empyer Documentation

Release .170

Carter Francis

Dec 06, 2019

Contents

WARNING! This project is still in beta, feel free to use the project as you wish, however, there are continuing changes being done to the codebase which may result in changes to programs which depend on the project. Additionally there may be spots in the documentation where things are missing, and we are working to fill.

CHAPTER 1

Quick Start

1.1 Philosophy

If you are interested in how Electron Correlation Microscopy (ECM), Angular Correlations (AC) and Fluctuation Electron Microscopy (FEM) can be used to help interpret your data you are in the right place. EMpyer hopes to provide a simple framework which simplifies the process and allows for quick visualization and analysis. Due to the size of many of these experiments EMpyer hopes to scale with the needs of whoever is using the software, weather it be analyzing one image on a desktop to dealing with terabytes of data on a cluster.

Maintained by the Voyle's Group at the University of Wisconsin Madison, if you have any questions/ bugs to report send an email to csfrancis@wisc.edu or submit an issue on github.

Much of the visualization tools are extensions of hyperspy, which has pretty extensive documentation. As a result it is probably worthwhile to check out their documentation to see if you can find your answer there.

We strive for an easy way to look at and understand your data. While each of these techniques alone provide some measure of information about some material, we hope that packaging them all together they are easily accessible.

1.2 Quickstart

Welcome to the Quick Start Page!

Empyer is an extension of the hyperspy package. It provides additional functionality related to analyzing 4 and 5 dimensional data sets. Especially STEM diffraction patterns from metallic glasses.

Downloading EMpyer is easy. You can download the latest version of EMpyer from PyPi using pip.

```
$pip install empyer
```

Assuming you are looking at Diffraction patterns from a STEM you can easily view the data by just sending the plot command to a *diffraction_signal* object. Utilizing the plotting and loading abilities from hyperspy the signal will be shown. The norm= 'log' command just plots on a logarithmic scale. Because Empyer is a registered extension of hyperspy just by importing hyperspy you will load all the additional functionality of Empyer.

```
import hyperspy.api
import matplotlib.pyplot as plt

dif_signal = hs.load(file, signal_type ='diffraction_signal')
dif_signal.plot(norm='log')
plt.show()
```

1.3 Analyzing Angular Correlations

While angular correlations can be easily (and quickly calculated) they are highly susceptible to even small misalignment in the microscope. Additionally, thick samples which are more dominated by dynamical scattering, render the data collected largely useless.

Starting with the equation for the angular correlation is given by:

$$C(\phi,k,n) = \frac{< I(\theta,k,n) * I(\theta+\phi,k,n) >_\theta - < I(\theta,k,n) >^2}{< I(\theta,k,n) >^2}$$

Where θ is the entire 2π radians the that ϕ (the angle of correlation) is averaged over. k is the radius of the reciporical space vector and n is the diffraction pattern number (assuming the correlation is being calculated for a series of diffraction patterns)

While the Electron Microscope community has decided to use the terminology of angular correlations, what is being calculated is in actuality the self-correlation as a function of angle instead of time.

As result efficient methods for convolution can be applied in which the actual computation occurring is:

$$C(\phi,k,n) = \frac{IFFT[FFT(I(\theta,k,n))_\theta * Conj(FFT(I(\theta,k,n)))_\theta]}{< I(\theta,k,n) >^2}$$

Which speeds up the calculation more than 100x

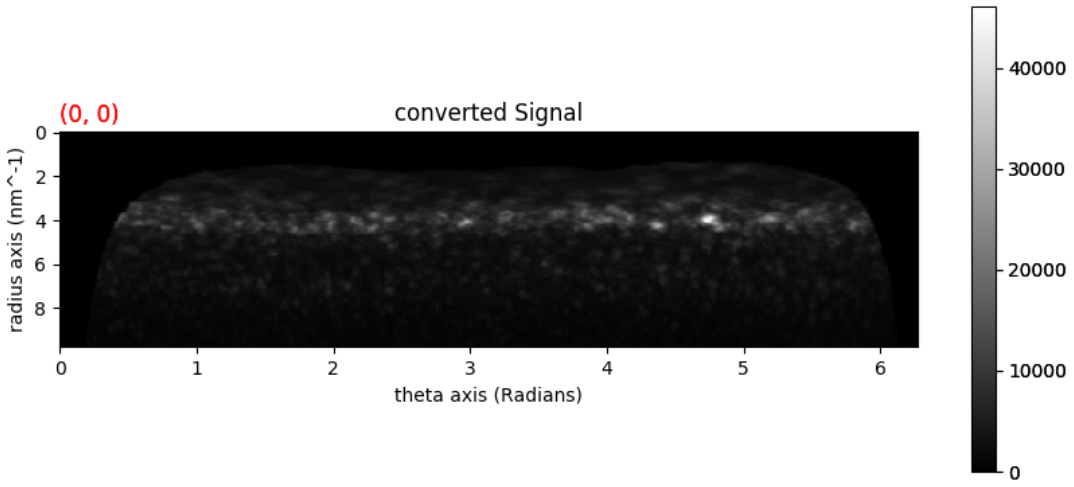
Calculating the Angular Correlations

In order to calculate angular correlations, start by loading a 4-D data set.

```
import hyperspy.api as hs
import matplotlib.pyplot as plt

dif_signal = hs.load(file, signal_type ='diffraction_signal')

# adding a mask to the signal for to block the beam stop
dif_signal.mask_below(.1)
dif_signal.show_mask()

# correcting for not elliptical diffraction patterns. Make sure there is not wobbling␣
↪from pattern to pattern
dif_signal.determine_ellipse()
pol_signal = dif_signal.calculate_polar_spectrum()
ang_signal = polar_signal.autocorrelation()
pow_signal = ang_signal.get_power_spectrum()
```

1.4 Advanced Guide

This section goes further into the methods for correcting and optimizing your data. While ideally most of these functions would work with minimal human interaction the reality of the situation is that whatever can go wrong will



inevitably go wrong.

So the purpose of this tutorial is to show some of the most common errors and problems with analysis and how to solve these problems.

Advanced Loading

Starting at the very beginning, the first thing you need to figure out the proper way to load your signal. While hyperspy is very good at loading a variety of signals. Sometimes there is a bit of difficultly in loading the data in a way that makes sense. I have wrote some scripts which load emi, tiff and .mrc files. While they might not be incredibly useful they are a good start for how to efficiently package microscope outputs into .hdf5 files.

Most of the work he is setting up your axes correctly, which while not the hardest thing to do, will save many headaches down the road.

Advanced Image Registration

In most cases the image resolution of the diffraction patterns is both a blessing and a curse. On one hand higher resolution gives a more accurate image and position, but on the other hand, higher resolutions means that for every distortion in a diffraction pattern there is a loss of intensity.

CHAPTER 2

Contents

2.1 empyer package

2.1.1 Subpackages

empyer.io module

empyer.io.**load**(*filenames=None*, *signal_type=None*, *stack=False*, *stack_axis=None*, *new_axis_name='stack_element'*, *lazy=False*, ***kwds*)

Extends the hyperspy loading functionality with additional ability to load empyer signals.

Parameters

- **filenames** (*list*) – The list of file paths or single file to be loaded
- **signal_type** (*str*) – The type of signal to be loaded. Can be read from file or given
- **stack** (*bool*) – Stack creates a new axis and loads a list of files into one signal
- **stack_axis**
- **new_axis_name** (*str*) – The name of the new axis created with stack
- **lazy** (*bool*) – Load the signal into memory or load chunks. May effect performance, but useful for large datasets.

Returns A signal of type signal_type

Return type signal

empyer.io.**to_correlation_signal**(*signal=None*)

Hyperspy signal to correlation_signal

Parameters **signal** (*Signal2D*)

Returns **ds** – A signal of type signal_type

Return type em_signal

empyer.io.**to_diffraction_signal**(*signal=None*)
Hyperspy signal to diffraction_signal

Parameters **signal** (*Signal2D*)

Returns **ds** – A signal of type signal_type

Return type em_signal

empyer.io.**to_em_signal**(*signal=None*)
Hyperspy signal to em_signal

Parameters **signal** (*Signal2D*)

Returns **ds** – A signal of type signal_type

Return type em_signal

empyer.io.**to_polar_signal**(*signal=None*)
Hyperspy signal to polar_signal

Parameters **signal** (*Signal2D*)

Returns **ds** – A signal of type signal_type

Return type em_signal

empyer.io.**to_power_signal**(*signal=None*)
Hyperspy signal to power_signal

Parameters **signal** (*Signal2D*)

Returns **ds** – A signal of type signal_type

Return type em_signal

empyer.signals

Submodules

empyer.signals.em_signal module

class empyer.signals.em_signal.**EMSignal**(**args*, ***kwargs*)
Bases: hyperspy._signals.signal2d.Signal2D

The Diffraction Signal class extends the Hyperspy 2d signal class

add_haadf_intensities(*intensity_array*, *slope*, *intercept*)
Add High Angle Annular Dark Field intensities for each point.

Parameters **intensity_array** (*nd array*) – An intensity array which is the same size of the navigation axis. Acts as a measure of the thickness if there is a calculated normalized intensity. For masking in real space. Matches data input NOT the real space coordinates from Hyperspy. (To match Hyperspy use np.transpose on intensity array)

add_mask()

as_lazy(**args*, ***kwargs*)

get_signal_axes_values()
Returns the values for each pixel of the signal. Useful for plotting without using hyperspy

Returns

- **axis0** (*array-like*) – The values for axis 0
- **axis1** (*array-like*) – The values for axis 1

get_thicknesses()

mask_above(*value*, *unmask=False*)
Applies a mask to every pixel with a value below some value

Parameters

- **value** (*float*) – The minimum pixel value to apply a mask to.
- **unmask** (*bool*) – Unmask any pixel with a value above value

mask_below(*value*, *unmask=False*)
Applies a mask to every pixel with an average value below value

Parameters

- **value** (*float*) – The maximum pixel value to apply a mask to.
- **unmask** (*bool*) – Unmask any pixel with a value below value

mask_border(*pixels=1*)

mask_circle(*center*, *radius*, *unmask=False*)
Applies a mask to every pixel using a shape and the appropriate definition

Parameters

- **shape** (*str*) – Acceptable shapes [‘rectangle, ‘circle’]
- **data** (*list*) – Define shapes. eg ‘rectangle’ -> [x1,x2,y1,y2] ‘circle’ -> [radius, x,y] data allows indexing with floats and the axes described for the signal
- **unmask** (*bool*) – Unmask any pixels in the defined shape

mask_where(*condition*)
Mask at some condition

condition: array_like Masking condition

reset_mask()

set_axes(*index*, *name=None*, *scale=None*, *units=None*, *offset=None*)
Set axes of the signal

Parameters

- **index** (*int*) – The index of the axes
- **name** (*str*) – The name of the axis
- **scale** (*float*) – The scale fo the axis
- **units** (*str*) – The units of the axis
- **offset** (*float*) – The offset of the axes

thickness_filter()
Filter based on HAADF intensities

Returns

- **th_filter** (*array-like*) – Integers which are used to filter into different thicknesses. Basically used to bin the signal
- **thicknesses** (*1-d array*) – The thicknesses for the signal at every integer.

class empyer.signals.em_signal.**LazyEMSignal**(**args*, ***kwargs*)
Bases: hyperspy._signals.lazy.LazySignal, *empyer.signals.em_signal.EMSignal*

class empyer.signals.em_signal.**MaskPasser**(*s*, *sl*, *nav*)
Bases: object

mask_above(*minimum*)
Mask above the minimum value

minimum: float Mask any values in the slice above the minimum value

mask_below(*maximum*)
Mask below the max value

maximum: float Mask any values in the slice below the maximum value

mask_circle(*center*, *radius*, *unmask=False*)
Applies a mask to every pixel using a shape and the appropriate definition

Parameters

- **center** (*tuple*) – The (x,y) center of the circle
- **radius** (*float or int*) – The radius of the circle
- **unmask** (*bool*) – Unmask any pixels in the defined shape

mask_where(*condition*)
Mask at some condition

condition: array_like Masking condition

class empyer.signals.em_signal.**MaskSlicer**(*obj*, *isNavigation*)
Bases: hyperspy.misc.slicing.SpecialSlicers

Expansion of the Special Slicer class. Used for applying a mask

empyer.signals.diffraction_signal module

class empyer.signals.diffraction_signal.**DiffractionSignal**(**args*, ***kwargs*)
Bases: *empyer.signals.em_signal.EMSignal*

The Diffraction Signal class extends the Hyperspy 2d signal class This class name should be changed. . . .

as_lazy(**args*, ***kwargs*)
Returns the signal as a lazy signal.

calculate_polar_spectrum(*phase_width=720, radius=[0, -1], parallel=False, inplace=False, segments=None, num_points=500*)
Take the Diffraction Pattern and unwrap the diffraction pattern.

Parameters

- **phase_width** (*int*) – The number of pixels in the x direction
- **radius** (*int*) – The number of pixels in the y direction
- **parallel** (*boolean*) – use multiple processors for calculations (useful for large numbers of diffraction patterns, more for large pixel size)
- **inplace** (*boolean*) – replaces diffraction pattern data with polar equivalent

Returns **polar** – Polar signal returned

Return type *PolarSignal*

determine_ellipse(*num_points=500*, *suspected_radius=None*, *interactive=False*, *plot=False*)
Determine the elliptical nature of the diffraction pattern.

Parameters

- **interactive** (*Boolean*) – 'interactive' nature means that points are chosen to create a ring
- **axis** (*int*) – 'axis' to determine ellipse along
- **num_points** (*int*) – number of points to define ellipse by (only used if interactive = False)
- **plot** (*Boolean*) – Weather or not to plot the ellipse

Returns

- **center** (*list of int*) – the center of the ellipse
- **lengths** (*list of int*) – the length in pixels of the major and minor axes
- **angle** (*float*) – the angle of the major axes

get_darkfield_image(*position*, *radius=0.5*)
Creates a dark-field image from an artifical appature at some position with some radius. Allows for decimial spacing.

Parameters

- **position** (*tuple*) – The position of the circle to create the darkfield image
- **radius** (*float*) – The radius of the circle for the dark-field image

Returns **darkfield_image**

Return type Signal2D

class empyer.signals.diffraction_signal.**LazyDiffractionSignal**(**args*, ***kwargs*)
Bases: hyperspy._signals.lazy.LazySignal, *empyer.signals.diffraction_signal.DiffractionSignal*

empyer.signals.polar_signal module

class empyer.signals.polar_signal.**LazyPolarSignal**(**args*, ***kwargs*)
Bases: hyperspy._signals.lazy.LazySignal, *empyer.signals.polar_signal.PolarSignal*

class empyer.signals.polar_signal.**PolarSignal**(**args*, ***kwargs*)
Bases: *empyer.signals.em_signal.EMSignal*

as_lazy(**args*, ***kwargs*)

autocorrelation(*binning_factor=1*, *cut=0*, *normalize=True*)
Create a Correlation Signal from a numpy array.

Parameters

- **binning_factor** (*int*) – Binning factor to speed up calculations
- **cut** (*int or float*) – The number of pixels or distance to cut off image
- **normalize** (*boolean*) – normalize with autocorrelation

Returns **angle**

Return type *CorrelationSignal*

correlation_lengths()
Calculates the average correlation length across the sample

fem(*version='omega'*, *indicies=None*)
Calculated the variance among some image

Parameters

- **version** (*str*) – The name of the FEM equation to use. 'rings' calculates the mean of the variances of all the patterns at some k. 'omega' calculates the variance of the annular means for every value of k.
- **patterns** (*indicies*) – Calculates the FEM pattern using only some of the patterns based on their indexes

empyer.signals.correlation_signal module

class empyer.signals.correlation_signal.**CorrelationSignal**(**args*, ***kwargs*)
Bases: *empyer.signals.em_signal.EMSignal*

Create a Correlation Signal from a numpy array.

Parameters

- **data** (*numpy array*) – The signal data. It can be an array of any dimensions.
- **axes** (*dictionary (optional)*) – Dictionary to define the axes (see the documentation of the AxesManager class for more details).
- **attributes** (*dictionary (optional)*) – A dictionary whose items are stored as attributes.
- **metadata** (*dictionary (optional)*) – A dictionary containing a set of parameters that will to stores in the *metadata* attribute. Some parameters might be mandatory in some cases.
- **original_metadata** (*dictionary (optional)*) – A dictionary containing a set of parameters that will to stores in the *original_metadata* attribute. It typically contains all the parameters that has been imported from the original data file.

as_lazy(**args*, ***kwargs*)
Returns the signal as a lazy signal.

get_power_spectrum(*method='FFT'*)
Calculate a power spectrum from the correlation signal

Parameters **method** (*str*) – 'FFT' gives fourier transformation of the angular power spectrum. Currently the only method available

get_summed_power_spectrum()
Returns the power spectrum from the summed correlation signal.

class empyer.signals.correlation_signal.**LazyCorrelationSignal**(**args*, ***kwargs*)
Bases: hyperspy._signals.lazy.LazySignal, *empyer.signals.correlation_signal.CorrelationSignal*

empyer.signals.power_signal module

class empyer.signals.power_signal.**LazyPowerSignal**(**args*, ***kwargs*)
Bases: hyperspy._signals.lazy.LazySignal, *empyer.signals.power_signal.PowerSignal*

class empyer.signals.power_signal.**PowerSignal**(**args*, ***kwargs*)
Bases: *empyer.signals.em_signal.EMSignal*

as_lazy(**args*, ***kwargs*)
Returns the signal as a lazy signal.

get_i_vs_k(*symmetry=None*)
Get the intensity versus k for the summed diffraction patterns

Parameters **symmetry** (*int or array-like*) – specific integers or list of symmetries to average over when creating the map of the correlations.

Returns **i** – The intensity as a function of k for some signal

Return type Signal-2D

get_map(*k_region=[3.0, 6.0], symmetry=None*)
Creates a 2 dimensional map of from the power spectrum.

Parameters

- **k_region** (*array-like*) – upper and lower k values to integrate over, allows both ints and floats for indexing
- **symmetry** (*int or array-like*) – specific integers or list of symmetries to average over when creating the map of the correlations.

Returns **symmetry_map** – 2 dimensional map of from the power spectrum

Return type 2-d array

plot_symmetries(*k_region=[3.0, 6.0], symmetry=[2, 4, 6, 8, 10], *args, **kwargs*)
Plots the symmetries in the list of symmetries. Plot symmetries takes all of the arguements that imshow does.

Parameters **k_region** (*array-like*) – upper and lower k values to integrate over, allows both ints and floats for indexing

symmetry: list specific integers or list of symmetries to average over when creating the map of the correlations.

empyer.misc

Submodules

empyer.misc.angular_correlation module

empyer.misc.angular_correlation.**angular_correlation**(*r_theta_img*, *mask=None*, *binning=1*, *cut_off=0*, *normalize=True*)
A program that takes a 2d image and then preforms an angular correlation on the image. :Parameters: * **r_theta_img** (*array_like*) – The index of the axes

- **mask** (*boolean array*) – The name of the axis
- **binning** (*int*) – binning factor
- **cut_off** (*int*) – The cut off in pixels to
- **normalize** (*bool*) – Subtract <I(heta)>^2 and divide by <I(heta)>^2

empyer.misc.angular_correlation.**get_S_Q**(*r_theta_img*, *plot=False*)
Get the S of Q for the images.

Parameters **r_theta_img**

empyer.misc.angular_correlation.**power_spectrum**(*correlation*, *method='FFT'*)
Take the power spectrum for some correlation. Takes the FFT of the correlation

Parameters

- **correlation** (*array-like*) – Taking the FFT of the angular correlation to find the symmetry present
- **method** (*str ("FFT")*) – Right now this doesn't actually do anything but I want to add in other methods.

Returns **pow_spectrum** – The resulting power spectrum from the angular correlation. Gives indexes 0-180.

Return type array-like

empyer.misc.cartesain_to_polar module

empyer.misc.cartesain_to_polar.**convert**(*img, center=None, angle=None, lengths=None, radius=[0, 100], phase_width=720*)
Function for converting an image in cartesian coordinates to polar coordinates.

Parameters

- **img** (*array-like*) – A n by 2-d array for the image to convert to polar coordinates
- **center** (*list*) – [X,Y] coordinates for the center of the image
- **angle** (*float*) – Angle of rotation if the sample is elliptical
- **lengths** (*list*) – The major and minor lengths of the ellipse
- **radius** (*list*) – The inner and outer indexes to define the radius by.
- **phase_width** (*int*) – The number of "pixels" in the polar image along the x direction

Returns **polar_img** – A numpy array of the input img in polar coordiates. Dim (radius[1]-radius[0]) x phase_width

Return type array-like

empyer.misc.ellipse_analysis module

empyer.misc.ellipse_analysis.**advanced_solve_ellipse**(*img*, *center*, *lengths*, *angle*, *phase_width*, *radius*, *num_points=500*)
This is a method in development. Better at optimizing angular correlations

This method is in development. Due to the fact that an ellipse will have maximum 2-fold symmetry when the center is correctly determined and maximum 2*n-fold symmetry when the major/ minor axes as well as the angle of rotation is correct. This algorithm optimizes for these quantities.

img: Array-like 2-d Array of some image

range: list The lower and upper limits to look at. Usually should just look at the first ring.

num_points: int The number of points to look at to determine the characteristic ellipse.

center: list The x and y coordinates of the center

lengths: list The major and minor axes

angle: float The angle of rotation for the ellipse

empyer.misc.ellipse_analysis.**get_max_positions**(*image*, *num_points=None*, *radius=None*)

empyer.misc.ellipse_analysis.**invcot**(*val*)

empyer.misc.ellipse_analysis.**solve_ellipse**(*img*, *interactive=False*, *num_points=500*, *plot=False*, *suspected_radius=None*)

Takes a 2-d array and allows you to solve for the equivalent ellipse. Everything is done in array coord.

Fitzgibbon, A. W., Fisher, R. B., Hill, F., & Eh, E. (1999). Direct Least Squres Fitting of Ellipses. IEEE Transactions on Pattern Analysis and Machine Intelligence, 21(5), 476–480.

http://nicky.vanforeest.com/misc/fitEllipse/fitEllipse.html

Parameters

- **img** (*array-like*) – Image with ellipse with more intense
- **interactive** (*bool*) – Allows you to pick points for the ellipse instead of taking the top 2000 points
- **plot** (*bool*) – plots the unwrapped image as well as a super imposed ellipse

Returns

- **center** (*array-like*) – In cartesian coordinates or (x,y)!!!!!! arrays are in y,x
- **lengths** (*array-like*) – The 'length' in pixels of the major and minor axis
- **angle** (*float*) – In radians based on the major axis

empyer.misc.fem module

empyer.misc.fem.**fem**(*r_theta_imgs*, *version='omega'*, *binning=1*, *cut=40*)

Calculated the variance among some image

Parameters

- **r_theta_imgs** (*array_like*) – polar images
- **version** (*str*) – The name of the FEM equation to use
- **binning** (*int*) – binning factor
- **cut** (*int*) – The cut off in pixels to not consider

empyer.misc.image module

empyer.misc.image.**bin_2d**(*image*, *binning_factor*)

Binning a 2-dimensional image by some factor

Parameters

- **image** (*2-d array*)
- **binning_factor** (*int*)

Returns **new_image**

Return type 2-d array

empyer.misc.image.**cartesian_list_to_polar**(*x_list*, *y_list*, *center*)
A function that converts a list of x,y coordinates to (r,theta)

Parameters

- **r** (*float*) – radius
- **theta** (*float*) – angle
- **center** (*array*) – center of the array [x,y]

Returns

- **theta_list** (*2-d array*)
- **r_list** (*2-d array*)

empyer.misc.image.**cartesian_to_polar**(*x*, *y*, *center*)
A function that converts the x,y coordinates to polar ones. -Does not do the circular correction

empyer.misc.image.**create_grid**(*dimension1*, *dimension2*)

Parameters

- **dimension1** (*array*)
- **dimension2** (*array*)

Returns

- **a** (*array*)
- **b** (*array*)

empyer.misc.image.**distort**(*image*, *center*, *angle*, *lengths*)
Takes an image and distorts the image based on an elliptical distortion

Parameters

- **image** (*array-like*) – The image to apply the elliptical distortion to
- **center** (*list*) – The center of the ellipse
- **angle** (*float*) – The angle of the major axis in radians
- **lengths** (*The lengths of the major and minor axis of the ellipse*)

Returns **distorted** – The elliptically distorted image

Return type array-like

empyer.misc.image.**ellipsoid_list_to_cartesian**(*r_list*, *theta_list*, *center*, *axes_lengths=None*, *angle=None*)
Takes a list of ellipsoid points and then use then find their cartesian equivalent

Parameters

- **r_list** (*array*) – list of all of the radius. Can either be all values or even_spaced
- **theta_list** (*array*) – list of all of the radius. Can either be all values or even_spaced
- **center** (*array_like*) – center of the ellipsoid
- **lengths** (*float*) – length of the major axis
- **minor** (*float*) – length of the minor axis
- **angle** (*float*) – angle of the major axis in radians

Returns

- **x_list** (*array_like*) – list of x points
- **y_list** (*array_like*) – list of y points

empyer.misc.image.**flatten_axis**(*array*, *axis*)

empyer.misc.image.**polar_list_to_cartesian**(*r_list*, *theta_list*, *center*)

Parameters

- **r_list** (*array_like*) – radius
- **theta_list** (*array_like*) – angle
- **center** (*array*) – center of the array [x,y]

Returns

- **x_list** (*array*)
- **y_list** (*array*)

empyer.misc.image.**polar_to_cartesian**(*r*, *theta*, *center*)

A function that converts polar (r,theta) coordinates to cartesian(x,y) ones

Parameters

- **r** (*float*) – radius
- **theta** (*float*) – angle
- **center** (*array*) – center of the array [x,y]

Returns

- **x** (*float*)
- **y** (*float*)

empyer.misc.image.**random_ellipse**(*num_points*, *center*, *foci*, *angle*)

empyer.misc.image.**rotate**(*x*, *y*, *angle*)

empyer.misc.image.**square**(*array*)

Module contents

empyer.simulate

Module contents

empyer.tests

Subpackages

empyer.tests.io

Submodules

empyer.tests.io.test_io module

class empyer.tests.io.test_io.**TestIOSignal**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_save_and_load()

test_to_correlation_signal()

test_to_diffraction_signal()

test_to_polar_signal()

test_to_power_signal()

Module contents

empyer.tests.misc

Submodules

empyer.tests.misc.test_angular_correlation module

class empyer.tests.misc.test_angular_correlation.**TestBinning**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_angular_correlation()

test_angular_correlation_mask()

test_power_spectrum()

empyer.tests.misc.test_conversion module

class empyer.tests.misc.test_conversion.**TestConvert**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_2d_convert()

empyer.tests.misc.test_ellipse_analysis module

class empyer.tests.misc.test_ellipse_analysis.**TestConvert**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_get_max_coords()

test_solve_ellipse()

test_solve_ellipse_mask()

test_solve_one_ellipse()

empyer.tests.misc.test_image module

class empyer.tests.misc.test_image.**TestBinning**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_binning()

class empyer.tests.misc.test_image.**TestConversions**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_ellipse_conversion()

test_rand_ellipse()

Module contents

empyer.tests.signal package

Submodules

empyer.tests.signal.test_correlation_signal module

class empyer.tests.signal.test_correlation_signal.**TestPolarSignal**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_lazy()

test_power_spectrum()

test_summed_power_spectrum()

empyer.tests.signal.test_diffraction_signal module

class empyer.tests.signal.test_diffraction_signal.**TestDiffractionSignal**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_conversion()

test_conversion_and_mask()

test_ellipse()

test_parallel_conversion()

class empyer.tests.signal.test_diffraction_signal.**TestSegmentedDiffractionSignal**(*methodName=*
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_lazy()

test_seg()

empyer.tests.signal.test_em_signal module

class empyer.tests.signal.test_em_signal.**TestEMSignal**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_HAADF()

test_HAADF_mask()

test_lazy_signal()

test_mask_above()

test_mask_below()

test_mask_circle_slice()

test_mask_slicing()

test_mask_slicing2()

test_slice_mask_above()

test_slice_mask_above2()

test_slice_mask_below()

test_slice_mask_below2()

test_slice_mask_where()

empyer.tests.signal.test_polar_signal module

class empyer.tests.signal.test_polar_signal.**TestPolarSignal**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_autocorrelation()

test_autocorrelation_mask()

test_fem_omega()

test_fem_rings()

test_fem_with_filter()

test_lazy()

empyer.tests.signal.test_power_signal module

class empyer.tests.signal.test_power_signal.**TestPowerSignal**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Hook method for setting up the test fixture before exercising it.

test_get_map()

test_i_vs_k()

test_lazy()

test_plot_maps()

Module contents

Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index

F

G

I

L

M

P

R

S

T