

---

# **nrefocus Documentation**

***Release 0.4.3***

**Paul Müller**

**Apr 23, 2021**



## CONTENTS:

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Obtaining nrefocus . . . . .	3
1.2	Citing nrefocus . . . . .	3
1.3	Acknowledgments . . . . .	4
1.4	References . . . . .	4
<b>2</b>	<b>Theory</b>	<b>5</b>
2.1	Optical transfer function . . . . .	5
2.2	Fresnel approximation . . . . .	6
2.3	Transfer functions in nrefocus . . . . .	7
<b>3</b>	<b>Examples</b>	<b>9</b>
3.1	2D Refocusing of an HL60 cell . . . . .	9
<b>4</b>	<b>Code reference</b>	<b>11</b>
4.1	Refocus interface . . . . .	11
4.2	Metrics . . . . .	15
4.3	Legacy methods . . . . .	15
4.3.1	Refocusing . . . . .	15
4.3.2	Autofocusing . . . . .	16
<b>5</b>	<b>Changelog</b>	<b>19</b>
5.1	version 0.4.3 . . . . .	19
5.2	version 0.4.2 . . . . .	19
5.3	version 0.4.1 . . . . .	19
5.4	version 0.4.0 . . . . .	19
5.5	version 0.3.1 . . . . .	20
5.6	version 0.3.0 . . . . .	20
5.7	version 0.2.1 . . . . .	20
5.8	version 0.2.0 . . . . .	20
5.9	version 0.1.8 . . . . .	20
5.10	version 0.1.7 . . . . .	20
5.11	version 0.1.6 . . . . .	21
5.12	version 0.1.5 . . . . .	21
5.13	version 0.1.4 . . . . .	21
<b>6</b>	<b>Bibliography</b>	<b>23</b>
<b>7</b>	<b>Indices and tables</b>	<b>25</b>
	<b>Bibliography</b>	<b>27</b>

<b>Python Module Index</b>	<b>29</b>
<b>Index</b>	<b>31</b>

Nrefocus is a Python 3 library that allows to numerically refocus (including autofocusing) complex wave fields. This is the documentaion of nrefocus version 0.4.3.



## 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 Fourier transform-based Fresnel propagation method which is suitable for far-field propagation is not implemented in this package.

### 1.1 Obtaining nrefocus

You can install nrefocus via:

```
pip install nrefocus
```

If you would like to take advantage of fast Fourier transforms with **PyFFTW**, please also install the *pyfftw* package or use the extras key *FFTW*:

```
pip install nrefocus[FFTW]
```

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<sup>1</sup>.

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

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

---

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

## 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.

## 1.4 References



## THEORY

The derivations given here are treated in more detail in the relevant literature, e.g. [[ST91]] and [[Goo05]].

### 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  $\lambda$  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  $\lambda/n_m$ , 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

$$\begin{aligned} u(\mathbf{r}_0) &= \iint dp dq A(p, q) e^{ik_m(px_0 + qy_0 + Mz_0)} \\ |\mathbf{s}| &= p^2 + q^2 + M^2 = 1 \\ M &= \sqrt{1 - p^2 - q^2}. \end{aligned}$$

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  $k_m = \frac{2\pi n_m}{\lambda}$ , weighted with the amplitude  $A(p, q)$ .

Let us now consider the 2D Fourier transform of  $u(\mathbf{r}_0)$ .

$$\begin{aligned} \hat{U}_0(k_x, k_y) &= \frac{1}{2\pi} \iint dx_0 dy_0 \iint dp dq A(p, q) e^{ik_m(px_0 + qy_0 + Mz_0)} e^{-i(k_x x_0 + k_y y_0)} \\ &= \frac{1}{2\pi} \iint dx_0 dy_0 \iint dp dq A(p, q) e^{ik_m M z_0} e^{ix_0(k_m p - k_x)} e^{iy_0(k_m q - k_y)} \\ &= \frac{2\pi}{k_m^2} A(k_x, k_y) e^{ik_m M z_0} \end{aligned}$$

Here we made use of the identity of the delta distribution

$$\begin{aligned} \frac{1}{2\pi} \int dx_0 e^{ix_0(k_m p - k_x)} &= \delta(k_m p - k_x) = \frac{1}{k_m} \delta(p - k_x/k_m) \\ \frac{1}{2\pi} \int dy_0 e^{iy_0(k_m q - k_y)} &= \delta(k_m q - k_y) = \frac{1}{k_m} \delta(q - k_y/k_m) \end{aligned}$$

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

$$\hat{U}_d(k_x, k_y) = \frac{2\pi}{k_m^2} A(k_x, k_y) e^{ik_m M z_d}.$$

Thus, the propagation of the field  $u(\mathbf{r}_0)$  by a distance  $d = z_d - z_0$  is described by a multiplication with the transfer function

$$\mathcal{H}^{\text{Helmholtz}} = e^{ik_m M d}$$

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

$$u(\mathbf{r}_d) = \mathcal{F}^{-1} \{ \mathcal{F} \{ u(\mathbf{r}_0) \} \cdot e^{ik_m M d} \}$$

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

$$u(\mathbf{r}_d) = u(\mathbf{r}_0) * \mathcal{F}^{-1} \{ e^{ik_m M d} \}.$$

## 2.2 Fresnel approximation

The Fresnel approximation (or paraxial approximation) uses a Taylor expansion to simplify the exponent of the transfer function  $e^{ik_m M d}$ . The exponent can be rewritten as

$$ik_m M d = ik_m d (1 - p^2 - q^2)^{1/2}.$$

If the angles of propagation  $\theta_x$  and  $\theta_y$  for each plane wave of the angular spectrum is small, then we can make the paraxial approximation:

$$\begin{aligned} \theta_x &\approx p \\ \theta_y &\approx q \\ \theta^2 &= \theta_x^2 + \theta_y^2 \approx p^2 + q^2 \end{aligned}$$

We now Taylor-expand the exponent around small values of  $\theta$

$$ik_m d (1 - \theta^2)^{1/2} \approx ik_m d \left( 1 - \frac{\theta^2}{2} + \frac{\theta^4}{8} - \dots \right).$$

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

$$\begin{aligned} e^{ik_m M d} &\approx e^{ik_m d} \cdot e^{-\frac{ik_m d(p^2 + q^2)}{2}} \\ e^{i\sqrt{k_m^2 - k_x^2 - k_y^2} d} &\approx e^{ik_m d} \cdot e^{-\frac{id(k_x^2 + k_y^2)}{2k_m}} \\ \mathcal{H}^{\text{Fresnel}} &= e^{ik_m d} \cdot e^{-\frac{id(k_x^2 + k_y^2)}{2k_m}} \end{aligned}$$

Thus, the propagation by a distance  $d = z_d - d$  in the Fresnel approximation can be written in the form of the convolution

$$u(\mathbf{r}_d) = e^{ik_m d} \cdot u(\mathbf{r}_0) * \mathcal{F}^{-1} \left\{ e^{-\frac{id(k_x^2 + k_y^2)}{2k_m}} \right\}.$$

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  $u_{\text{in}}$  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

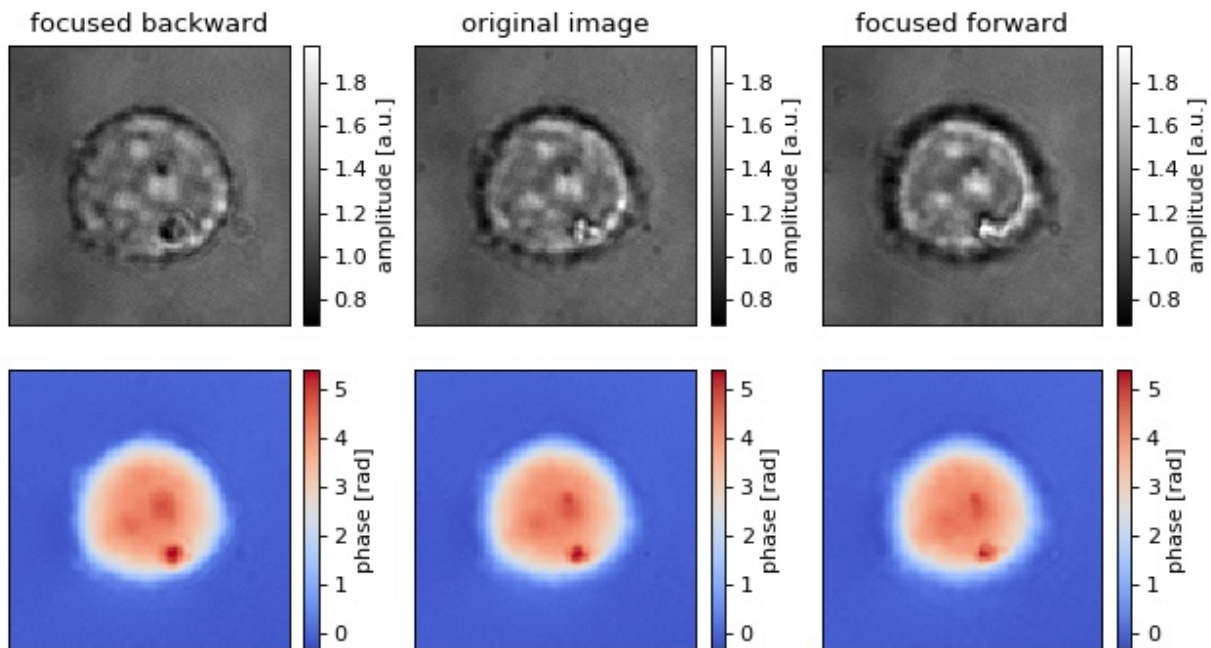
$$\begin{aligned}\mathcal{H}_{\text{norm}}^{\text{Helmholtz}} &= e^{ik_{\text{m}}(M-1)d} = e^{id(\sqrt{k_{\text{m}}^2 - k_{\text{x}}^2 - k_{\text{y}}^2} - k_{\text{m}})} \\ \mathcal{H}_{\text{norm}}^{\text{Fresnel}} &= e^{-\frac{id(k_{\text{x}}^2 + k_{\text{y}}^2)}{2k_{\text{m}}}}.\end{aligned}$$



## EXAMPLES

### 3.1 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  $20\mu\text{m}$ .



refocus\_cell.py

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import unwrap
4
5 import nrefocus
6
7 from example_helper import load_cell
8
9 # load initial cell
10 cell1 = load_cell("HL60_field.zip")
```

(continues on next page)

(continued from previous page)

```

11
12 # refocus to two different positions
13 cell2 = nrefocus.refocus(cell1, 15, 1, 1) # forward
14 cell3 = nrefocus.refocus(cell1, -15, 1, 1) # backward
15
16 # amplitude range
17 vmina = np.min(np.abs(cell1))
18 vmaxa = np.max(np.abs(cell1))
19 ampkw = {"cmap": plt.get_cmap("gray"),
20          "vmin": vmina,
21          "vmax": vmaxa}
22
23 # phase range
24 cell1p = unwrap.unwrap(np.angle(cell1))
25 cell2p = unwrap.unwrap(np.angle(cell2))
26 cell3p = unwrap.unwrap(np.angle(cell3))
27 vminp = np.min(cell1p)
28 vmxp = np.max(cell1p)
29 phakw = {"cmap": plt.get_cmap("coolwarm"),
30          "vmin": vminp,
31          "vmax": vmxp}
32
33 # plots
34 fig, axes = plt.subplots(2, 3, figsize=(8, 4.5))
35 axes = axes.flatten()
36 for ax in axes:
37     ax.xaxis.set_major_locator(plt.NullLocator())
38     ax.yaxis.set_major_locator(plt.NullLocator())
39
40 # titles
41 axes[0].set_title("focused backward")
42 axes[1].set_title("original image")
43 axes[2].set_title("focused forward")
44
45 # data
46 mapamp = axes[0].imshow(np.abs(cell3), **ampkw)
47 axes[1].imshow(np.abs(cell1), **ampkw)
48 axes[2].imshow(np.abs(cell2), **ampkw)
49 mappha = axes[3].imshow(cell3p, **phakw)
50 axes[4].imshow(cell1p, **phakw)
51 axes[5].imshow(cell2p, **phakw)
52
53 # colobars
54 cbkwargs = {"fraction": 0.045}
55 plt.colorbar(mapamp, ax=axes[0], label="amplitude [a.u.]", **cbkwargs)
56 plt.colorbar(mapamp, ax=axes[1], label="amplitude [a.u.]", **cbkwargs)
57 plt.colorbar(mapamp, ax=axes[2], label="amplitude [a.u.]", **cbkwargs)
58 plt.colorbar(mappha, ax=axes[3], label="phase [rad]", **cbkwargs)
59 plt.colorbar(mappha, ax=axes[4], label="phase [rad]", **cbkwargs)
60 plt.colorbar(mappha, ax=axes[5], label="phase [rad]", **cbkwargs)
61
62 plt.tight_layout()
63 plt.show()

```

## CODE REFERENCE

### 4.1 Refocus interface

*Refocus* is a user-convenient interface for numerical refocusing. Each class implements refocusing for a specific dimensionality (1D or 2D fields) using a specific method for refocusing (e.g. numpy FFT or FFTW).

`nrefocus.get_best_interface()`

Return the fastest refocusing interface available

If *pyfftw* is installed, `nrefocus.RefocusPyFFTW` is returned. The fallback is `nrefocus.RefocusNumpy`.

**class** `nrefocus.RefocusPyFFTW`(*field*, *wavelength*, *pixel\_size*, *medium\_index*=1.3333, *distance*=0, *kernel*='helmholtz', *padding*=True)

Refocusing with FFTW

New in version 0.4.0.

#### Parameters

- **field** (2d complex-valued ndarray) – Input field to be refocused
- **wavelength** (float) – Wavelength of the used light [m]
- **pixel\_size** (float) – Pixel size of the input image [m]
- **medium\_index** (float) – Refractive index of the medium, defaults to water (1.3333 at 21.5°C)
- **distance** (float) – Initial focusing distance [m]
- **kernel** (str) – Propagation kernel, one of
  - "helmholtz": the optical transfer function  $\exp\left(id\left(\sqrt{k_m^2 - k_x^2 - k_y^2} - k_m\right)\right)$
  - "fresnel": paraxial approximation  $\exp(-id(k_x^2 + k_y^2)/2k_m)$
- **padding** (bool) – Whether or not to perform zero-padding

**autofocus** (*metric*='average gradient', *minimizer*='legacy', *minimizer\_kwargs*=None, *interval*=(None, None), *roi*=None)

Autofocus the initial field

#### Parameters

- **metric** (str) –
  - "average gradient" : average gradient metric of amplitude
  - "rms contrast" : RMS contrast of phase data

- "spectrum": sum of filtered Fourier coefficients
- **minimizer** (*str*) –
  - "legacy": custom nrefocus minimizer
- **interval** (*tuple of floats*) – Approximate interval to search for optimal focus [m]
- **roi** (*list or tuple or slice or ndarray*) – Region of interest for which the metric will be minimized. This can be either a list [x1, y1, x2, y2], a tuple or list of slices or a numpy indexing array. If not given, the entire field will be used.
- **minimizer\_kwargs** (*dict*) – Any additional keyword arguments for the minimizer

**Returns**

- **af\_field** (*2d ndarray*) – Autofocused field
- **af\_distance** (*float*) – Autofocusing distance
- [*other*] – Any other objects returned by *minimizer* defined via *minimizer\_kwargs*

**get\_kernel** (*distance*)

Return the current kernel

The kernel type *self.kernel* is used (see `Refocus.__init__()`)**propagate** (*distance*)

Propagate the initial field to a certain distance

**Parameters** **distance** (*float*) – Absolute focusing distance [m]**Returns** **refocused\_field** – Initial field refocused at *distance***Return type** 2d ndarray**Notes**Any subclass should perform padding with `nrefocus.pad.pad_rem()` during initialization.**property shape**

Shape of the padded input field or Fourier transform

**class** `nrefocus.RefocusNumpy` (*field, wavelength, pixel\_size, medium\_index=1.3333, distance=0, kernel='helmholtz', padding=True*)

Refocusing with numpy-based Fourier transform

New in version 0.3.0.

**Parameters**

- **field** (*2d complex-valued ndarray*) – Input field to be refocused
- **wavelength** (*float*) – Wavelength of the used light [m]
- **pixel\_size** (*float*) – Pixel size of the input image [m]
- **medium\_index** (*float*) – Refractive index of the medium, defaults to water (1.3333 at 21.5°C)
- **distance** (*float*) – Initial focusing distance [m]
- **kernel** (*str*) – Propagation kernel, one of
  - "helmholtz": the optical transfer function  $\exp\left(id\left(\sqrt{k_m^2 - k_x^2 - k_y^2} - k_m\right)\right)$



– “fresnel”: paraxial approximation  $\exp(-id(k_x^2 + k_y^2)/2k_m)$

- **padding** (*bool*) – Whether or not to perform zero-padding

**autofocus** (*metric='average gradient', minimizer='legacy', minimizer\_kwargs=None, interval=(None, None), roi=None*)

Autofocus the initial field

#### Parameters

- **metric** (*str*) –
  - “average gradient”: average gradient metric of amplitude
  - “rms contrast”: RMS contrast of phase data
  - “spectrum”: sum of filtered Fourier coefficients
- **minimizer** (*str*) –
  - “legacy”: custom nrefocus minimizer
- **interval** (*tuple of floats*) – Approximate interval to search for optimal focus [m]
- **roi** (*list or tuple or slice or ndarray*) – Region of interest for which the metric will be minimized. This can be either a list [x1, y1, x2, y2], a tuple or list of slices or a numpy indexing array. If not given, the entire field will be used.
- **minimizer\_kwargs** (*dict*) – Any additional keyword arguments for the minimizer

#### Returns

- **af\_field** (*2d ndarray*) – Autofocused field
- **af\_distance** (*float*) – Autofocusing distance
- [*other*] – Any other objects returned by *minimizer* defined via *minimizer\_kwargs*

**get\_kernel** (*distance*)

Return the current kernel

The kernel type *self.kernel* is used (see `Refocus.__init__()`)

**propagate** (*distance*)

Propagate the initial field to a certain distance

**Parameters** **distance** (*float*) – Absolute focusing distance [m]

**Returns** **refocused\_field** – Initial field refocused at *distance*

**Return type** 2d ndarray

#### Notes

Any subclass should perform padding with `nrefocus.pad.pad_rem()` during initialization.

**property shape**

Shape of the padded input field or Fourier transform

**class** `nrefocus.RefocusNumpy1D` (*field, wavelength, pixel\_size, medium\_index=1.3333, distance=0, kernel='helmholtz', padding=True*)

Refocus a 1D field with numpy

New in version 0.3.0.

#### Parameters

- **field** (*1d complex-valued ndarray*) – Input 1D field to be refocused
- **wavelength** (*float*) – Wavelength of the used light [m]
- **pixel\_size** (*float*) – Pixel size of the input image [m]
- **medium\_index** (*float*) – Refractive index of the medium, defaults to water (1.3333 at 21.5°C)
- **distance** (*float*) – Initial focusing distance [m]
- **kernel** (*str*) – Propagation kernel, one of
  - “helmholtz”: the optical transfer function  $\exp\left(id\left(\sqrt{k_m^2 - k_x^2} - k_m\right)\right)$
  - “fresnel”: paraxial approximation  $\exp(-idk_x^2/2k_m)$
- **padding** (*bool*) – Whether or not to perform zero-padding

**autofocus** (*metric='average gradient', minimizer='legacy', minimizer\_kwargs=None, interval=(None, None), roi=None*)

Autofocus the initial field

#### Parameters

- **metric** (*str*) –
  - “average gradient”: average gradient metric of amplitude
  - “rms contrast”: RMS contrast of phase data
  - “spectrum”: sum of filtered Fourier coefficients
- **minimizer** (*str*) –
  - “legacy”: custom nrefocus minimizer
- **interval** (*tuple of floats*) – Approximate interval to search for optimal focus [m]
- **roi** (*list or tuple or slice or ndarray*) – Region of interest for which the metric will be minimized. This can be either a list [x1, y1, x2, y2], a tuple or list of slices or a numpy indexing array. If not given, the entire field will be used.
- **minimizer\_kwargs** (*dict*) – Any additional keyword arguments for the minimizer

#### Returns

- **af\_field** (*2d ndarray*) – Autofocused field
- **af\_distance** (*float*) – Autofocusing distance
- *[other]* – Any other objects returned by *minimizer* defined via *minimizer\_kwargs*

**get\_kernel** (*distance*)

Return the kernel for a 1D propagation

**propagate** (*distance*)

Propagate the initial field to a certain distance

**Parameters** **distance** (*float*) – Absolute focusing distance [m]

**Returns** **refocused\_field** – Initial 1D field refocused at *distance*

**Return type** 1d ndarray

**property** **shape**

Shape of the padded input field or Fourier transform

## 4.2 Metrics

`nrefocus.metrics.metric_average_gradient` (*rfi*, *distance*, *roi=None*, *\*kwargs*)

Compute mean average gradient norm of the amplitude

### Notes

The absolute value of the gradient is returned.

`nrefocus.metrics.metric_rms_contrast` (*rfi*, *distance*, *roi=None*, *\*kwargs*)

Compute RMS contrast of the phase

### Notes

The negative angle of the field is used for contrast estimation.

`nrefocus.metrics.metric_spectrum` (*rfi*, *distance*, *roi=None*, *\*\*kwargs*)

Compute spectral contrast

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

`nrefocus.metrics.METRICS = {'average gradient': <function metric_average_gradient>, 'rms c`

Available metrics

## 4.3 Legacy methods

These methods are legacy functions which are kept for backwards-compatibility.

### 4.3.1 Refocusing

`nrefocus.refocus` (*field*, *d*, *nm*, *res*, *method='helmholtz'*, *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*) – Wavelength in pixels
- **method** (*str*) – Defines the method of propagation; one of
  - "helmholtz" : the optical transfer function  $\exp(ik(M-1))$
  - "fresnel" : paraxial approximation  $\exp(ik^2/k)$
- **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  $d$ .

### Notes

This method uses `nrefocus.RefocusNumpy` for refocusing of 2D fields. This is because the `nrefocus.refocus_stack()` function uses `async` which appears to not work with e.g. `pyfftw`.

`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*) – Wavelength in pixels
- **method** (*str*) – Defines the method of propagation; one of
  - “helmholtz” : the optical transfer function  $\exp(ik(M-1))$
  - “fresnel” : paraxial approximation  $\exp(ik^2/k)$
- **num\_cpus** (*int*) – 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  $d$ .

## 4.3.2 Autofocusing

`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
- **ret\_d** (*bool*) – Return the autofocusing distance in pixels. Defaults to False.
  - **ret\_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.*

#### Notes

This method uses `nrefocus.RefocusNumpy` for refocusing of 2D fields. This is because the `nrefocus.refocus_stack()` function uses `async` which appears to not work with e.g. `pyfftw`.

```
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.
  - **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*) – 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
- **same\_dist** (*bool*) – Refocus entire sinogram with one distance.
  - **ret\_ds** (*bool*) – Return the autofocusing distances in pixels. Defaults to False. If `same_dist` is True, still returns autofocusing distances of first pass. The used refocusing distance is the average.
  - **ret\_grads** (*bool*) – Return the computed gradients as a list.
  - **num\_cpus** (*int*) – Number of CPUs to use
  - **copy** (*bool*) – If False, overwrites input array.

**Returns**

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

## CHANGELOG

List of changes in-between nrefocus releases.

### 5.1 version 0.4.3

- ref: deprecate minimizer argument
- ref: legacy minimizer now thinks in SI units
- ref: minor cleanup in autofocusing code

### 5.2 version 0.4.2

- docs: minor improvements

### 5.3 version 0.4.1

- fix: *autofocus* method of Refocus was not functional
- ref: use *Refocus.autofocus* for legacy autofocus method
- docs: fix rtd builds

### 5.4 version 0.4.0

- feat: implement *nrefocus.RefocusPyFFTW* for faster refocusing using *pyfftw* (#11)
- enh: speed-up propagation kernel computation using *numexpr*
- docs: cleanup

## 5.5 version 0.3.1

- dist: include submodules in wheel/dist

## 5.6 version 0.3.0

- feat: introduce `nrefocus.RefocusNumpy` and `nrefocus.RefocusNumpy1D` interface class for user-convenience and efficiency
- docs: cleanup
- ref: new submodule for metrics and metrics now accept a `Refocus` instance as an argument
- ref: new submodule for minimizers and minimizers now accept a `Refocus` instance
- ref: make legacy autofocusing code use the new `Refocus` class

## 5.7 version 0.2.1

- fix: fix several minor bugs (deprecations?) that caused the tests to fail
- ci: migrate to GitHub Actions
- setup: `setup.py` test is deprecated
- docs: refurbish documentation

## 5.8 version 0.2.0

- Drop support for Python 2 (#8)
- Code cleanup

## 5.9 version 0.1.8

- Include docs in sdist

## 5.10 version 0.1.7

- Update documentation and examples



## 5.11 version 0.1.6

- Move documentation from GitHub to readthedocs.io
- Add universal wheel on PyPI
- Update tests on travis with new versions of NumPy

## 5.12 version 0.1.5

- Code cleanup

## 5.13 version 0.1.4

- Padding is now available in all methods (#2)
- Added new convenient submodule *pad*
- Bugfix: autofocusing did not return the correct focusing distance. This resulted in a slight offset in the refocusing distance for the method *autofocus\_stack* when *same\_dist=True* was set.
- New test functions for *pad*



## BILBLIOGRAPHY



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## BIBLIOGRAPHY

- [Goo05] Joseph W. Goodman. *Introduction to Fourier Optics 3rd ed.* Roberts & Company Publishers, 2005.
- [ST91] Bahaa E. A. Saleh and Malvin Carl Teich. *Fundamentals of Photonics*. John Wiley & Sons, Inc., aug 1991.  
[doi:10.1002/0471213748](https://doi.org/10.1002/0471213748).





## PYTHON MODULE INDEX

### n

nrefocus.metrics, [15](#)



## A

`autofocus()` (in module *nrefocus*), 16  
`autofocus()` (*nrefocus.RefocusNumpy* method), 13  
`autofocus()` (*nrefocus.RefocusNumpy1D* method), 14  
`autofocus()` (*nrefocus.RefocusPyFFTW* method), 11  
`autofocus_stack()` (in module *nrefocus*), 17

## G

`get_best_interface()` (in module *nrefocus*), 11  
`get_kernel()` (*nrefocus.RefocusNumpy* method), 13  
`get_kernel()` (*nrefocus.RefocusNumpy1D* method), 14  
`get_kernel()` (*nrefocus.RefocusPyFFTW* method), 12

## M

`metric_average_gradient()` (in module *nrefocus.metrics*), 15  
`metric_rms_contrast()` (in module *nrefocus.metrics*), 15  
`metric_spectrum()` (in module *nrefocus.metrics*), 15  
**METRICS** (in module *nrefocus.metrics*), 15  
**module**  
     *nrefocus.metrics*, 15

## N

*nrefocus.metrics*  
     **module**, 15

## P

`propagate()` (*nrefocus.RefocusNumpy* method), 13  
`propagate()` (*nrefocus.RefocusNumpy1D* method), 14  
`propagate()` (*nrefocus.RefocusPyFFTW* method), 12

## R

`refocus()` (in module *nrefocus*), 15  
`refocus_stack()` (in module *nrefocus*), 16  
*RefocusNumpy* (class in *nrefocus*), 12  
*RefocusNumpy1D* (class in *nrefocus*), 13  
*RefocusPyFFTW* (class in *nrefocus*), 11

## S

`shape()` (*nrefocus.RefocusNumpy* property), 13  
`shape()` (*nrefocus.RefocusNumpy1D* property), 14  
`shape()` (*nrefocus.RefocusPyFFTW* property), 12