# shampoo Documentation

***Release 0.1.dev156***

**Brett Morris**

November 30, 2016

Contents

shampoo is an open source Python package for digital holographic microscopy with SHAMU, the Submersible Holographic Astrobiology Microscope with Ultraresolution. At present, shampoo does holographic reconstruction, and in the near future, shampoo will support specimen detection and tracking.

**Warning:** This code is in development and may break without warning.

# Part I

# Links

- Source code
- Issues
- Docs

# Part II

# General Documentation

CHAPTER 1

# Installation

## 1.1 Requirements

**Note:** Users are strongly recommended to manage these dependencies with the excellent Anaconda Python Distribution which provides easy access to all of the above dependencies and more.

**shampoo** works on Linux, Mac OS X and Windows. It requires Python 3.5 or 2.7 (earlier versions are not supported) as well as the following packages:

- Numpy
- scipy
- Matplotlib
- skimage
- sklearn
- Astropy
- h5py
- pyfftw

### 1.1.1 pyFFTW

shampoo depends on a package called pyfftw for speedy, multithreaded Fourier transforms, which is easy to install on Mac OS X and linux but may be tricky on Windows machines. We recommend that Windows users install pyfftw by doing the following steps via conda:

```
conda install -c salilab fftw
pip install pyfftw
```

## 1.2 Install shampoo

You can install the latest developer version of shampoo by cloning the git repository:

```
git clone https://github.com/bmorris3/shampoo.git
```

...then installing the package with:

```
cd shampoo
python setup.py install
```

## 1.3 Testing

If you want to check that all the tests are running correctly with your Python configuration, start up python, and type:

```
import shampoo
shampoo.test()
```

If there are no errors, you are good to go!

### 1.3.1 More

shampoo follows Astropy's guidelines for affiliated packages–installation and testing for the two are quite similar! Please see Astropy's installation page for more information.

CHAPTER 2

# Getting Started

## 2.1 Simple numerical reconstruction

shampoo comes with a sample hologram of a US Air Force resolution target to reconstruct in the `data` directory. Let's reconstruct it with shampoo.

Let's start in the shampoo repository's top level directory. First, one must import shampoo, and specify the path to the file to reconstruct and the propagation distance:

```
from shampoo import Hologram
hologram_path = 'data/USAF_test.tif'
propagation_distance = 0.03685  # m
```

Now we will import the hologram, which is stored in TIF format, using the `from_tif` method:

```
h = Hologram.from_tif(hologram_path)
```

If your digital holographic microscope has different properties from SHAMU, for example, a different wavelength and pixel size, you will want to set those properties in the `Hologram` constructor, like this:

```
h = Hologram.from_tif(hologram_path, wavelength=650e-6, dx=3.5e-6, dy=3.5e-6)
```

The defaults are set for SHAMU's configuration, so the defaults will work for this example hologram which was recorded by SHAMU.

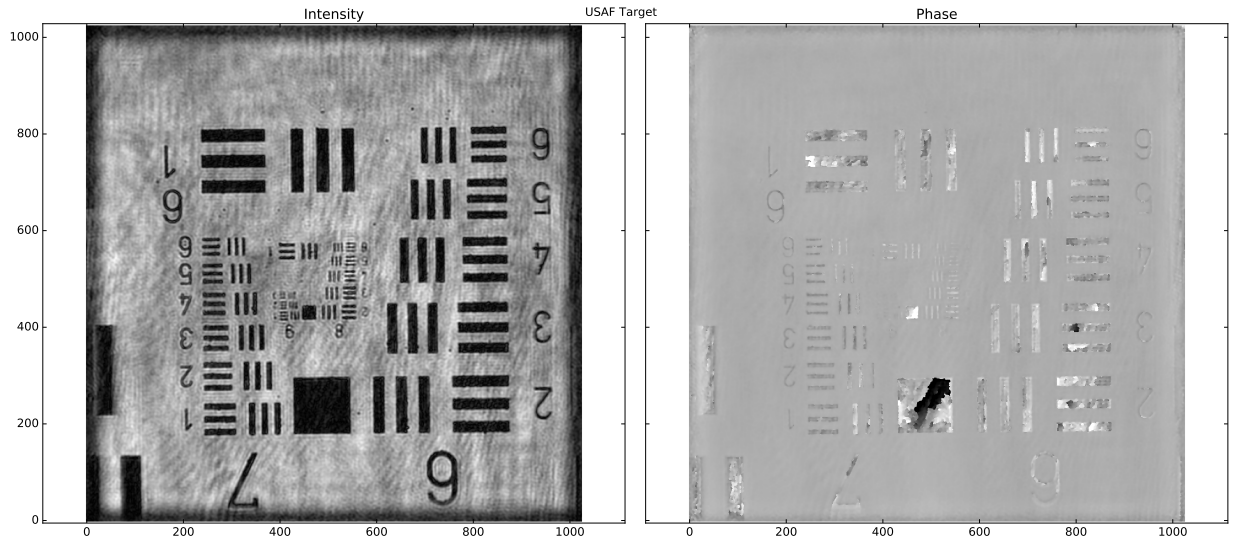
Now let's reconstruct the hologram:

```
wave = h.reconstruct(propagation_distance)
```

That's it! The reconstructed wave is stored within `wave`, which is a `ReconstructedWave` object that stores the 2D complex `ndarray` reconstructed wave, plus has attributes to get the phase and intensity arrays (`phase` and `intensity`), and a plotting function `plot` which we will use to plot the phase and intensity from the reconstructed wave:

```
import matplotlib.pyplot as plt
fig, ax = wave.plot()
fig.suptitle("USAF Target")
fig.tight_layout()
plt.show()
```

Here's the result:




Now you're ready to reconstruct your holograms!

**Warning:** This release should be considered a "preview", as shampoo is still under development.

For more tutorials, see the Tutorials documentation.

*Return to Top*

CHAPTER 3

# Tutorials

We've put together some basic tutorials to show you how shampoo works.

If you're looking for details about specific functions' inputs/outputs, see instead the *Reference/API*.

If you have any trouble reproducing these examples, you are welcome to submit an issue on our GitHub issue tracker, or to submit a pull request with a fix, if you're adventurous.

We currently have the following tutorials:

## 3.1 Reconstruction Tutorials

If you're looking for a quick reference for beginning your first reconstruction, see *Getting Started*. Below we'll dive into some more in-depth examples.

### 3.1.1 Contents



### 3.1.2 Reconstructing multiple z-planes

If you want to reconstruct several z-planes – for example, to find the propagation distance to the USAF test target – you can use the following pattern. First, we will load the raw hologram (only once):

```
from shampoo import Hologram
import numpy as np

hologram_path = 'data/USAF_test.tif'
h = Hologram.from_tif(hologram_path)
```

Next we must set the range of propagation distances to reconstruct. We will use `linspace` to create a linearly-spaced array of five propagation distances:

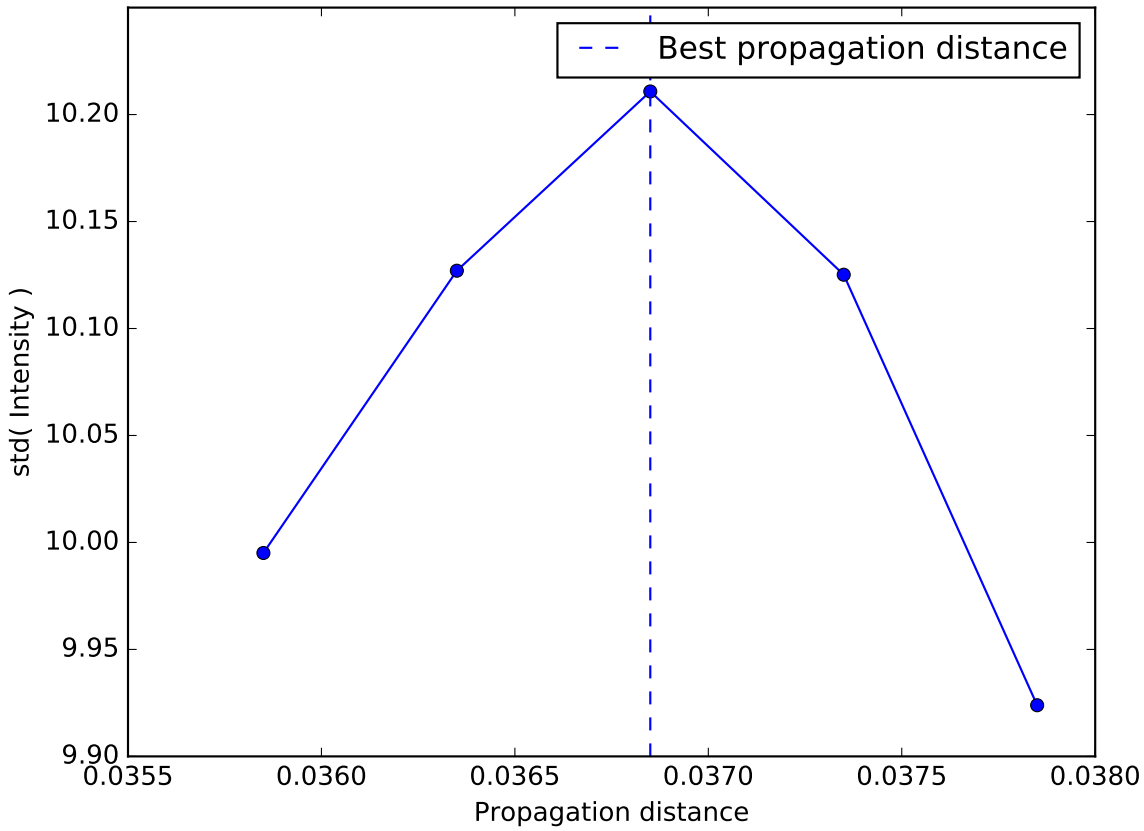
```
n_z_slices = 5
propagation_distances = np.linspace(0.03585, 0.03785, n_z_slices)
```

Then we will loop over the propagation distances, calling `reconstruct` for each one. We will store the reconstructed wave intensities into a data cube called `intensities`:

```
# Allocate some memory for the complex reconstructed waves
intensities = np.zeros((n_z_slices, h.hologram.shape[0], h.hologram.shape[1]),
                       dtype=np.complex128)

# Loop over all propagation distances
for i, distance in enumerate(propagation_distances):

    # Reconstruct at each distance
    wave = h.reconstruct(distance)

    # Save the result to the data cube
    intensities[i, ...] = wave.intensity
```

Now `intensities` contains all of the reconstructed wave intensities, so if we want to find the propagation distance at best focus, we could apply a very crude focus metric to the intensity arrays. For example, the standard deviation of the intensities will be maximal at the propagation distance nearest to the USAF target, so let's plot the standard deviation of the intensities at each propagation distance:

```
# Measure standard deviation within each intensity image
std_intensities = np.std(intensities, axis=(1, 2))

# Initialize a figure object
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.plot(propagation_distances, std_intensities, 'o-')
ax.axvline(propagation_distances[np.argmax(std_intensities)], ls='--',
           label='Best propagation distance')
ax.set(xlabel='Propagation distance', ylabel='std( Intensity )')
ax.legend()
plt.show()
```




You can see that the best propagation distance is near the same distance that we used in the first example (that's why we picked it!).

*Return to Top*

### 3.1.3 Reconstructing a cropped hologram

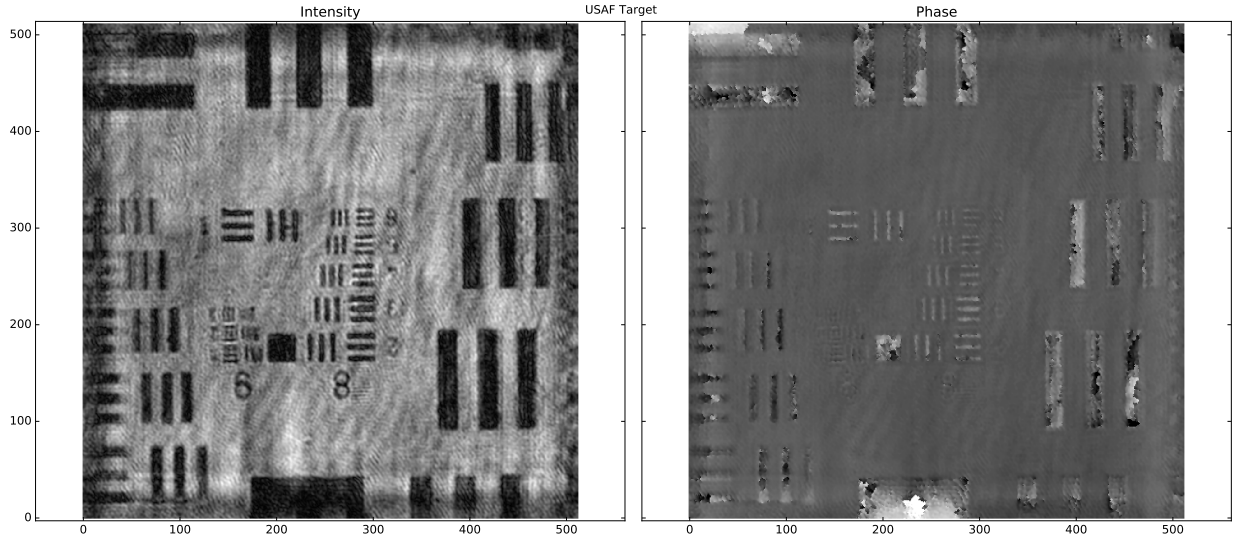
Sometimes you need speed. Sometimes you can afford to reconstruct only the central part the hologram's your field of view if it will save you some time. `Hologram` has a built-in option to help you in this situation.

The keyword argument `crop_fraction` sets the fraction of the original image to crop out. By default it is set to `None`. If you set `crop_fraction=0.5`, a 1024x1024 hologram will be cropped to 512x512 before being reconstructed:

```
# Import package, set hologram path, propagation distance
from shampoo import Hologram
hologram_path = 'data/USAF_test.tif'
propagation_distance = 0.03685  # m

# Construct the hologram object, reconstruct the complex wave
h = Hologram.from_tif(hologram_path, crop_fraction=0.5)
wave = h.reconstruct(propagation_distance)

# Plot the reconstructed phase/intensity
import matplotlib.pyplot as plt
fig, ax = wave.plot()
```

```
fig.suptitle("USAF Target")
fig.tight_layout()
plt.show()
```

Since the implementations of the FFT used by shampoo generally run on order $O(NlogN)$ time, the cropped hologram reconstruction should compute about 2.2x faster than the full-sized one.




*Return to Top*

CHAPTER 4

# Reference/API

## 4.1 shampoo Package

### 4.1.1 Functions

| | |
|---|---|
| cluster_focus_peaks(xyz[, eps, min_samples]) | Use DBSCAN to identify single particles through multiple focus planes. |
| create_hdf5_archive(hdf5_path, ...[, ...]) | Create HDF5 file structure for holograms and phase/intensity reconstructions. |
| fftshift(x[, additional_shift, axes]) | Shift the zero-frequency component to the center of the spectrum, or with some |
| find_focus_plane(roi_cube[, focus_on, plot]) | Find focus plane in a cube of reconstructed waves at different propagation dista |
| glue_focus(xyz, labels) | Launch a glue session with focusing results. |
| locate_specimens(wave_cube, positions, ...) | Identify the (x, y, z) coordinates of a specimen. |
| open_hdf5_archive(hdf5_path) | Load and return a shampoo HDF5 archive. |
| save_scaled_image(image, filename[, margin, ...]) | Save an image to png. |
| test([package, test_path, args, plugins, ...]) | Run the tests using py.test. |
| unwrap_phase(reconstructed_wave[, seed]) | 2D phase unwrap a complex reconstructed wave. |

#### cluster_focus_peaks

shampoo.**cluster_focus_peaks**(*xyz*, *eps=5*, *min_samples=3*)

Use DBSCAN to identify single particles through multiple focus planes.

**Parameters**

**xyz** : ndarray

Matrix of (x, y, z) positions for each peak detected

**eps** : float

Passed to the eps argument of DBSCAN

**min_samples** : int

Passed to the min_samples argument of DBSCAN

**Returns**

**labels** : ndarray

List of cluster labels for each peak. Labels of -1 signify noise points.

### create_hdf5_archive

shampoo.**create_hdf5_archive**(*hdf5_path*, *hologram_paths*, *n_z*, *metadata={}*, *compression=u'lzf'*, *overwrite=False*)

Create HDF5 file structure for holograms and phase/intensity reconstructions.

**Parameters**

**hdf5_path** : string

Name of new HDF5 archive

**hologram_paths** : string

List of all holograms

**n_z** : int

Number of z-stacks to allocate space for

**meta** : dict

Metadata to store with in top-level of the HDF5 archive

**Returns**

**f** : `File`

Opened HDF5 file

### fftshift

shampoo.**fftshift**(*x*, *additional_shift=None*, *axes=None*)

Shift the zero-frequency component to the center of the spectrum, or with some additional offset from the center.

This is a more generic fork of `fftshift`, which doesn't support additional shifts.

**Parameters**

**x** : array_like

Input array.

**additional_shift** : list of length `M`

Desired additional shifts in `x` and `y` directions respectively

**axes** : int or shape tuple, optional

Axes over which to shift. Default is None, which shifts all axes.

**Returns**

**y** : `ndarray`

The shifted array.

### find_focus_plane

shampoo.**find_focus_plane**(*roi_cube*, *focus_on=u'amplitude'*, *plot=False*)

Find focus plane in a cube of reconstructed waves at different propagation distances.

Uses the autofocus method of Dubois et al. 2006 who showed that the integral over the image plane of the amplitude of the reconstructed wave is minimum at the focal plane for a pure amplitude object *[R4]*. This will only work for small cubes centered on the specimen.

**Parameters**

**roi_cube** : ndarray

Reconstructed waves at `N` propagation distances with `M` by `M` pixels, with a shape of `(N, M, M)`

**focus_on** : {"amplitude", "phase"} (optional)

Focus on the phase or amplitude?

**plot** : bool (optional)

Make plots of the integral of the amplitude of the reconstructed wave as a function of distance. Default is False.

**Returns**

**focus_index** : int

Index of the z-plane that is in focus

**significance** : float

Significance of a detection of a particle in the ROI cube, measured by the significance of the negative peak in the derivative of the sum of the unwrapped phase in the image plane, with respect to propagation distance. For example, if `significance < 3` the detection of a specimen may be legitimate.

### glue_focus

shampoo.**glue_focus**(*xyz*, *labels*)

Launch a glue session with focusing results.

**Parameters**

**xyz** : ndarray

Matrix of (x, y, z) positions of particles

**labels** : ndarray

Labels for particles assigned by clustering algorithm

**Returns**

**ga** : GlueApplication

Glue qt GUI application session

### locate_specimens

shampoo.**locate_specimens**(*wave_cube*, *positions*, *labels*, *distances*, *plots=False*)

Identify the (x, y, z) coordinates of a specimen.

**Parameters**

**wave_cube** : ndarray

Cube of reconstructed waves

**positions** : ndarray

(x,y,z) positions of objects detected by the blob finder

**labels** : ndarray

Clustering labels for each (x,y,z) coordinate, identifying groups of positions, i.e., single particles detected at multiple z-planes

**distances** : `ndarray`

Propagation distances, same length as the first axis of `complex_cube`

**Returns**

**specimen_coordinates** : `ndarray`

(x, y, z) coordinates of each detected specimen

**specimen_coordinates** : `ndarray`

Significance of each specimen detection. See docs of `find_focus_plane` for hints on how to interpret the significance quantity.

### open_hdf5_archive

shampoo.**open_hdf5_archive**(*hdf5_path*)

Load and return a shampoo HDF5 archive.

**Parameters**

**hdf5_path** : string

Name of HDF5 archive

**Returns**

**f** : `File`

Opened HDF5 file

### save_scaled_image

shampoo.**save_scaled_image**(*image*, *filename*, *margin=100*, *blobs=None*, *min=0.01*, *max=99.99*)

Save an image to png.

**Parameters**

**image** : `ndarray`

Image to save

**filename** : str

Path to where to save the png file

**blobs** : list or `ndarray` or `None`

(x, y, z) positions

**margin :**

**min** : float

Colormap scaling minimum

**max** : float

Colormap scaling maximum

### test

shampoo.**test**(*package=None*, *test_path=None*, *args=None*, *plugins=None*, *verbose=False*, *pastebin=None*, *remote_data=False*, *pep8=False*, *pdb=False*, *coverage=False*, *open_files=False*, ***kwargs*)

Run the tests using py.test. A proper set of arguments is constructed and passed to pytest.main.

**Parameters**

**package** : str, optional

The name of a specific package to test, e.g. 'io.fits' or 'utils'. If nothing is specified all default tests are run.

**test_path** : str, optional

Specify location to test by path. May be a single file or directory. Must be specified absolutely or relative to the calling directory.

**args** : str, optional

Additional arguments to be passed to pytest.main in the `args` keyword argument.

**plugins** : list, optional

Plugins to be passed to pytest.main in the `plugins` keyword argument.

**verbose** : bool, optional

Convenience option to turn on verbose output from py.test. Passing True is the same as specifying `'-v'` in `args`.

**pastebin** : {'failed','all',None}, optional

Convenience option for turning on py.test pastebin output. Set to `'failed'` to upload info for failed tests, or `'all'` to upload info for all tests.

**remote_data** : bool, optional

Controls whether to run tests marked with @remote_data. These tests use online data and are not run by default. Set to True to run these tests.

**pep8** : bool, optional

Turn on PEP8 checking via the pytest-pep8 plugin and disable normal tests. Same as specifying `'--pep8 -k pep8'` in `args`.

**pdb** : bool, optional

Turn on PDB post-mortem analysis for failing tests. Same as specifying `'--pdb'` in `args`.

**coverage** : bool, optional

Generate a test coverage report. The result will be placed in the directory htmlcov.

**open_files** : bool, optional

Fail when any tests leave files open. Off by default, because this adds extra run time to the test suite. Requires the psutil package.

**parallel** : int, optional

When provided, run the tests in parallel on the specified number of CPUs. If parallel is negative, it will use the all the cores on the machine. Requires the pytest-xdist plugin installed. Only available when using Astropy 0.3 or later.

**kwargs**

Any additional keywords passed into this function will be passed on to the astropy test runner. This allows use of test-related functionality implemented in later versions of astropy without explicitly updating the package template.

### unwrap_phase

shampoo.**unwrap_phase**(*reconstructed_wave*, *seed=42*)

2D phase unwrap a complex reconstructed wave.

Essentially a wrapper around the `unwrap_phase` function. The output will be type float64.

**Parameters**

**reconstructed_wave** : `ndarray`

Complex reconstructed wave

**seed** : float (optional)

Random seed, optional.

**Returns**

`ndarray`

Unwrapped phase image

## 4.1.2 Classes

| | |
|---|---|
| `FFT`(shape, float_precision, complex_precision) | Convenience wrapper around `pyfftw.builders.fft2`. |
| `Hologram`(hologram[, crop_fraction, ...]) | Container for holograms and methods to reconstruct them. |
| `ReconstructedWave`(reconstructed_wave) | Container for reconstructed waves and their intensity and phase arrays. |

### FFT

**class** shampoo.**FFT**(*shape*, *float_precision*, *complex_precision*, *threads=2*)

Bases: `object`

Convenience wrapper around `pyfftw.builders.fft2`.

**Parameters**

**shape** : tuple

Shape of the arrays which you will take the Fourier transforms of.

**float_precision** : `dtype`

**complex_precision** : `dtype`

**threads** : int, optional

This FFT implementation uses multithreading, with two threads by default.

#### Methods Summary

| | |
|---|---|
| `fft2`(array) | 2D Fourier transform. |
| `ifft2`(array) | Inverse 2D Fourier transform. |

#### Methods Documentation

**fft2**(*array*)
2D Fourier transform.

**Parameters**
**array** : ndarray (real)

Input array

**Returns**
**ft_array** : ndarray (complex)

Fourier transform of the input array

**ifft2**(*array*)
Inverse 2D Fourier transform.

**Parameters**
**array** : ndarray

Input array

**Returns**
**ift_array** : ndarray

Inverse Fourier transform of input array

### Hologram

**class** shampoo.**Hologram**(*hologram*, *crop_fraction=None*, *wavelength=4.05e-07*, *rebin_factor=1*, *dx=3.45e-06*, *dy=3.45e-06*, *threads=2*)
Bases: object

Container for holograms and methods to reconstruct them.

**Parameters**
**hologram** : ndarray

Input hologram

**crop_fraction** : float

Fraction of the image to crop for analysis

**wavelength** : float [meters]

Wavelength of laser

**rebin_factor** : int

Rebin the image by factor `rebin_factor`. Must be an even integer.

**dx** : float [meters]

Pixel width in x-direction (unbinned)

**dy** : float [meters]

Pixel width in y-direction (unbinned)

### Notes

Non-square holograms will be cropped to a square with the dimensions of the smallest dimension.

### Methods Summary

| | |
|---|---|
| apodize(array[, alpha]) | Force the magnitude of an array to go to zero at the boundaries. |
| Hologram.detect_specimens | |
| fourier_peak_centroid(fourier_arr[, ...]) | Calculate the centroid of the signal spike in Fourier space near the frequencie |
| Hologram.fourier_trans_of_impulse_resp_func | |
| from_tif(hologram_path, **kwargs) | Load a hologram from a TIF file. |
| get_digital_phase_mask(psi[, plots]) | Calculate the digital phase mask (i.e. |
| real_image_mask(center_x, center_y, radius) | Calculate the Fourier-space mask to isolate the real image |
| reconstruct(propagation_distance[, ...]) | Reconstruct the wave at propagation_distance. |
| Hologram.reconstruct_multithread | |

### Methods Documentation

**apodize**(*array*, *alpha=0.075*)
Force the magnitude of an array to go to zero at the boundaries.

**Parameters**
**array** : ndarray

Array to apodize

**alpha** : float between zero and one

Alpha parameter for the Tukey window function. For best results, keep between 0.075 and 0.2.

**Returns**
**apodized_arr** : ndarray

Apodized array

**fourier_peak_centroid**(*fourier_arr*, *mask_radius=None*, *margin_factor=0.1*, *plot=False*)
Calculate the centroid of the signal spike in Fourier space near the frequencies of the real image.

**Parameters**
**fourier_arr** : ndarray

Fourier-transform of the hologram

**margin_factor** : int

Fraction of the length of the Fourier-transform of the hologram to ignore near the edges, where spurious peaks occur there.

**plot** : bool

Plot the peak-centroiding visualization of the fourier transform of the hologram? Default is False.

**Returns**
**pixel** : ndarray

Pixel at the centroid of the spike in Fourier transform of the hologram near the real image.

**classmethod** **from_tif**(*hologram_path*, ***kwargs*)
Load a hologram from a TIF file.

This class method takes the path to the TIF file as the first argument. All other arguments are the same as `Hologram`.

**Parameters**
**hologram_path** : str

Path to the hologram to load

**get_digital_phase_mask**(*psi*, *plots=False*)
Calculate the digital phase mask (i.e. reference wave), as in Colomb et al. 2006, Eqn. 26 *[R3]*.

Fit for a second order polynomial, numerical parametric lens with least squares to remove tilt, spherical aberration.

**Parameters**
**psi** : `ndarray`

The product of the Fourier transform of the hologram and the Fourier transform of impulse response function

**plots** : bool

Display plots after calculation if `True`

**Returns**
**phase_mask** : `ndarray`

Digital phase mask, used for correcting phase aberrations.

**real_image_mask**(*center_x*, *center_y*, *radius*)
Calculate the Fourier-space mask to isolate the real image

**Parameters**
**center_x** : int

x centroid [pixels] of real image in Fourier space

**center_y** : int

y centroid [pixels] of real image in Fourier space

**radius** : float

Radial width of mask [pixels] to apply to the real image in Fourier space

**Returns**
**mask** : `ndarray`

Binary-valued mask centered on the real-image peak in the Fourier transform of the hologram.

**reconstruct**(*propagation_distance*, *plot_aberration_correction=False*, *plot_fourier_peak=False*, *cache=False*)
Reconstruct the wave at `propagation_distance`.

If `cache` is `True`, the reconstructed wave will be cached onto the `Hologram` object for quick retrieval.

**Parameters**
**propagation_distance** : float

Propagation distance [m]

**plot_aberration_correction** : bool

Plot the abberation correction visualization? Default is False.

**plot_fourier_peak** : bool

Plot the peak-centroiding visualization of the fourier transform of the hologram? Default is False.

**cache** : bool

Cache reconstructions onto the hologram object? Default is False.

**Returns**

**reconstructed_wave** : ReconstructedWave

The reconstructed wave.

## ReconstructedWave

**class** shampoo.**ReconstructedWave**(*reconstructed_wave*)

Bases: object

Container for reconstructed waves and their intensity and phase arrays.

### Attributes Summary

| | |
|---|---|
| intensity | Reconstructed intensity (ndarray, real) |
| phase | Reconstructed, unwrapped phase (ndarray, real) |
| reconstructed_wave | Reconstructed wave (ndarray, complex) |

### Methods Summary

| | |
|---|---|
| plot([phase, intensity, all, cmap]) | Plot the reconstructed phase and/or intensity images. |

### Attributes Documentation

**intensity**

Reconstructed intensity (ndarray, real)

**phase**

Reconstructed, unwrapped phase (ndarray, real)

Phase unwrapping comes from unwrap_phase.

**reconstructed_wave**

Reconstructed wave (ndarray, complex)

### Methods Documentation

**plot**(*phase=False*, *intensity=False*, *all=False*, *cmap=<matplotlib.colors.LinearSegmentedColormap object>*)

Plot the reconstructed phase and/or intensity images.

**Parameters**

**phase** : bool

Toggle unwrapped phase plot. Default is False.

**intensity** : bool

Toggle intensity plot. Default is False.

**all** : bool

Toggle unwrapped phase plot and . Default is False.

**cmap** : `Colormap`

Matplotlib colormap for phase and intensity plots.

**Returns**

**——-**

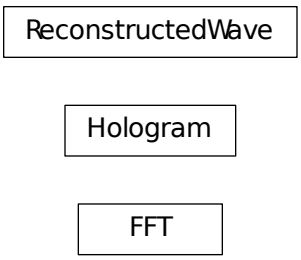
**fig** : `Figure`

Matplotlib figure object

**ax** : `Axes`

Matplotlib axis object

### 4.1.3 Class Inheritance Diagram



# Part III

# Authors

- Brett Morris (UW)

[R4] https://www.osapublishing.org/oe/abstract.cfm?uri=oe-14-13-5895

[R3] http://www.ncbi.nlm.nih.gov/pubmed/16512526

## S