nrefocus Documentation

Release 0.2.0

Paul Müller

Mar 10, 2021

Contents:

Nrefocus is a Python 3 library that allows to numerically refocus (including autofocusing) complex wave fields. This is the documentaion of nrefocus version 0.2.0.

CHAPTER 1

Introduction

This package provides methods for numerical propagation of a complex wave in free space. The available propagators are the angular spectrum method (*helmholtz*) and the Fresnel approximation (*fresnel*). Both implementations are convolution-based. The angular spectrum method is suited for near-field propagation (numerical focusing) and yields better results than the Fresnel approximation. The single Fourer transform-based Fresnel propagation method which is suitable for far-field propagation is not implemented in this package.

1.1 Obtaining nrefocus

If you have Python and `numpy` installed, simply run

> pip install nrefocus

The source code of nrefocus is available at https://github.com/RI-imaging/nrefocus.

1.2 Citing nrefocus

Please cite this package if you are using it in a scientific publication.

This package should be cited like this[1].

You can find out what version you are using by typing (in a Python console):

```
>>> import nrefocus
>>> nrefocus.__version__
'0.1.2'
```

[1] Paul Müller (2013) *nrefocus: Python algorithms for numerical focusing* (Version x.x.x) [Software]. Available at https://pypi.python.org/pypi/nrefocus/.

References

1.3 Acknowledgments

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 282060.

CHAPTER 2

Theory

The derivations given here are treated in more detail in the relevant literature, e.g.[1] and[2].

2.1 Optical transfer function

Let us consider a wave field $u(\mathbf{r_0})$ whose values we know at an initial plane $\mathbf{r_0} = (x_0, y_0, z_0)$ (z_0 fixed). The field has a certain vacuum wavelength λ and is traveling through a homogeneous medium with refractive index n_m. From the knowledge of the wave field at the plane $\mathbf{r_0}$ and its wavelength λ/n_m, we can infer the direction of propagation of the wave field for every point in $\mathbf{r_0}$. We rewrite the field at $\mathbf{r_0}$ as an angular spectrum, a sum over all possible directions $\mathbf{s} = (p, q, M)$, assuming that the field is only traveling from left to right

$$\begin{aligned}
u(\mathbf{r_0}) &= \iint dp dq \, A(p, q) e^{ik_\mathrm{m}(px_0 + qy_0 + Mz_0)} \\
|\mathbf{s}| &= p^2 + q^2 + M^2 = 1 \\
M &= \sqrt{1 - p^2 - q^2}.
\end{aligned}$$

The equation above describes the Huygens-Fresnel principle: the value of the field u at a certain position $\mathbf{r_0}$ at the initial plane (point source) is defined as an integral over all possible plane waves with wavenumber $k_\mathrm{m} = \frac{2\pi n_\mathrm{m}}{\lambda}$, weighted with the amplitude $A(p, q)$.

Let us now consider the 2D Fourier transform of $u(\mathbf{r_0})$.

$$\begin{aligned}
\widehat{U}_0(k_\mathrm{x}, k_\mathrm{y}) &= \frac{1}{2\pi} \iint dx_0 dy_0 \iint dp dq \, A(p, q) e^{ik_\mathrm{m}(px_0 + qy_0 + Mz_0)} e^{-i(k_\mathrm{x} x_0 + k_\mathrm{y} y_0)} \\
&= \frac{1}{2\pi} \iint dx_0 dy_0 \iint dp dq \, A(p, q) e^{ik_\mathrm{m} M z_0} e^{ix_0(k_\mathrm{m} p - k_\mathrm{x})} e^{iy_0(k_\mathrm{m} q - k_\mathrm{y})} \\
&= \frac{2\pi}{k_\mathrm{m}^2} A(k_\mathrm{x}, k_\mathrm{y}) e^{ik_\mathrm{m} M z_0}
\end{aligned}$$

[1] Bahaa E. A. Saleh, Malvin Carl Teich, *Fundamentals of Photonics*, Chapter 4, John Wiley & Sons, Inc., **2001** DOI 10.1002/0471213748

[2] Joseph W. Goodman, *Introduction to Fourier Optics* 3rd ed., Roberts & Company Publishers, **2005** Publisher

Here we made use of the identity of the delta distribution

$$\frac{1}{2\pi}\int dx_0\, e^{ix_0(k_\mathrm{m}p-k_\mathrm{x})} = \delta(k_\mathrm{m}p-k_\mathrm{x}) = \frac{1}{k_\mathrm{m}}\delta(p-k_\mathrm{x}/k_\mathrm{m})$$
$$\frac{1}{2\pi}\int dy_0\, e^{iy_0(k_\mathrm{m}q-k_\mathrm{y})} = \delta(k_\mathrm{m}q-k_\mathrm{y}) = \frac{1}{k_\mathrm{m}}\delta(q-k_\mathrm{y}/k_\mathrm{m})$$

If we now perform the same procedure for a different position $\mathbf{r}_\mathrm{d} = (x_0, y_0, z_\mathrm{d})$, we will see that the Fourier transform of the field becomes

$$\widehat{U}_\mathrm{d}(k_\mathrm{x}, k_\mathrm{y}) = \frac{2\pi}{k_\mathrm{m}^2} A(k_\mathrm{x}, k_\mathrm{y}) e^{ik_\mathrm{m}Mz_\mathrm{d}}.$$

Thus, the propagation of the field $u(\mathbf{r_0})$ by a distance $d = z_\mathrm{d} - z_0$ is described by a multiplication with the transfer function

$$\mathcal{H}^\text{Helmholtz} = e^{ik_\mathrm{m}Md}$$

in Fourier space. This is the basis of the convolution-based numerical propagation algorithms implemented in `nrefocus`. The process of numerical propagation with the angular spectrum method can be written as

$$u(\mathbf{r_d}) = \mathcal{F}^{-1}\left\{\mathcal{F}\{u(\mathbf{r_0})\} \cdot e^{ik_\mathrm{m}Md}\right\}$$

with the Fourier transform $\mathcal{F}$ and its inverse $\mathcal{F}^{-1}$. With the convolution operator $*$, we may rewrite this equation to

$$u(\mathbf{r_d}) = u(\mathbf{r_0}) * \mathcal{F}^{-1}\left\{e^{ik_\mathrm{m}Md}\right\}.$$

2.2 Fresnel approximation

The Fresnel approximation (or paraxial approximation) uses a Taylor expansion to simplify the exponent of the transfer function $e^{ik_\mathrm{m}Md}$. The exponent can be rewritten as

$$ik_\mathrm{m}Md = ik_\mathrm{m}d\left(1-p^2-q^2\right)^{1/2}.$$

If the angles of propagation θ_x and θ_y for each plane wave of the angular spectrum is small, then we can make the paraxial approximation:

$$\theta_\mathrm{x} \approx p$$
$$\theta_\mathrm{y} \approx q$$
$$\theta^2 = \theta_\mathrm{x}^2 + \theta_\mathrm{y}^2 \approx p^2 + q^2$$

We now Taylor-expand the exponent around small values of θ

$$ik_\mathrm{m}d\left(1-\theta^2\right)^{1/2} \approx ik_\mathrm{m}d\left(1 - \frac{\theta^2}{2} + \frac{\theta^4}{8} - \ldots\right).$$

The Fresnel approximation discards the third term ($\sim \theta^4$) and the transfer function then reads:

$$e^{ik_\mathrm{m}Md} \approx e^{ik_\mathrm{m}d} \cdot e^{-\frac{ik_\mathrm{m}d(p^2+q^2)}{2}}$$
$$e^{i\sqrt{k_\mathrm{m}^2-k_\mathrm{x}^2-k_\mathrm{y}^2}d} \approx e^{ik_\mathrm{m}d} \cdot e^{-\frac{id(k_\mathrm{x}^2+k_\mathrm{y}^2)}{2k_\mathrm{m}}}$$
$$\mathcal{H}^\text{Fresnel} = e^{ik_\mathrm{m}d} \cdot e^{-\frac{id(k_\mathrm{x}^2+k_\mathrm{y}^2)}{2k_\mathrm{m}}}$$

Thus, the propagation by a distance distance $d = z_\mathrm{d} - d$ in the Fresnel approximation can be written in the form of the convolution

$$u(\mathbf{r_d}) = e^{ik_\mathrm{m}d} \cdot u(\mathbf{r_0}) * \mathcal{F}^{-1}\left\{e^{-\frac{id(k_\mathrm{x}^2+k_\mathrm{y}^2)}{2k_\mathrm{m}}}\right\}.$$

Note that the Fresnel approximation results in paraboloidal waves $(p^2 + q^2)$ whereas spherical waves are used with the Helmholtz equation.

2.3 Transfer functions in nrefocus

The numerical focusing algorithms in this package require the input data u_{in} to be normalized by the incident plane wave $u_0(\mathbf{r_0})$ according to

$$u_{\text{in}}(\mathbf{r_0}) = \frac{u(\mathbf{r_0})}{u_0(\mathbf{r_0})}$$

As a result, the transfer functions change to

$$\mathcal{H}_{\text{norm}}^{\text{Helmholtz}} = e^{ik_{\text{m}}(M-1)d}$$
$$\mathcal{H}_{\text{norm}}^{\text{Fresnel}} = e^{-\frac{id(k_{\text{x}}^2+k_{\text{y}}^2)}{2k_{\text{m}}}}.$$

2.4 References

CHAPTER 3

Code reference

3.1 Refocus 1D/2D fields

`fft_propagate`	
refocus(field, d, nm, res[, method, ...])	Refocus a 1D or 2D field
refocus_stack(fieldstack, d, nm, res[, ...])	Refocus a stack of 1D or 2D fields

3.1.1 Fourier-domain propagation

3.1.2 Refocus individual fields

nrefocus.**refocus** (*field*, *d*, *nm*, *res*, *method='helmholtz'*, *num_cpus=1*, *padding=True*)

Refocus a 1D or 2D field

Parameters

- **field** (*1d or 2d array*) – 1D or 2D background corrected electric field (Ex/BEx)
- **d** (*float*) – Distance to be propagated in pixels (negative for backwards)
- **nm** (*float*) – Refractive index of medium
- **res** (*float*) – Wavelenth in pixels
- **method** (*str*) – Defines the method of propagation; one of
  - "helmholtz" : the optical transfer function *exp(idk(M-1))*
  - "fresnel" : paraxial approximation *exp(idk*^{2}*/k)*
- **num_cpus** (*int*) – Not implemented. Only one CPU is used.
- **padding** (*bool*) – perform padding with linear ramp from edge to average to reduce ringing artifacts.

  New in version 0.1.4.

Returns

Return type Electric field at *d*.

3.1.3 Refocus field stacks

`nrefocus.refocus_stack`(*fieldstack*, *d*, *nm*, *res*, *method='helmholtz'*, *num_cpus=2*, *copy=True*, *padding=True*)

Refocus a stack of 1D or 2D fields

Parameters

- **fieldstack** (*2d or 3d array*) – Stack of 1D or 2D background corrected electric fields (Ex/BEx). The first axis iterates through the individual fields.
- **d** (*float*) – Distance to be propagated in pixels (negative for backwards)
- **nm** (*float*) – Refractive index of medium
- **res** (*float*) – Wavelenth in pixels
- **method** (*str*) – Defines the method of propagation; one of
  - "helmholtz" : the optical transfer function *exp(idk(M-1))*
  - "fresnel" : paraxial approximation $exp(idk^2/k)$
- **num_cpus** (*str*) – Defines the number of CPUs to be used for refocusing.
- **copy** (*bool*) – If False, overwrites input stack.
- **padding** (*bool*) – Perform padding with linear ramp from edge to average to reduce ringing artifacts.

  New in version 0.1.4.

Returns

Return type Electric field stack at *d*.

3.2 Autofocus 1D/2D fields

autofocus(field, nm, res, ival[, roi, ...])	Numerical autofocusing of a field using the Helmholtz equation.
autofocus_stack(fieldstack, nm, res, ival[, ...])	Numerical autofocusing of a stack using the Helmholtz equation.

3.2.1 Metrics

`nrefocus.metrics.average_gradient`(*data*, **kwargs*)

Compute average gradient norm of an image

`nrefocus.metrics.contrast_rms`(*data*, **kwargs*)

Compute RMS contrast norm of an image

`nrefocus.metrics.spectral`(*data*, *lambd*, **kwargs*)

Compute spectral contrast of image

Performs bandpass filtering in Fourier space according to optical limit of detection system, approximated by twice the wavelength.

Parameters

- **data** (*2d ndarray*) – the image to compute the norm from
- **lambd** (*float*) – wavelength of the light in pixels

3.2.2 Autofocus single fields

nrefocus.**autofocus**(*field*, *nm*, *res*, *ival*, *roi=None*, *metric='average gradient'*, *padding=True*, *ret_d=False*, *ret_grad=False*, *num_cpus=1*)

Numerical autofocusing of a field using the Helmholtz equation.

Parameters

- **field** (*1d or 2d ndarray*) – Electric field is BG-Corrected, i.e. field = EX/BEx
- **nm** (*float*) – Refractive index of medium.
- **res** (*float*) – Size of wavelength in pixels.
- **ival** (*tuple of floats*) – Approximate interval to search for optimal focus in px.
- **roi** (*rectangular region of interest (x1, y1, x2, y2)*) – Region of interest of *field* for which the metric will be minimized. If not given, the entire *field* will be used.
- **metric** (*str*) –
  - "average gradient" : average gradient metric of amplitude
  - "rms contrast" : RMS contrast of phase data
  - "spectrum" : sum of filtered Fourier coefficients
- **padding** (*bool*) – Perform padding with linear ramp from edge to average to reduce ringing artifacts.

  Changed in version 0.1.4: improved padding value and padding location
- **red_d** (*bool*) – Return the autofocusing distance in pixels. Defaults to False.
- **red_grad** (*bool*) – Return the computed gradients as a list.
- **num_cpus** (*int*) – Not implemented.

Returns

- *field, [d, [grad]]*
- *The focused field and optionally, the optimal focusing distance and*
- *the computed gradients.*

3.2.3 Autofocus field stacks

nrefocus.**autofocus_stack**(*fieldstack*, *nm*, *res*, *ival*, *roi=None*, *metric='average gradient'*, *padding=True*, *same_dist=False*, *ret_ds=False*, *ret_grads=False*, *num_cpus=2*, *copy=True*)

Numerical autofocusing of a stack using the Helmholtz equation.

Parameters

- **fieldstack** (*2d or 3d ndarray*) – Electric field is BG-Corrected, i.e. Field = EX/BEx
- **nm** (*float*) – Refractive index of medium.
- **res** (*float*) – Size of wavelength in pixels.
- **ival** (*tuple of floats*) – Approximate interval to search for optimal focus in px.
- **metric** (*str*) – see *autofocus_field*.
- **padding** (*bool*) – Perform padding with linear ramp from edge to average to reduce ringing artifacts.

  Changed in version 0.1.4: improved padding value and padding location
- **ret_dopt** (*bool*) – Return optimized distance and gradient plotting data.
- **same_dist** (*bool*) – Refocus entire sinogram with one distance.
- **red_ds** (*bool*) – Return the autofocusing distances in pixels. Defaults to False. If sam_dist is True, still returns autofocusing distances of first pass. The used refocusing distance is the average.
- **red_grads** (*bool*) – Return the computed gradients as a list.
- **copy** (*bool*) – If False, overwrites input array.

Returns

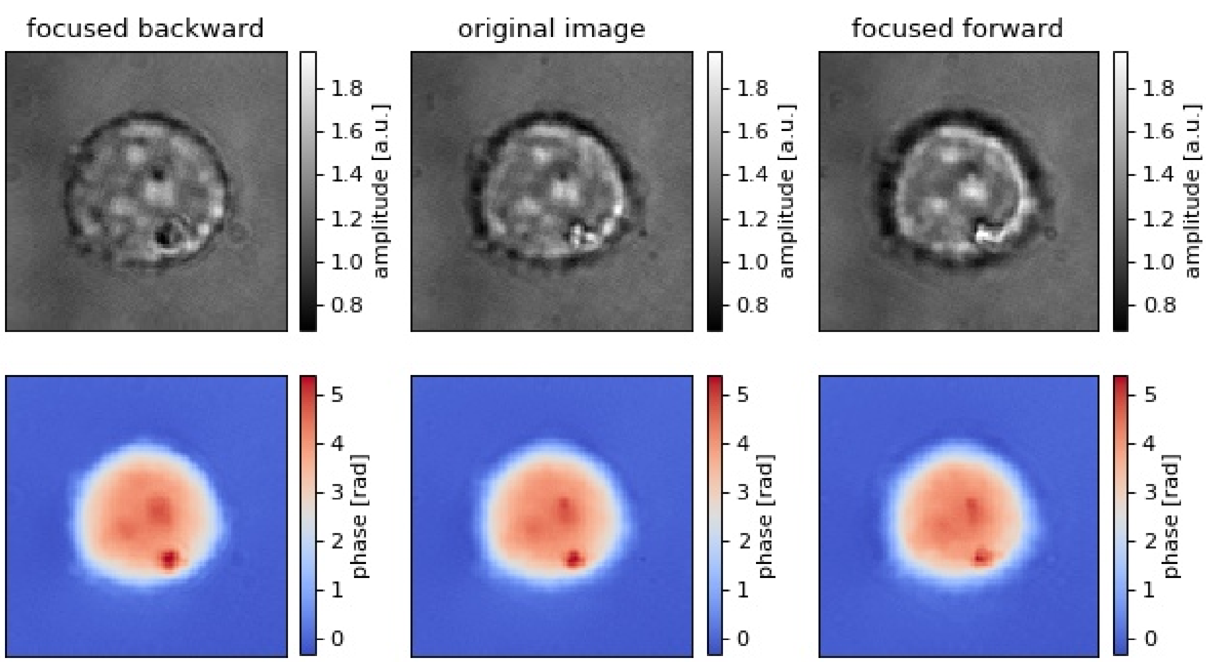
Return type The focused field (and the refocussing distance + data if d is None)

CHAPTER 4

Examples

4.1 2D Refocusing of an HL60 cell

The data show a live HL60 cell imaged with quadriwave lateral shearing interferometry (SID4Bio, Phasics S.A., France). The diameter of the cell is about 20µm.



refocus_cell.py

```
import matplotlib.pylab as plt
import numpy as np
import unwrap

import nrefocus

from example_helper import load_cell

# load initial cell
cell1 = load_cell("HL60_field.zip")

# refocus to two different positions
cell2 = nrefocus.refocus(cell1, 15, 1, 1)  # forward
cell3 = nrefocus.refocus(cell1, -15, 1, 1)  # backward

# amplitude range
vmina = np.min(np.abs(cell1))
vmaxa = np.max(np.abs(cell1))
ampkw = {"cmap": plt.get_cmap("gray"),
         "vmin": vmina,
         "vmax": vmaxa}

# phase range
cell1p = unwrap.unwrap(np.angle(cell1))
cell2p = unwrap.unwrap(np.angle(cell2))
cell3p = unwrap.unwrap(np.angle(cell3))
vminp = np.min(cell1p)
vmaxp = np.max(cell1p)
phakw = {"cmap": plt.get_cmap("coolwarm"),
         "vmin": vminp,
         "vmax": vmaxp}

# plots
fig, axes = plt.subplots(2, 3, figsize=(8, 4.5))
axes = axes.flatten()
for ax in axes:
    ax.xaxis.set_major_locator(plt.NullLocator())
    ax.yaxis.set_major_locator(plt.NullLocator())

# titles
axes[0].set_title("focused backward")
axes[1].set_title("original image")
axes[2].set_title("focused forward")

# data
mapamp = axes[0].imshow(np.abs(cell3), **ampkw)
axes[1].imshow(np.abs(cell1), **ampkw)
axes[2].imshow(np.abs(cell2), **ampkw)
mappha = axes[3].imshow(cell3p, **phakw)
axes[4].imshow(cell1p, **phakw)
axes[5].imshow(cell2p, **phakw)

# colobars
cbkwargs = {"fraction": 0.045}
plt.colorbar(mapamp, ax=axes[0], label="amplitude [a.u.]", **cbkwargs)
plt.colorbar(mapamp, ax=axes[1], label="amplitude [a.u.]", **cbkwargs)
plt.colorbar(mapamp, ax=axes[2], label="amplitude [a.u.]", **cbkwargs)
```

(continues on next page)

(continued from previous page)

```
plt.colorbar(mappha, ax=axes[3], label="phase [rad]", **cbkwargs)
plt.colorbar(mappha, ax=axes[4], label="phase [rad]", **cbkwargs)
plt.colorbar(mappha, ax=axes[5], label="phase [rad]", **cbkwargs)

plt.tight_layout()
plt.show()
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

Index

A

C

N

R

S