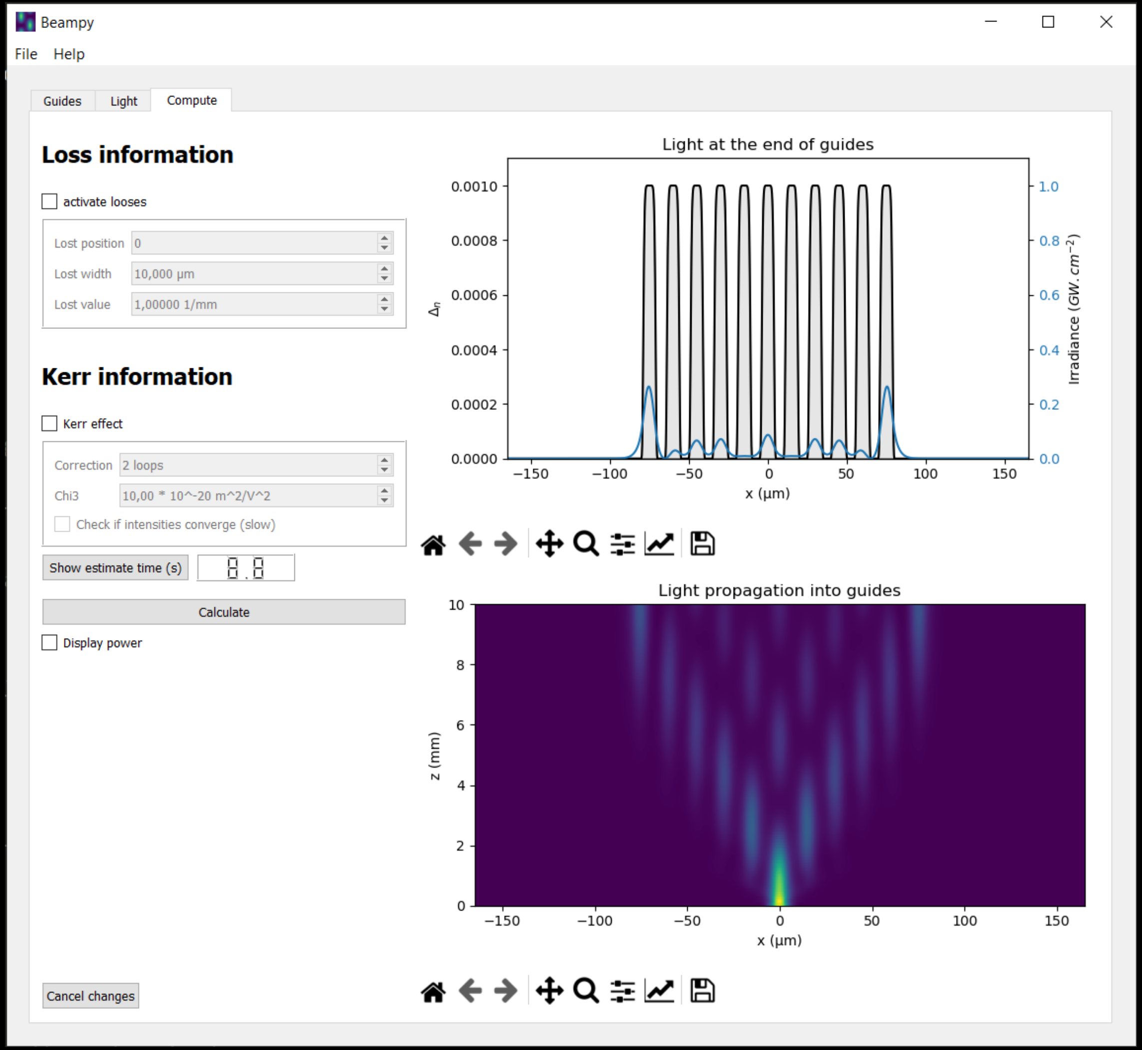
Beampy - scientific Python software development, Guided User Interface and Finite Difference Beam Propagation Method feature

Marcel Reis-Soubkovsky
M2 PAIP-POM
Université de Lorraine

Beampy

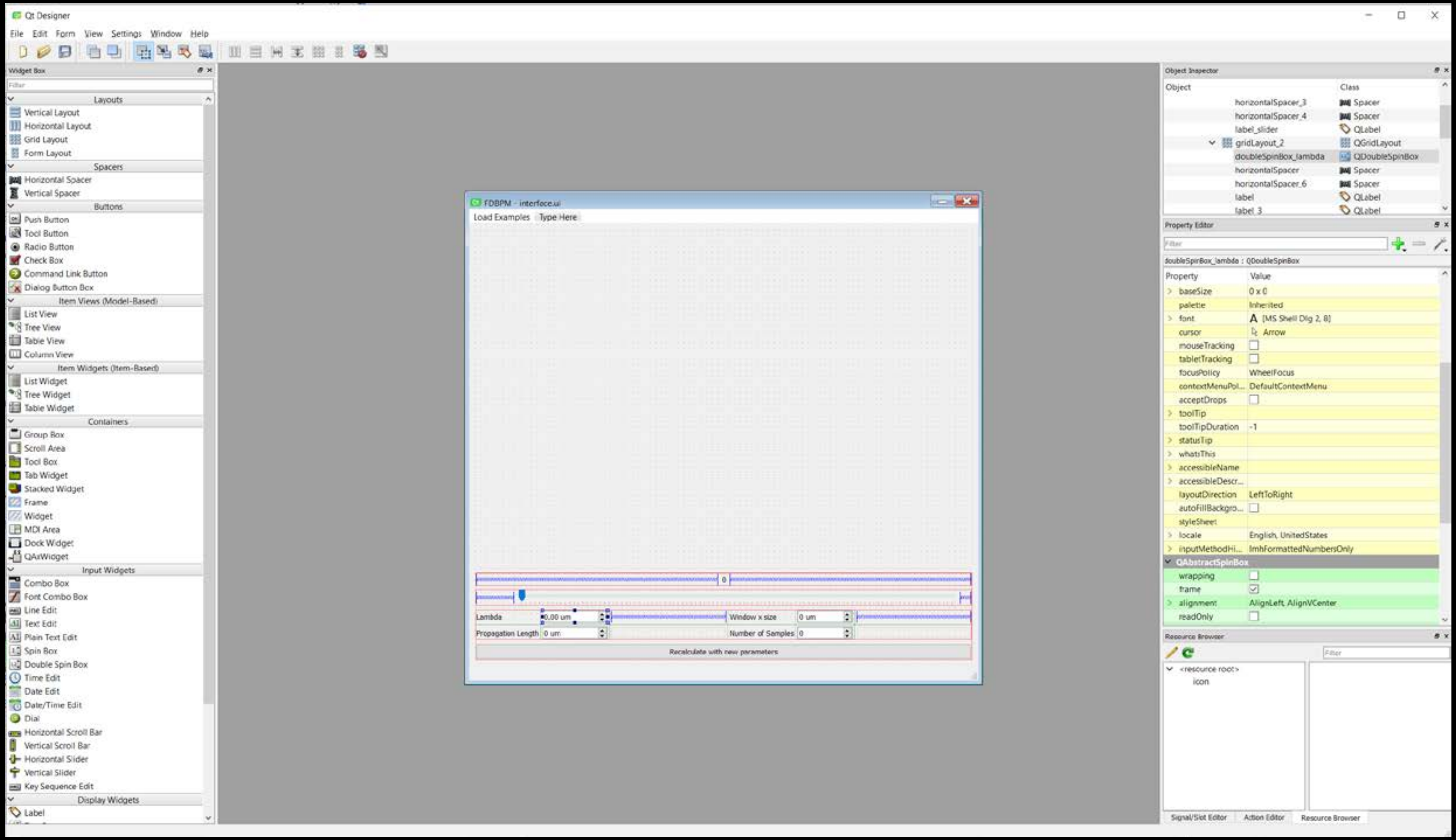


Beampy
File Help
Guides Light Compute
Loss information
activate looses
Lost position 0
Lost width 10,000 µm
Lost value 1,00000 1/mm
Kerr information
Kerr effect
Correction 2 loops
Chi3 10,00 * 10^-20 m^2/V^2
Check if intensities converge (slow)
Show estimate time (s)
Calculate
Display power
Cancel changes
Light at the end of guides
x (µm)
Irradiance (GW.cm^{-2})
Light propagation into guides
z (mm)
x (µm)

GUI Creation

- Python GUI modules:
  - Tkinter
  - PyQt5
  - PySide2

PyQt5 - Designer



- Graphical interface for creating interfaces
- Output is an XML file of extension .ui
- Can be converted to .py using PyQt5

FD-BPM

- Scalar 2D wave equation:

$$2in_0k_0\frac{\partial u}{\partial z} = \frac{\partial^2 u}{\partial x^2} + k_0^2(n^2 - n_0^2)u$$

z : propagation axis

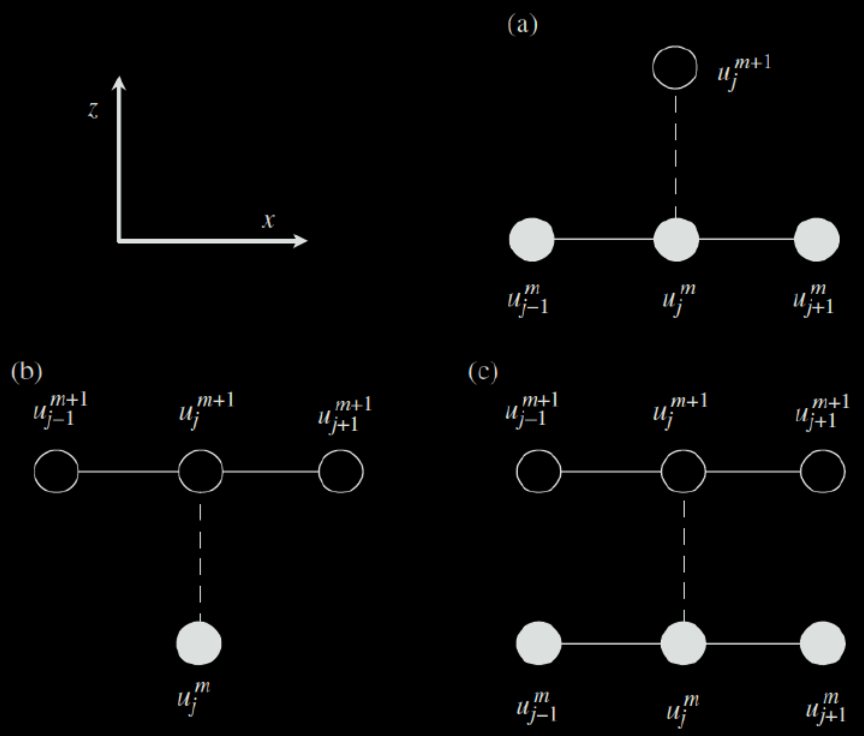
FD-BPM - Principle

- Discrete derivatives

$$\frac{\partial u}{\partial z} \text{ becomes } \frac{u_j^{m+1} - u_j^m}{\Delta z}$$

$$\frac{\partial^2 u}{\partial x^2} \text{ becomes } \frac{u_{j-1}^m - 2u_j^m + u_{j+1}^m}{(\Delta x)^2}$$

FD-BPM - Principle



Three methods of computing propagation based on finite difference: (a) fully explicit, (b) fully implicit and (c) Crank–Nicolson. (Source: Pedrola)

FD-BPM - Principle

$$a_j u_{j-1}^{m+1} + b_j u_j^{m+1} + c_j u_{j+1}^{m+1} = r_j$$

where:

$$a_j = -\frac{\alpha}{(\Delta x)^2}$$

$$b_j = \frac{2\alpha}{(\Delta x)^2} - \alpha\left[(n_j^{m+1})^2 - n_o^2\right] k_o^2 + \frac{2ik_0n_0}{\Delta z}$$

$$c_j = a_j$$

$$r_j = \frac{(1-\alpha)}{(\Delta x)^2}\left[u_{j-1}^m + u_j + 1^m\right] + \left[(1-\alpha)[(n_j^m)^2 - n_0^2]k_0^2 - 2\frac{(1-\alpha)}{(\Delta x)^2} + \frac{2ik_0n_0}{\Delta z}\right]$$

This system can be solved by using a tridiagonal calculation as in Thomas method [3].

$$
\begin{pmatrix}
b_1 & c_1 & 0 & 0 & . & . & 0 & 0 & 0 & 0 \\
a_2 & b_2 & c_2 & 0 & . & . & 0 & 0 & 0 & 0 \\
0 & a_3 & b_3 & c_3 & . & . & 0 & 0 & 0 & 0 \\
0 & 0 & a_4 & b_4 & . & . & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & a_5 & . & . & 0 & 0 & 0 & 0 \\
. & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . \\
0 & 0 & 0 & 0 & . & . & a_{n-2} & b_{n-2} & c_{n-2} & 0 \\
0 & 0 & 0 & 0 & . & . & 0 & a_{n-1} & b_{n-1} & c_{n-1} \\
0 & 0 & 0 & 0 & . & . & 0 & 0 & a_n & b_n
\end{pmatrix}
\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ . \\ . \\ u_{n-2} \\ u_{n-1} \\ u_n \end{pmatrix}
=
\begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ . \\ . \\ r_{n-2} \\ r_{n-1} \\ r_n \end{pmatrix}
\tag{B.3}
$$

This linear system can be also expressed in a compact form by:

$$
a_j u_{j-1} + b_j u_j + c_j u_{j+1} = r_j, \quad \text{for } j = 1 \text{ to } n \tag{B.4}
$$

Source: Gines Pedrola – Beam Propagation Method for Design of Optical Waveguide Devices

1. Set: $\beta = b_1$, $u_1 = r_1/\beta$.
2. Evaluate for $j = 2$ to n:

$$\gamma_j = \frac{c_{j-1}}{\beta}$$

$$\beta = b_j - a_j\gamma_j$$

$$u_j = \frac{r_j - a_j u_{j-1}}{\beta}$$

3. Find for $j = 1$ to $n - 1$:

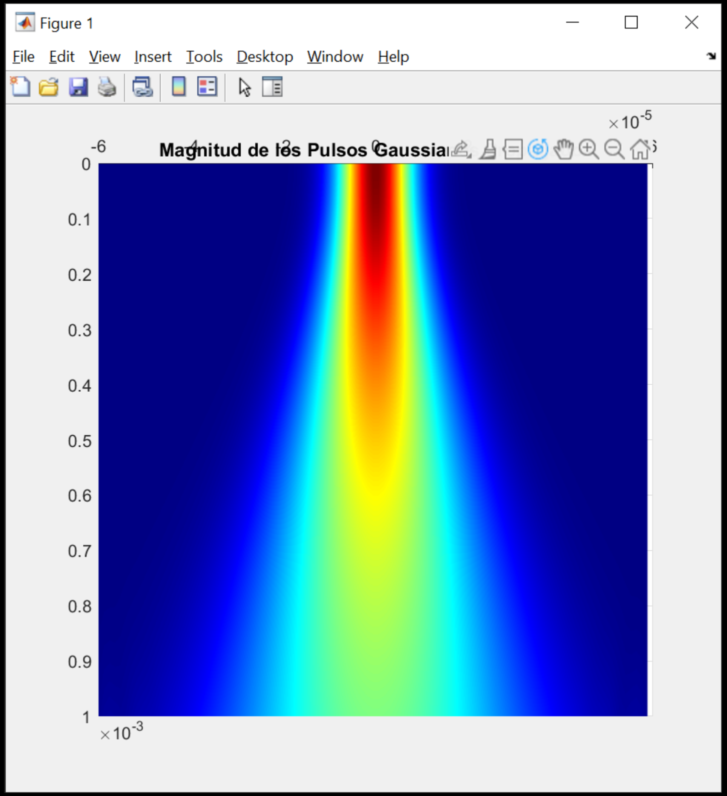
$$k = n - j$$

$$u_k = u_k - \gamma_{k+1} u_{k+1}.$$

Source: Gines Pedrola – Beam Propagation Method for Design of Optical Waveguide Devices

FD-BPM on Beampy

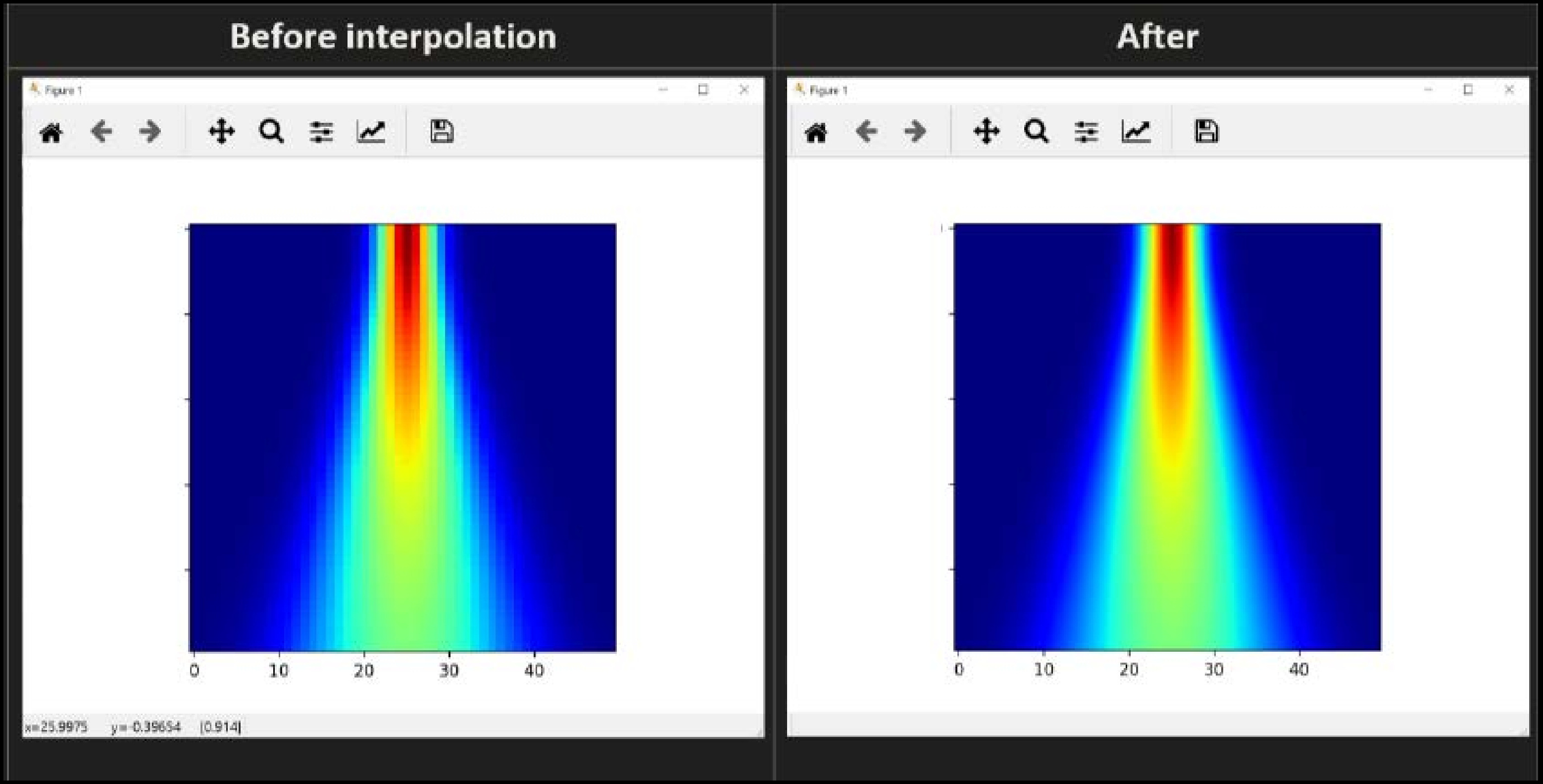
- Matlab code by Edgar Guevara



FD-BPM on Beampy - Structure

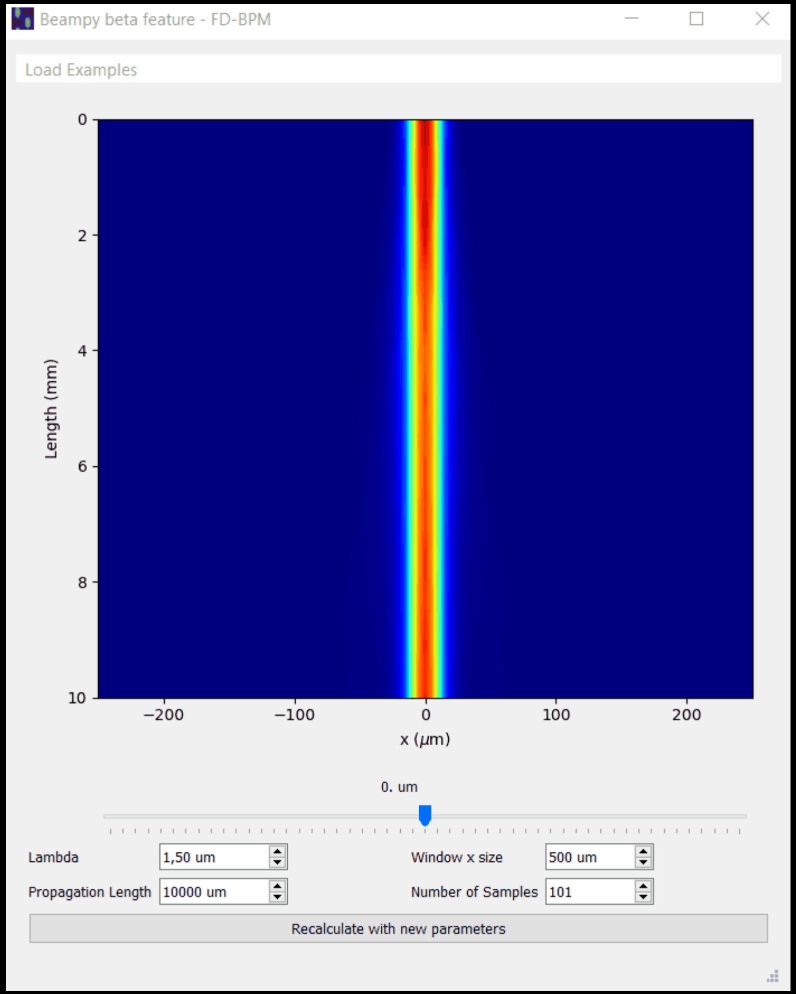
- FdBpm class
  - create_space()
  - create_source()
  - create_guides()
  - *make_tri_matrix()
  - calculate_propagation()
  - GUI methods: update_interactivity(), update_graph()

FD-BPM on Beampy – Implementation and Optimization

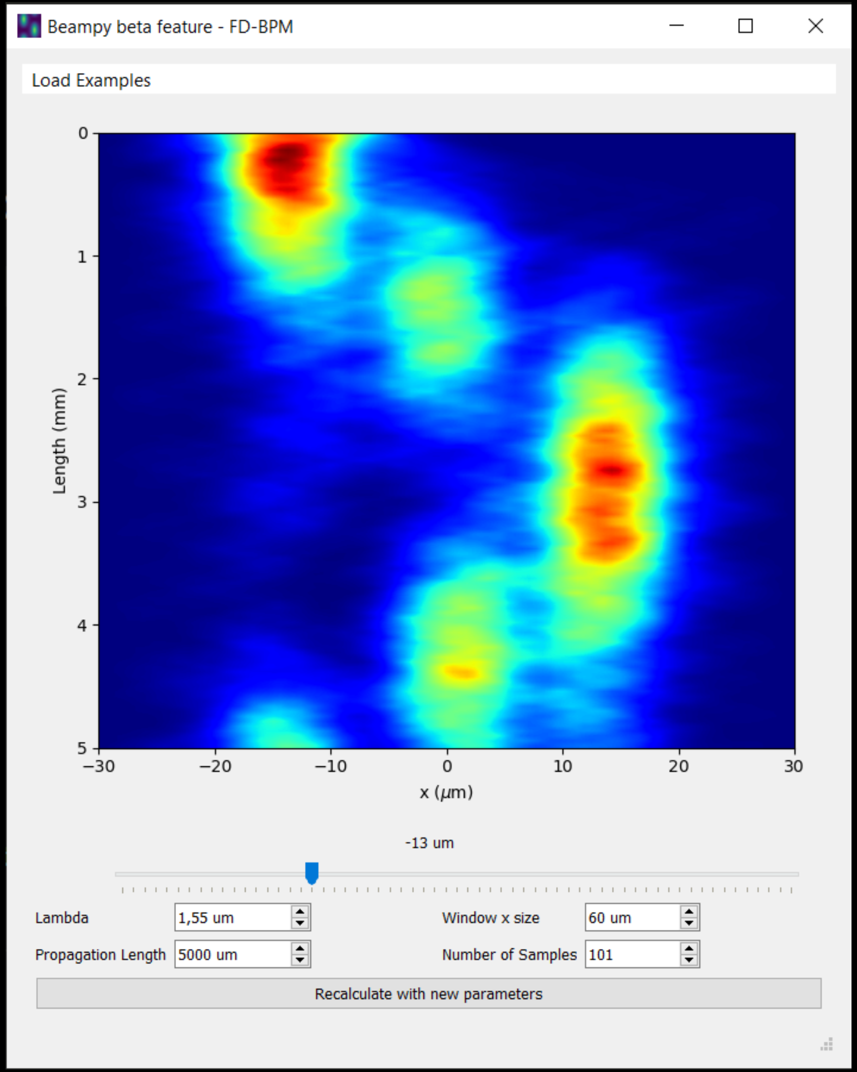


Calculated in 0.028 s ! => Possibility of Real Time calculation

FD-BPM on Beampy – Waveguide creation



FD-BPM on Beampy - Coupling Waveguides



EasyGui Module

- Matplotlib already implemented
- Just import and create GUI design
- No GUI coding needed, all through designer

+ update on Mandelbrot Set Exploration code from last year

- Multithreading makes code 4 times as fast
- Numpy optimization using masked arrays makes it run 10x faster
- OOP makes code easy to package

Thank you!