HeliQ

Wouter Heyvaert, Sara Bals, Wiebke Albrecht

Apr 19, 2022

CONTENTS:

CHAPTER

ONE

INSTALLATION

HeliQ is available on PyPI and can be installed using:

```
pip install heliq
```

CHAPTER

TWO

DATA ALIGNMENT

To calculate the helicity of a certain object, it is important to make sure that the helical axis is aligned to the z-axis and is centered in the volume. This can be done by finding the center and orientation of your object and entering that information in `align_helical_axis`. That function will rotate and translate your data such that the given orientation becomes parallel with the z-axis and is centered in the volume. You can find the orientation of the helical axis and the center either manually using external 3D visualization software, or for relatively simple objects these can also be found automatically using `center_of_mass` and/or `helical_axis_pca`. The former assumes that the helical axis goes through the center of mass of you object and will therefore select the center of mass as the center of the object. The latter assumes that the object is longer in one direction compared to the other directions, and will use this direction as the helical axis. Don't use these functions if these assumptions are not correct for your object!

align_helical_axis(*data: numpy.ndarray*, *orientation: Sequence[float]*, *center: Sequence[float]*) → numpy.ndarray

Transform a 3D object to align its helical axis to the z-axis.

Parameters

- **data** – A 3D array containing the shape of a (helical) object. The shape of the array should correspond to the (y, x, z) axes.
- **orientation** – A vector that points in the direction of the helical axis. The vector should be a numpy array, tuple or list with three components (x, y, z).
- **center** – The location of the center of the object represented as a nnumpy array, tuple or list with components (x, y, z). This is typically the center of mass of the object.

Returns The transformed data. The helical axis of this object is parallel with the z-axis and is positioned in the center of the array in each dimension.

center_of_mass(*data: numpy.ndarray*) → numpy.ndarray

Calculate the center of mass of a 3D object.

Parameters **data** – A 3D array containing the shape of an object. The shape of the array should correspond to the (y, x, z) axes.

Returns An array of 3 elements (x, y, z) of the center of mass of the object.

helical_axis_pca(*data: numpy.ndarray*, *threshold: float*) → numpy.ndarray

Estimate the orientation of the helical axis of a 3D object using PCA.

The helical axis is assumed to be along the longest direction of the object. This is often true for rods, wires or similar shapes, but is not guaranteed to be the case. The longest direction can be detected using principal component analysis (PCA) by binarizing it using a certain threshold value. A list will then be made with the (x, y, z) coordinates of all the voxels that are considered part of the object, i.e. all the voxels with an intensity that is larger than the threshold value. PCA is executed on this list of coordinates such that the component with the largest score will be oriented in the direction along which the points are spread the most. This is the direction along which the object is the longest.

Parameters

- **data** – A 3D array containing the shape of an object. The shape of the array should correspond to the (y, x, z) axes.
- **threshold** – The threshold that will be used to binarize the data. Only voxels with a value larger than this threshold will be used.

Returns An array of 3 elements (x, y, z) of the orientation of the estimated helical axis.

CHAPTER

THREE

HELICITY QUANTIFICATION

The helicity of an objct can be analyzed in different ways. The main method presented in our paper is to calculate the helicity function (`helicity_function`), which gives the helicity as a function of the distance between the helical axis and the inclination angle. It is also possible to get the total helicity (`total_helicity`), which is a single value between -1 and +1 that indicates the total helicity of an object based on the results of the helicity function.

If you want to analyze the distribution of helical features across an object, you can use `helicity_map`, which returns an array that indicates the local helicity around each voxel. Use `helicity_descriptor` for more controll about how to calculate this helicity map, this function is internally used by `helicity_function` and `helicity_descriptor` and returns two arrays: one array containing the magnitude of helicity (i.e. the gradient magnitude) in each voxel and one array containing the inclination angle. That way, you can select for example only the parts of your object that have a certain inclination angle.

class HelicityFunction(*delta_alpha: float*, *delta_rho: float*, *histogram: numpy.ndarray*)
Holds the contents of a helicity function.

helicity_descriptor(*data: numpy.ndarray*, *kernel_size: int = 5*) → Tuple[numpy.ndarray, numpy.ndarray]
Calculate the inclination angle and gradient magnitude in each voxel.

This is not a measure of helicity itself, but is used as the first step in most other methods. It can however be used to analyze the helicity of a particle with a custom method.

Parameters

- **data** – The voxel data as a 3D array that describes the object.
- **kernel_size** – The size of the Sobel kernel used for calculating the gradient, should be an odd integer.

Returns A tuple of two elements, the first is the normalized gradient magnitude in each voxel and the second element is the inclination angle in degrees in each voxel.

helicity_function(*data: numpy.ndarray*, *voxel_size: float*, *delta_alpha: float = 1.0*, *delta_rho: Optional[float] = None*, *kernel_size: int = 5*) → *heliq.helicity.HelicityFunction*
Calculate the helicity function for a given object.

Parameters

- **data** – The voxel data as a 3D array that describes the object.
- **voxel_size** – The width of the voxels in the data.
- **delta_alpha** – The bin size for the inclination angles in degrees.
- **delta_rho** – The bin size in the radial direction in the same units as the voxel_size, by default `delta_rho = voxel_size`.
- **kernel_size** – The size of the Sobel kernel used for calculating the gradient, should be an odd integer.

Returns A `HelicityFunction` containing the results.

helicity_map(*data: numpy.ndarray*, *sigma: float*, *threshold: Optional[float] = None*, *kernel_size: int = 5*) → numpy.ndarray

Create a helicity map that indicates the helicity around each voxel

Parameters

- **data** – The voxel data (3D array) that describes the object.
- **sigma** – The effective size of a region over which the local helicity will be averaged, this is done using Gaussian smoothing in 3D.
- **threshold** – If specified, all voxels in the result where `data < threshold` will be set to zero.
- **kernel_size** – The size of the Sobel kernel used for calculating the gradient, should be an odd integer.

Returns A 3D array with the same shape as `data` that contains the 3D helicity map.

plot_helicity_function(*hfunc:* heliq.helicity.HelicityFunction, *vmax: Optional[float] = None*, *axis: Optional[matplotlib.axes._axes.Axes] = None*, *cmap: str = 'coolwarm'*) → matplotlib.image.AxesImage

Plot the helicity function.

Parameters

- **hfunc** – The helicity function of the object.
- **vmax** – The limits for the intensity will be [-vmax, vmax], by default this will be calculated as `max(abs(helicity_function))`.
- **axis** – The axis on which to draw the helicity function. If not provided, the currect active axis will be used.
- **cmap** – The colormap that will be applied, which should ideally be a diverging colormap to properly visualize the difference between left- and right-handed helicity.

Returns A matplotlib axis image containing the helicity function.

total_helicity(*hfunc:* heliq.helicity.HelicityFunction) → float

Calculate the total helicity of an object.

The total helicity is a value between +1 and -1 that indicates the total helicity of the object: a value close to zero indicates achirality, while a value close to +/-1 indicates near-perfect helicity. Positive values indicate right-handed helicity and negative values indicate left-handed helicity.

Parameters **hfunc** – The helicity function of the object.

Returns The total helicity of the object.

CHAPTER

FOUR

CYLINDRICAL COORDINATES

In our paper, we presented the helicity measures starting from a volume in cylindrical coordinates. This was mainly useful for giving an intuitive explanation, but can be skipped in the implementation (see the `heliq.helicity` module). However, it can still be useful to analyze the cylindrical sections because in some cases they can nicely visualize the helical features of a structure.

cylindrical_sections(*data: numpy.ndarray*, *rho: Optional[Union[float, Sequence[float]]] = None*, *n_theta: int = 360*) → numpy.ndarray

Get cylindrical sections from a volume in Cartesian coordinates

The volume should be aligned such that the helical axis goes through the center and is parallel to the z-axis.

Note that the pixels will not be square (different width and height) and that the real width of the pixels will be different at different cylindrical sections. If the voxel size is d, then the height of the pixels is also $h = d$, and the width can be calculated using $w = d \cdot \rho \cdot \arctan(360/n_\theta)$.

Parameters

- **data** – A 3D array containing the volumetric data of the object.
- **rho** – A list of the radii at which to create cylindrical sections, expressed in voxels, the default radii are calculated using `rho = numpy.arange(0, min(data.shape[:2]))`.
- **n_theta** – The number of pixels in the -direction of the cylindrical sections.

Returns A 3D array with axes (z, ,).

CHAPTER

FIVE

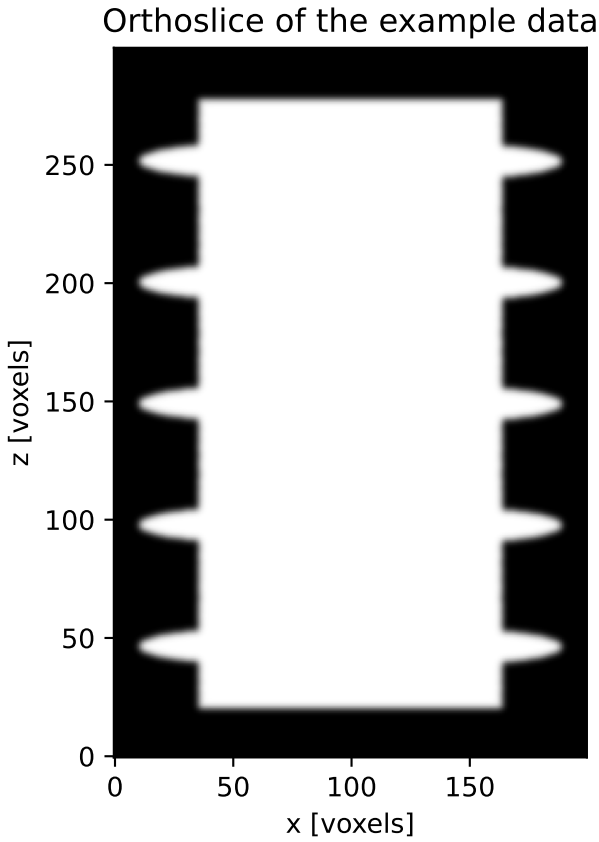
DATA ALIGNMENT AND BASIC ANALYSIS

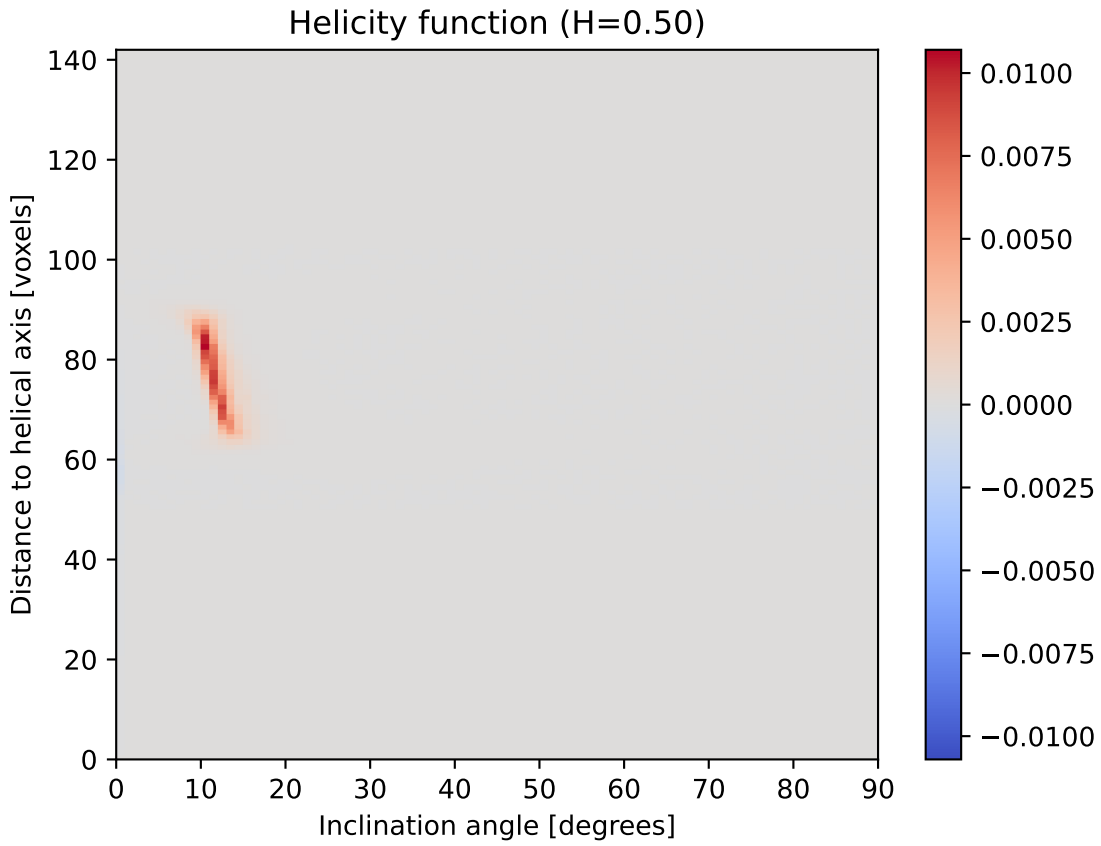
```
"""
In this example I will show what a standard analysis workflow looks like. For
the example I will use a right-handed helical shape that consists of an achiral
core surrounded by two helices from the file ``helix.npy``. This shape is
already pre-aligned, so the alignment steps are not necessary, but for the
purpose of this example I will nevertheless show how to use them.
"""
import numpy as np
import matplotlib.pyplot as plt
import heliq


# First we need to load the data, it is up to you to load your data from any
# file type you want to use. For this example I stored the data with numpy. It
# is important to know that the data should be a 3D numpy array of which the
# first axis corresponds to the real-world y-axis, the second axis to the real
# x-axis, and the third axis to the z-axis. This is the commonly used
# rows-columns-channels convention. You also need to know the width of the
# voxels to be able to compare the helicity of different samples recorded with
# different settings. For the purpose of this example, I'll use a voxel size of
# one.
data = np.load("helix.npy")
voxel_size = 1.0

_, ax = plt.subplots(1, 1)
ax.imshow(data[:, data.shape[1]//2, :].T, cmap='gray', origin='lower')
ax.set_title("Orthoslice of the example data")
ax.set_xlabel("x [voxels]")
ax.set_ylabel("z [voxels]")


# The data needs to be aligned such that the helical axis is in the center and
# parallel to the z-axis. For typical elongated shapes, this can be done by
# aligning the longest axis with the z-axis and moving the center of mass to
# the center of the data. For atypical shapes you might have to do this
# manually.
center = heliq.center_of_mass(data)
orientation = heliq.helical_axis_pca(data, 0)

data = heliq.align_helical_axis(data, orientation, center)
```

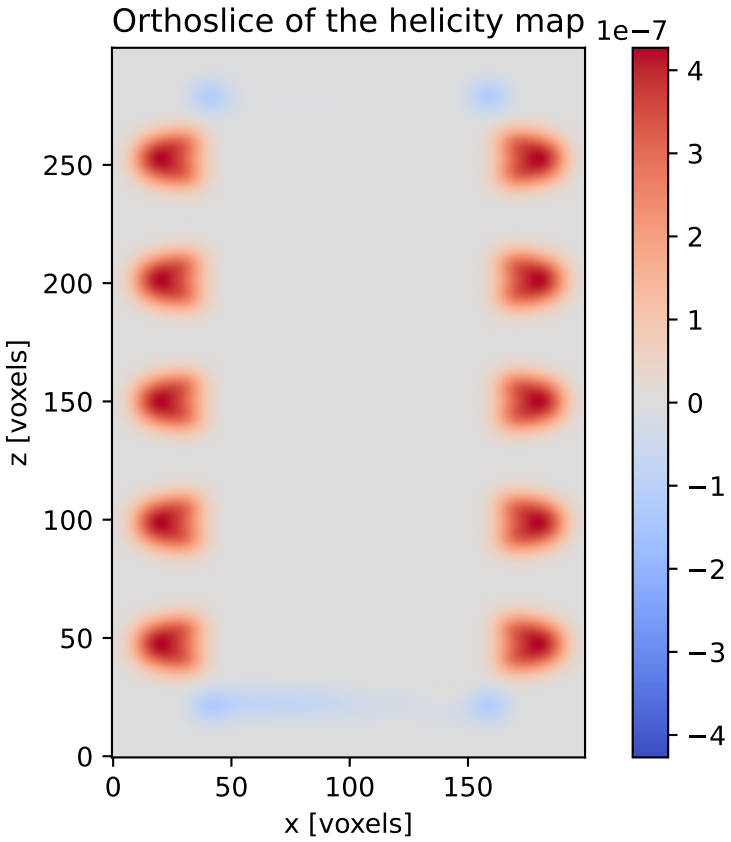
(continues on next page)

(continued from previous page)

```
# Next you can calculate the helicity. The helicity function gives a 2D
# histogram that shows the distribution of helical features in the data as
# function of their inclination angle (alpha) and distance to the helical axis
# (rho). Note that the voxel size is important to be able to compare the
# results of different shapes. Negative values represent left-handed helicity
# and positive values indicate right-handed helicity. The total helicity is a
# value between -1 and 1 that gives an idea of the helicity of the entire
# shape.
hfunc = heliq.helicity_function(data, voxel_size)
htot = heliq.total_helicity(hfunc)


# To plot the helicity function, you can use the following function, which will
# apply a diverging colormap and set the intensity limits appropriatly. You do
# have to call ``plt.show()`` at the end of your script though!
fig, ax = plt.subplots(1, 1)
im = heliq.plot_helicity_function(hfunc, axis=ax)
fig.colorbar(im)
ax.set_title(f"Helicity function (H={htot:0.2f})")
ax.set_xlabel("Inclination angle [degrees]")
ax.set_ylabel("Distance to helical axis [voxels]")


# You can also create a helicity map, which is a 3D numpy array that indicates
# the location of right- and left-handed helical features. The argument
# ``sigma`` needs to be chosen manually according to the expected size of the
# helical features.
hmap = heliq.helicity_map(data, sigma=5)

fig, ax = plt.subplots(1, 1)
im = ax.imshow(hmap[:, hmap.shape[1]//2, :].T, cmap='coolwarm', origin='lower')
ax.set_title("Orthoslice of the helicity map")
ax.set_xlabel("x [voxels]")
ax.set_ylabel("z [voxels]")

vmax = np.max(np.abs(hmap))
im.set_clim(-vmax, vmax)
fig.colorbar(im)


plt.show()
```


Orthoslice of the example data
z [voxels]
x [voxels]
0
50
100
150
200
250
0
50
100
150


Helicity function (H=0.50)
Distance to helical axis [voxels]
Inclination angle [degrees]
0
20
40
60
80
100
120
140
0
10
20
30
40
50
60
70
80
90
0.0100
0.0075
0.0050
0.0025
0.0000
−0.0025
−0.0050
−0.0075
−0.0100

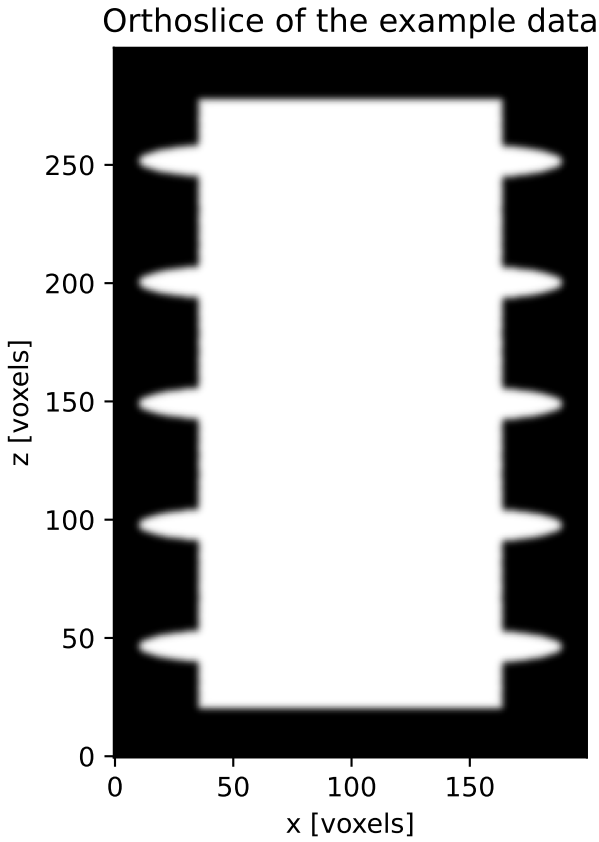

Orthoslice of the helicity map
1e−7
4
3
2
1
0
−1
−2
−3
−4
250
200
150
100
50
0
z [voxels]
0
50
100
150
x [voxels]

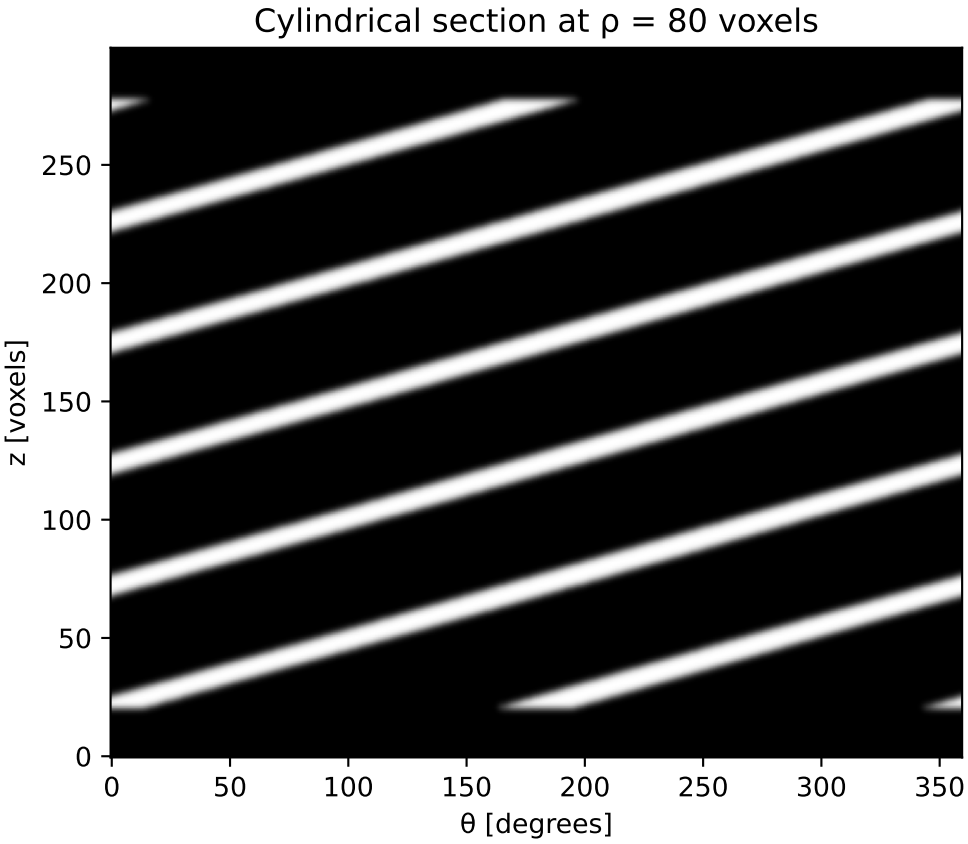
CHAPTER

SIX

CALCULATING AND VISUALIZING CYLINDRICAL SECTIONS

```
"""
This example shows how to calculate and visualize cylindrical sections from
volumetric data in Cartesian coordinates.
"""
import numpy as np
import matplotlib.pyplot as plt
import heliq


# First we need to load the data, see example 1 for more details.
data = np.load("helix.npy")

_, ax = plt.subplots(1, 1)
ax.imshow(data[:, data.shape[1]//2, :].T, cmap='gray', origin='lower')
ax.set_title("Orthoslice of the example data")
ax.set_xlabel("x [voxels]")
ax.set_ylabel("z [voxels]")


# Calculate a cylindrical section using ``heliq.cylindrical_sections``. You can
# calculate multiple sections at once by providing a list to the ``rho``
# argument. If your voxel size is not 1, but e.g. 0.15 nm, and you want to
# calculate a cylindrical section at  = 10 nm, you should enter ``rho=10/0.15``
# or in general ``rho=desired_rho/voxel_size``.
data_cyl = heliq.cylindrical_sections(data, rho=80)

_, ax = plt.subplots(1, 1)
ax.imshow(data_cyl, cmap='gray', origin='lower')
ax.set_title("Cylindrical section at  = 80 voxels")
ax.set_xlabel(" [degrees]")
ax.set_ylabel("z [voxels]")


plt.show()
```


Orthoslice of the example data
z [voxels]
x [voxels]
0
50
100
150
200
250
0
50
100
150


Cylindrical section at ρ = 80 voxels
z [voxels]
θ [degrees]
0
50
100
150
200
250
300
350

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX