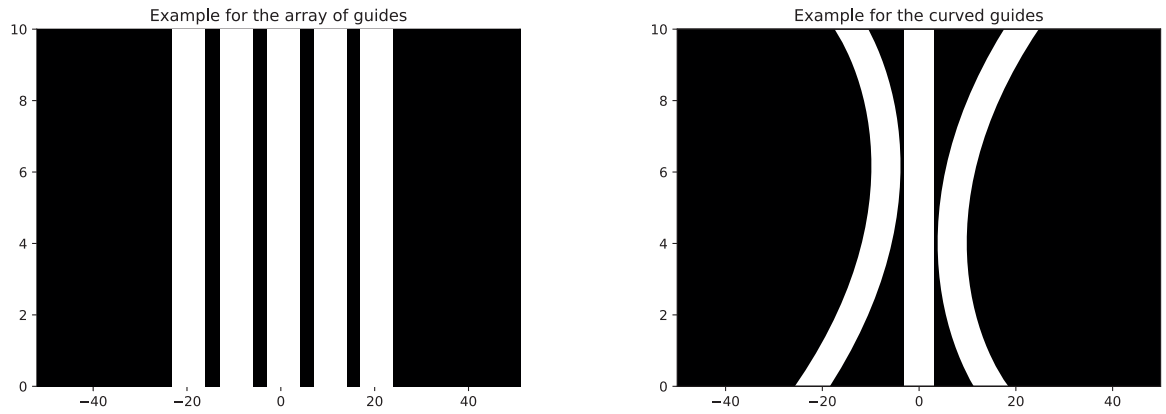
The former return an array of guides with a given shape and the latter return two curved guides with a central linear guide between. The figure 2 represent the two possibles configurations.

Noes that the offset is done by rolling the array and not by computing the $x - x_0$ values. This allow to compute one guide and to clone it for any offset needed. But, this choice forced the guide to exist inside the windows and to have the left values coming from the right side if a offset is set. However, this definition allows to reduce drastically the compute time needed for the curved array and for, in less measure, the array of guide. Indeed, the old definition (compute Gauss guides for every z with the proper offset) takes 16s for a given configuration, and drop to 1.6s with the roll definition.

1.4 Beams

1.4.1 Gaussian

The *bpm.gauss_light(fwhm, offset_light=*x_0*)* method return a Gaussian beam with an x_0 offset from the center with a full width at half maximum (fwhm) define for the intensity ($|E^2|$) such as w_0 = fwhm / $\sqrt{2 * ln(2)}$ at 1/e for the field and $1/e^2$ for the intensity (figure 3).



(a) Array of guides (b) Curved guides

Figure 2: The two configuration possible for guides. From *examples.example_guides_z()*

$$E(0) = exp\left(-(\frac{x - x_0}{w_0})^2\right) \tag{2}$$

Noes that, like the guide, the offset is done by rolling the array and not by computing the $x - x_0$ value. This allow to compute one beam and to clone it for any offset needed. But, this choice forced the beam to exist inside the windows and to have the left values coming from the right side if a offset is set. The good side is that the beam can be positioned relatively to the guide even when the guide exceed the windows and come back in the other side. Such cases should not exist but at least it is taken into account.

1.4.2 Mode based

The *bpm.mode_light()* compute a beam based on the chosen propagation mode for a planar waveguide (if it exists). If the chosen guide is Gaussian, the mode will not be adapted because the theory is based on a planar waveguide.

The theory of guided optic ([1] p9) give us the mode shape define as below.

For the m mode in the core $\forall x \in \left[-\frac{W}{2}, \frac{W}{2}\right]$:

$$E(x) = cos(h_m x) \text{ even modes} \tag{3}$$
$$E(x) = sin(h_m x) \text{ odd modes} \tag{4}$$

For the m mode in the cladding $\forall x \notin \left[-\frac{W}{2}, \frac{W}{2}\right]$:

$$E(x) = cos\left(h_m \frac{W}{2}\right) exp\left(-\gamma_m |x - \frac{W}{2}|\right) \text{ even modes} \tag{5}$$
$$E(x) = -sin\left(h_m \frac{W}{2}\right) exp\left(\gamma_m (x + \frac{W}{2})\right) \text{ odd modes x<-W/2} \tag{6}$$
$$E(x) = sin\left(h_m \frac{W}{2}\right) exp\left(-\gamma_m (x - \frac{W}{2})\right) \text{ odd modes x>W/2} \tag{7}$$

The *mode_determ* method resolves a transcendental equation which allows to determine all the constants.

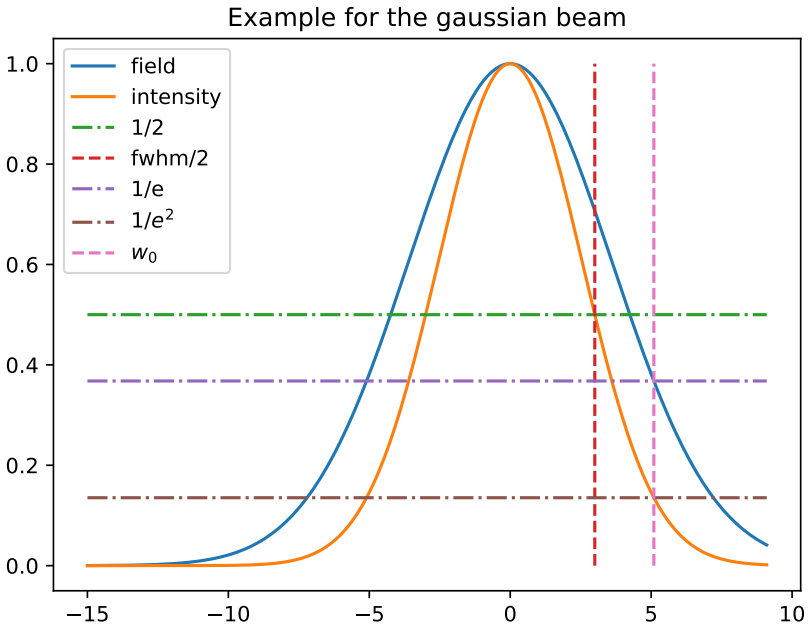


Figure 3: fwhm of the beam defined for the intensity, w_0 at 1/e for the field and at $1/e^2$ for the intensity. From *examples.example_gaussian_beam()*.

1.5 Compute

The *main_compute* method take into account the refractive index modulation, the free propagation over dz and eventual looses or Kerr effects.

After having tested in great detail the best decomposition to implement the BPM principle (see the function *examples.example_stability()* and also *examples.example_kerr()*), the following one has been selected.

Each step dz=*dist_z* is decomposed into three intermediate steps:

- A free space propagation over dz/2
- The index modulation perturbation over dz (lens)
- A free space propagation over dz/2

This decomposition is the most stable one compared to the decomposition dz+lens or lens+dz. When using the dz+lens notation, "+" must be interpreted not as a addition but as the order of operations.

One other idea was to conserve the dz/2+lens+dz/2 decomposition but only for the last field by doing dz/2 + loop over lens+dz then final lens+dz/2. But it has been abandon because each intermediate field would have been approximated by dz/2 and also because if a Kerr effect is present,the loop must come back to lens+dz/2 due to the corrective loop implemented. The algorithm is about 1.3 times slower than the former method but this choice increases the results precision.

1.6 Kerr

The Kerr effect used in Beampy is the optical Kerr (AC Kerr) and not the Kerr electro-optic effect (DC Kerr).

This effect appeared in χ^3 materials such as Niobate Lithium or any non-symmetrical materials. And become significant when a powerful optical field go through the material.

The optical beam changes the refractive index of the material:

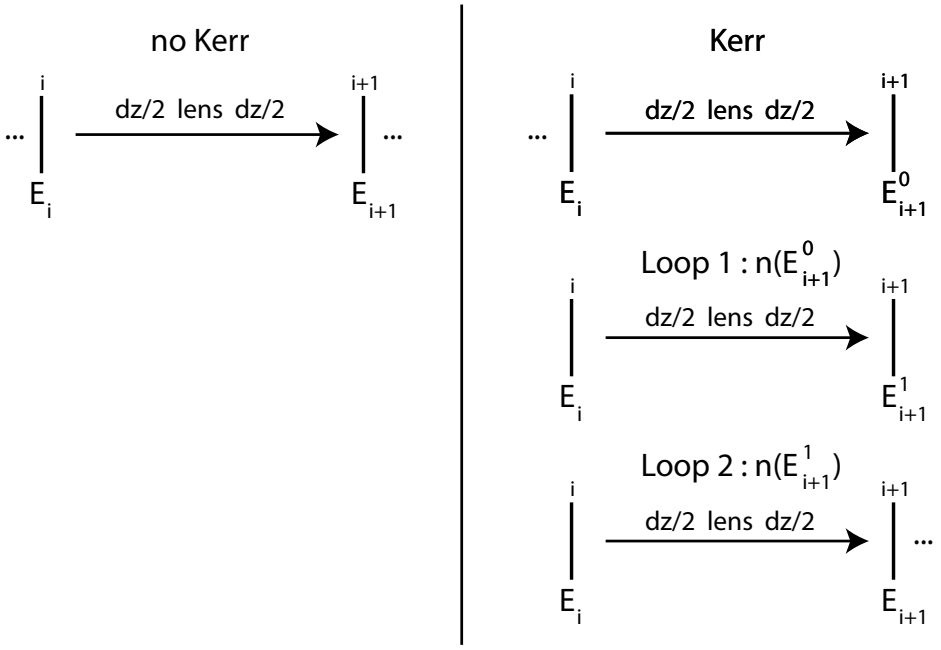


Figure 4: Kerr effect implementation. Inspired by optiwave.com

$$n = n_0 + \frac{3\chi^{(3)}}{8n_0}|E|^2 = n_0 + n_2 I \tag{8}$$

With n_0 the initial refractive index, χ^3 the third term of the electric susceptibility tensor and E the beam field.

Like mention is the compute section, several methods were consider to obtain the more stable results. The results can be found in the example section of the online documentation or by executing the *examples.example_kerr()* function.

The stable solution consist of computing the propagation following the dz/2+lens+dz/2 decomposition.

A simple implementation of the Kerr effect can be describe as follow: dz/2+dn(E)+lens+dz/2.

The problem with this solution is that the refractive index is changed by the previous field or by a half propagated field.

The Beampy implementation is described on the figure 4.

A first step propagation is done without the Kerr effect, then the resulting field is used to calculate the new refractive index, then the propagation step is done again with this new refractive index.

The resulting field could be directly used but, in some cases (high power), the result can be improved by using loops to have a converging solution (figure 5).

1.7 looses

The *absorption* method applies a loss over x from *lost_beg* to *lost_end* at a given z position for a given field. Those variables can be multidimensional to have several areas. And, the *losses_position* method return the area define relatively to guides so, the loss follows the guide over z in the case of curved guides.

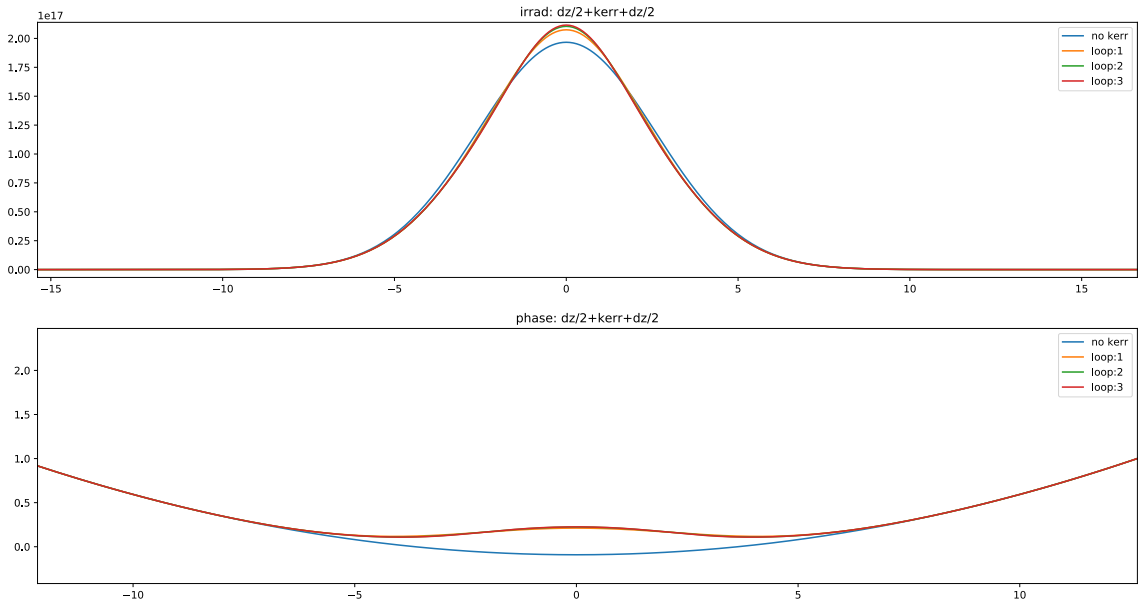


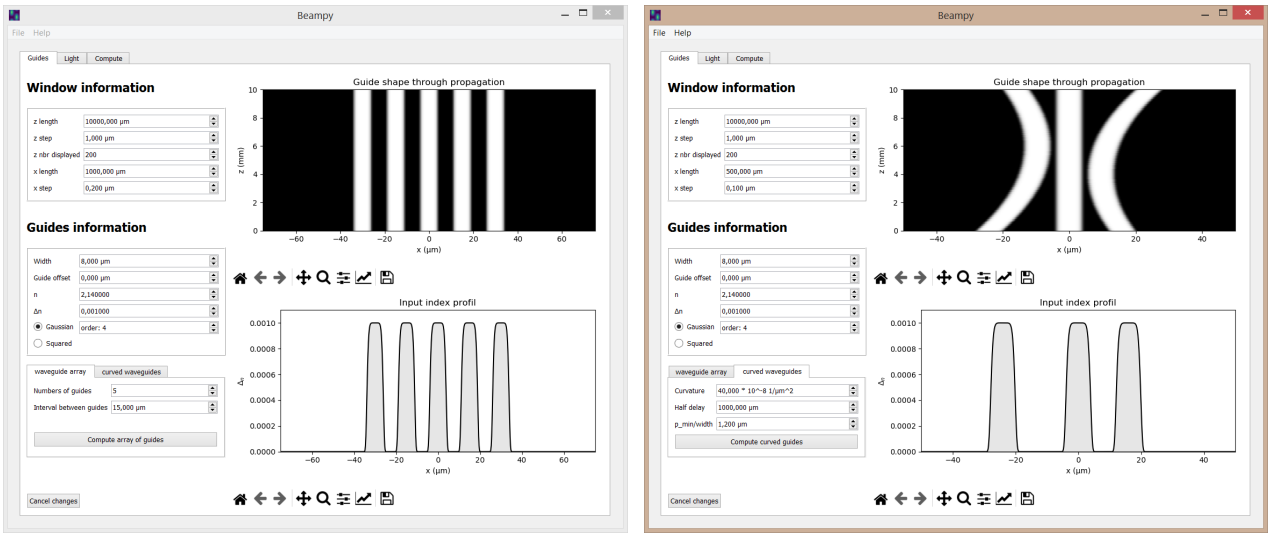
Figure 5: Impact of loops for the Kerr effect

2 User interface module

2.1 UserInterface class summary

```
calculate_guide(topology='array')
calculate_light()
calculate_propagation()
addmpl(tab='guide', pow_index=0)
rmmpl(tab, pow_index=0)
save_guide()
get_guide()
save_light()
get_light()
save_compute()
get_compute()
on_click_array()
on_click_curved()
on_click_light()
on_click_compute()
on_click_create_light()
on_click_delete_light()
open_file_name()
open_file(filename)
save_quick()
save_file_name()
save_file()
```

Code Listing 2: MainApp methods from user_interface.py



(a) Array of guides (b) Curved guide

Figure 6: Different guide configuration

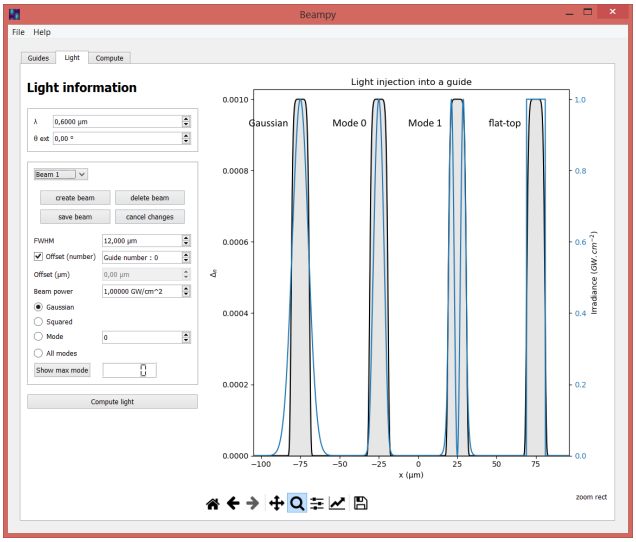
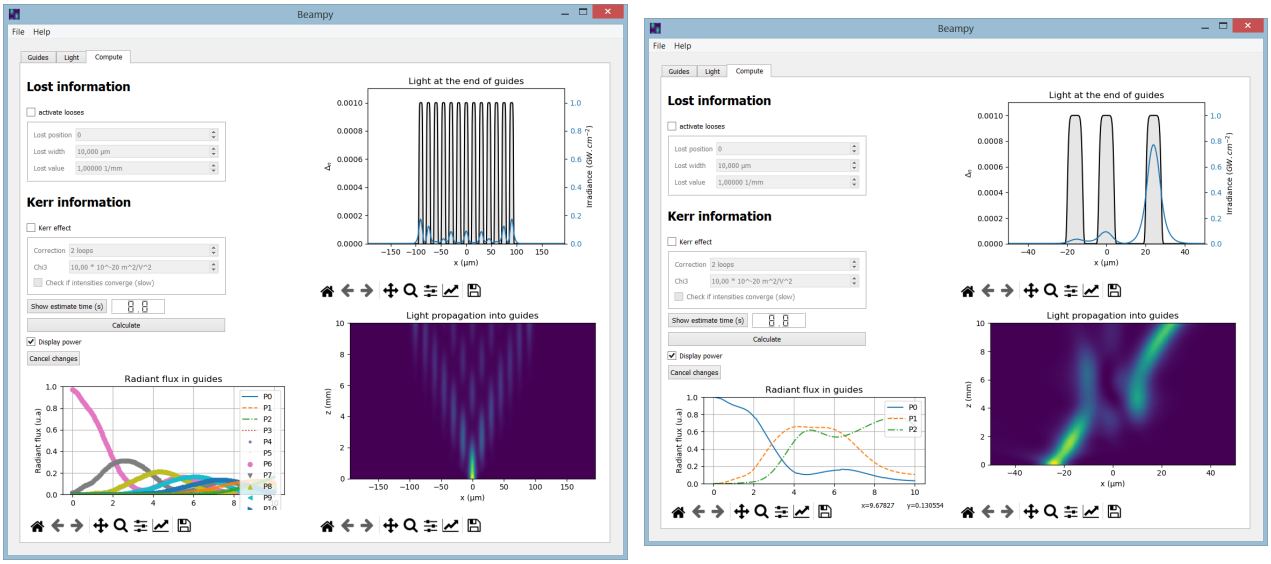


Figure 7: Compute interface



(a) Array of guides

(b) Curved guide

Figure 8: Compute interface

2.2 calculate guide

2.3 calculate light

2.4 calculate propagation

2.5 Add plot

2.6 Open and save file

2.7 Create beam

3 Validation

This section contain the comparison of Beampy's results to known results.

3.1 Free space

This demonstration is done using the *example_free_propag()* function of the *Beampy.examples* module.

The free propagation of a Gaussian beam increases its waist w according to:

$$w(z) = w_0 \sqrt{1 + \left(\frac{z}{z_0}\right)^2} \tag{9}$$

With $w(z) = \frac{\mathrm{FWHM}(z)}{\sqrt{2\ln 2}}$ and z_0 the Rayleigh length:

$$z_0 = \frac{\pi w_0^2 n}{\lambda} \tag{10}$$

For n=1, λ=1.5 μm, z=10 000 μm, x=1000 μm, dx=0.1 μm and fwhm=20 μm:
Beampy return a final fwhm=281.47 μm.
The theory returns a fwhm=281.60 μm.
Beampy have a relative error of 0.05 % compared to the theory.

If we take smaller x step, we can get a better results, for example with dx=0.01 μm:
Beampy return a final fwhm=281.601 μm.
The theory returns a fwhm=281.599 μm.
Beampy have a relative error of 0.0009 % compared to the theory.

We can conclude from this result that the free propagation include in Beampy is correct when using small x step but can give false results if the step is too large.

3.2 Coupled mode theory

From one of my internship report [2].

Two guides:

$$I_l(z) = I_0\, cos^2(Cz) \tag{11}$$

$$I_r(z) = I_0\, sin^2(Cz) \tag{12}$$

N guides:

$$a_n(z) = i^n\, a_0\, J_n(2Cz) \tag{13}$$

With C:

$$C = \frac{2h^2\gamma^2 e^{-\gamma s}}{\beta(W\gamma + 2)(h^2 + \gamma^2)} \tag{14}$$

We can observed on the figure 9b that the light in second guide has a first peak at 0.31. The First order Bessel function has a first peak at 0.58, so the squared Bessel function has a first peak at $0.58^2 = 0.34$ (figure 10).

Knowing the approximations (the power measurement is always under estimate between guide due to the recovering of lights, the Gaussian beam is not well adapted to have a perfect injection), this results can be consider good enough.

3.3 Kerr effect

4 Approximations

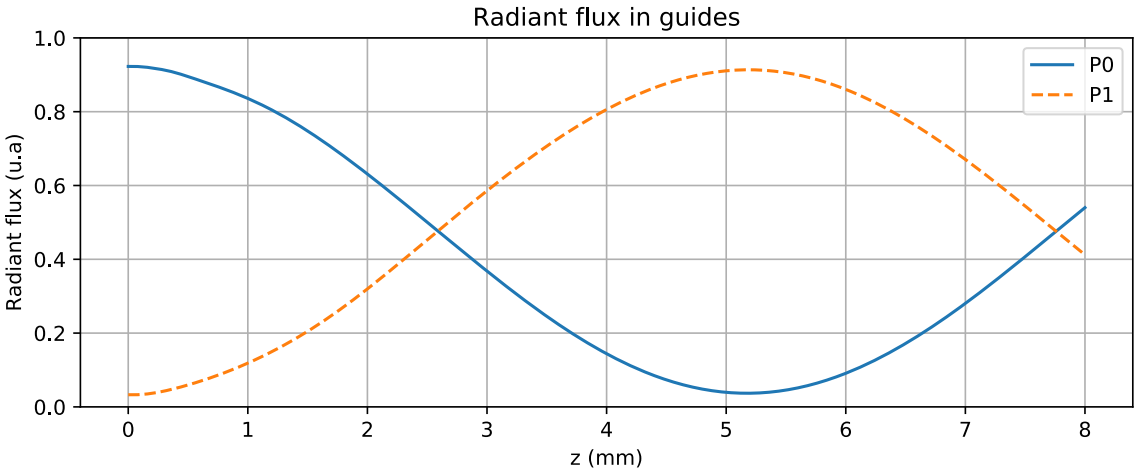
4.1 In Beampy

– The x length is define with a dist_x less than expected to have a even number (see 1.2).

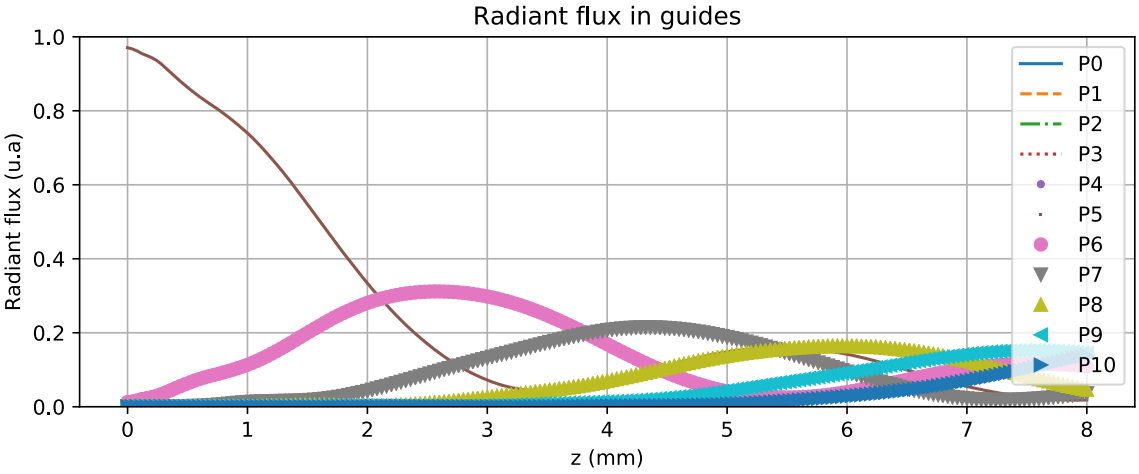
4.2 Not in Beampy

– Changing the code to have dz/2, loop[lens, dz], then final lens and dz/2 instead of our choice loop[dz/2, lens, dz/2] reduces the computation time by 1.3. The irradiance power become define for z = 0, dz+dz/2, 2*dz+dz/2, n*dz instead of z = 0, dz, 2*dz, n*dz. The last loop of the propagation is done over dz/2 to have a correct last value. So, with this approximation, the first and last power will be correct but all the powers between propagates over a extra dz/2.

– The kerr effect could be speed up drastically using the dz+lens choice because the dn variation depend in the beam intensity thus, independent of the phase. So no corrective loops should be used. However, in some extreme cases, this approximation diverge from the stable solution.



(a) Two guides



(b) 11 guides

Figure 9: Power in guides

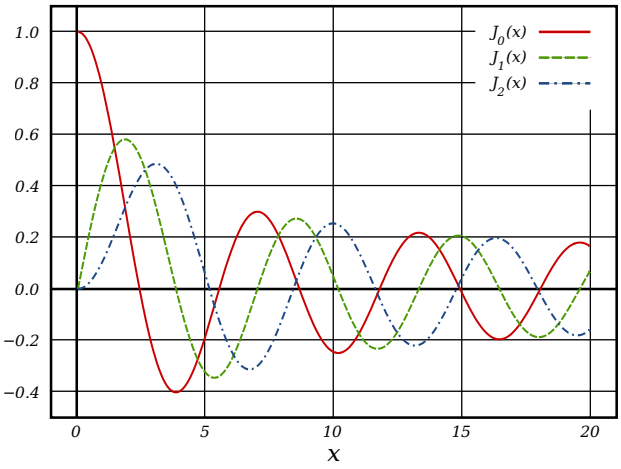


Figure 10: Bessel function. Source: wikipedia Bessel function

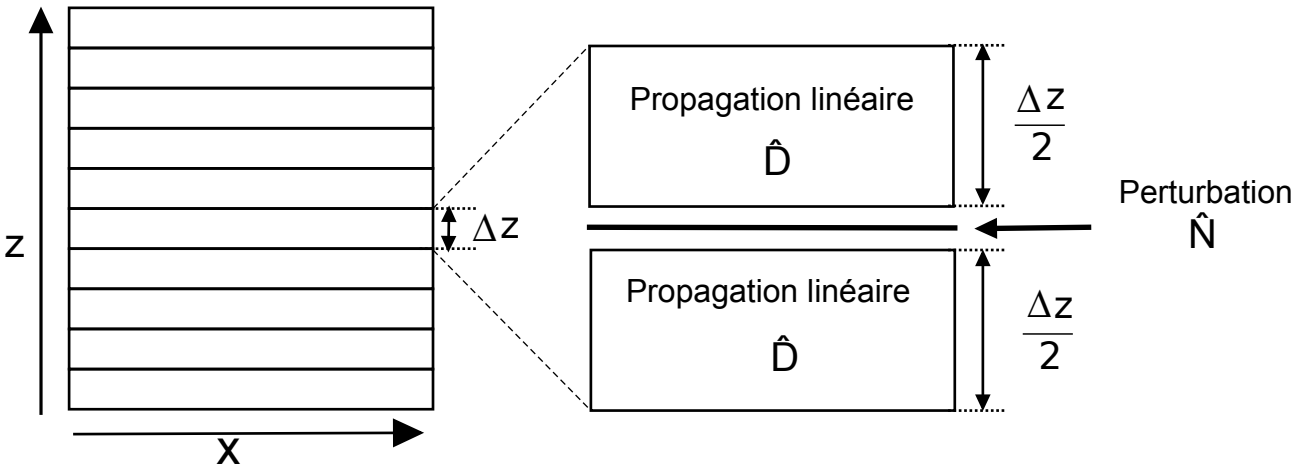


Figure 11: BPM principle [1]

5 Beam Propagation Method (BPM)

The Beam Propagation Method (p165 [1]) is a method that resolve the Helmoltz equation by using perturbations as a phase correction. The optical field propagation is decomposed into a free space propagation and a phase variation due to the perturbation. The perturbation can be the refractive index modulation (guides) or the Kerr effect (creation of guides).

This method necessitate some approximations:

- paraxial, beam with small angle
- small refractive index variation, smooth slope for the edges of guides
- small refractive index variation, core refractive index close to the cladding one
- slowly varying envelope, slow spacial variations compare to the wavelength and time variations.

The following demonstration is mostly based on Nicolas Fressengeas's course: "The Beam Propagation Method"

Solution to the Helmoltz equation:

$$E(z+dz) = exp\left(-idz\sqrt{\Delta_\perp + \frac{\omega^2}{c^2}n^2}\right)E(z) \tag{15}$$

slowly varying envelope approximation $\Delta_\perp \gg \frac{\omega}{c}n$:

$$\sqrt{\Delta_\perp + \frac{\omega^2}{c^2}n^2} \approx \frac{\Delta_\perp}{\sqrt{\Delta_\perp + \frac{\omega^2}{c^2}n^2} + \frac{\omega}{c}n} + \frac{\omega}{c}n \tag{16}$$

Small index variation approximation:

$$\sqrt{\Delta_\perp + \frac{\omega^2}{c^2}n^2} \approx \frac{\Delta_\perp}{\sqrt{\Delta_\perp + \frac{\omega^2}{c^2}n_0^2} + \frac{\omega}{c}n_0} + \frac{\omega}{c}n = \frac{\Delta_\perp}{\sqrt{\Delta_\perp + k^2} + k} + k\frac{n}{n_0} \tag{17}$$

Thus,

$$E(z+dz) = exp\left(-idz\left[\frac{\Delta_\perp}{\sqrt{\Delta_\perp + k^2} + k} + k(\frac{n}{n_0} - 1)\right]\right)E(z) \tag{18}$$

With the linear and free propagation operator:

$$\widehat{D} = \frac{\Delta_\perp}{\sqrt{\Delta_\perp + k^2} + k} \tag{19}$$

And the perturbation operator caused by the refractive index modulation:

$$\widehat{N} = k(\frac{n}{n_0} - 1) \tag{20}$$

The usual BPM approximation is to separate the $\widehat{D} + \widehat{N}$ operation into $\widehat{D}/2 + \widehat{N} + \widehat{D}/2$. This would not normally be acceptable because the operators doesn't commute but, if the dz step is small enough, the operators can be decorrelated.

Finally we got the linear propagation in the Fourier space:

$$\tilde{E}(z + dz) = exp\left(-i\frac{dz}{2}\frac{2\pi\nu^2 n_0}{k}\right)\tilde{E}(z) \tag{21}$$

And the index modulation:

$$E(z + dz) = exp\left(ikdz\Delta n\right)E(z) \tag{22}$$

However, the linear propagation was not implemented like described above but with the follow equation, which is more stable:

$$\tilde{E}(z + dz) = exp\left(-i\frac{dz}{2}\left[\frac{2\pi\nu^2}{\sqrt{(\frac{n_o}{\lambda})^2 - \nu^2} + \frac{n_o}{\lambda}}\right]\right)\tilde{E}(z) \tag{23}$$

References

[1] C. Ciret, *Structures de guides d'onde photo-induits et analogies quantiques.* PhD thesis, Université de Lorraine, Metz, Sept. 2013.

[2] J. Peltier, "Etude de la diffraction discrète dans un réseau de guides activé électriquement," tech. rep., Université de Lorraine, Metz, June 2018.