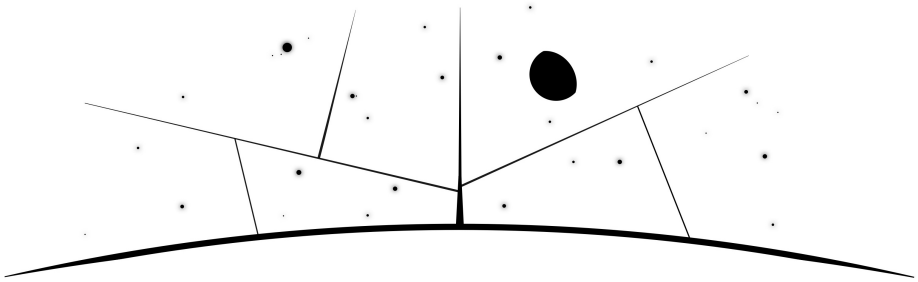
SkySegmentor

Krishna Naidoo

Jun 03, 2025

CONTENTS



SkySegmentor

Version	0.0.6
Repository	https://github.com/knaidoo29/SkySegmentor
Documentation	https://skysegmentor.readthedocs.io/

CHAPTER

ONE

CONTENTS

CHAPTER

TWO

INTRODUCTION

SkySegmentor is a `python 3` package for dividing points or maps (in `HEALPix` format) on the celestial sphere into equal-sized segments. It employs a sequential binary space partitioning scheme – a generalization of the k-d tree algorithm – that supports segmentation of arbitrarily shaped sky regions. By design, all partitions are approximately equal in area, with discrepancies no larger than the `HEALPix` pixel scale.

CHAPTER

THREE

DEPENDENCIES

- numpy – versions: *>=1.22,<1.27*
- healpy – versions: *>=1.15.0*

CHAPTER

FOUR

INSTALLATION

First cloning the repository

```
git clone https://github.com/knaidoo29/SkySegmentor.git
cd SkySegmentor
```

and install by running

```
pip install . [--user]
```

You should now be able to import the module:

```
import skysegmentor
```

CHAPTER

FIVE

TUTORIALS

5.1 Tutorials

5.1.1 Basic Usage

Segmenting a Healpix Maps

```
import healpy
import skysegmentor

# Healpix mask, where zeros are regions outside of the mask and ones inside the
# mask. You can also input a weighted map, where instead of 1s you give weights.
mask = # define mask values

Npartitions = 100 # Number of partitions
partitionmap = skysegmentor.segmentmapN(mask, Npartitions)
```

Segmenting Points on the Sphere

```
import skysegmentor

# Define points on the sphere to be segmented.
phi = # longitude defined in radians from [0, 2*pi]
the = # latitude defined in radians from [0, pi], where 0 = North Pole.

Npartitions = 100 # Number of partitions
partitionIDs = skysegmentor.segmentpointsN(phi, the, Npartitions)
```

if using RA and Dec in degrees you can convert to logitudinal and latitude coordinates *phi* and *the* using

```
phi = np.deg2rad(ra)
the = np.deg2rad(90. - dec)
```

if not all points are equal, you can specify a weight

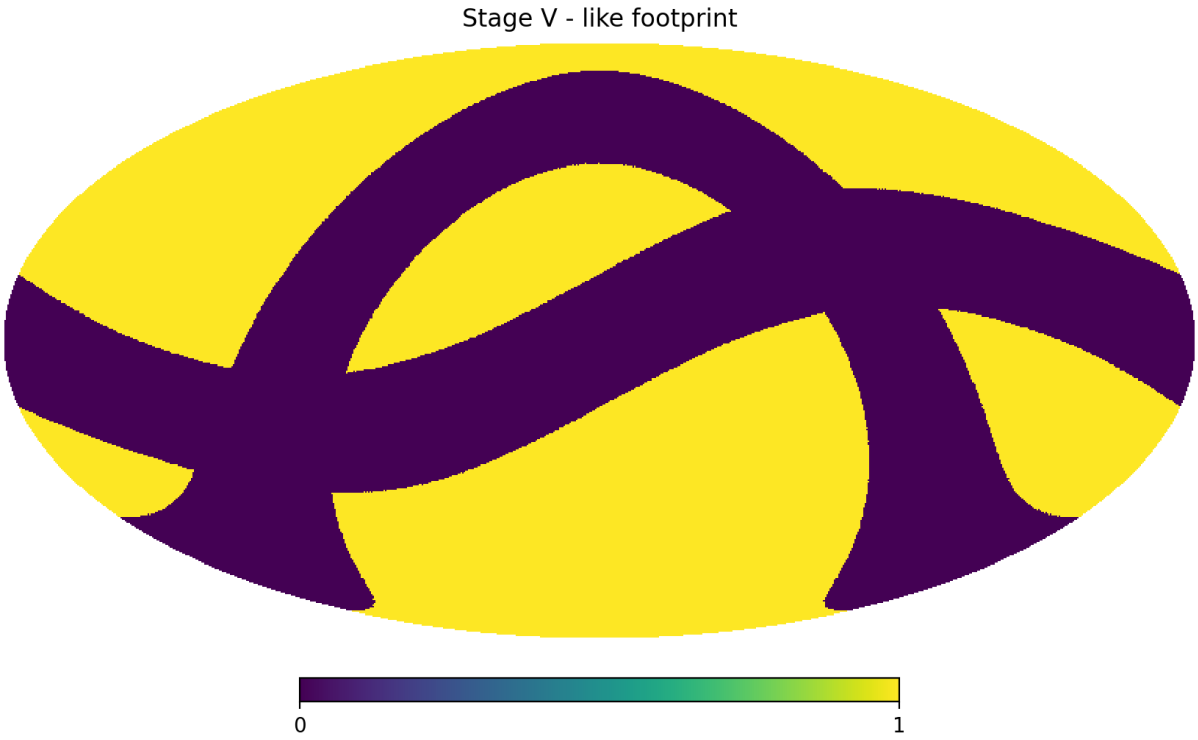
```
weights = # define point weights
partitionIDs = skysegmentor.segmentpointsN(phi, the, Npartitions, weights=weights)
```

5.1.2 Advanced Usage

Region finding

There will be cases when the region on the sky that we want to partition is complex and is non-contiguous. These regions may be trivial, but they also may be complex – for the latter it is useful to have software that can automatically identify these regions. To demonstrate how this works, let's start by first defining a stage V survey like footprint.

```
import numpy as np
import matplotlib.pylab as plt
import healpy as hp
import skysegmentor

nside = 256

# ------------------------------------------------------------------------
# the exact details used to define this are not important but placed
# here so you can reproduce this example.

pixID = np.arange(hp.nside2npix(nside))
map0 = np.ones(hp.nside2npix(nside))

rot = hp.Rotator(coord=['E', 'G'])

map0 = rot.rotate_map_pixel(map0)

the, phi = hp.pix2ang(nside, pixID)

cond = np.where((the >= np.deg2rad(90.-15.)) & (the <= np.deg2rad(90.+15.)))[0]
map0[cond] = 0.

rot = hp.Rotator(coord=['G', 'C'])

map0 = rot.rotate_map_pixel(map0)

cond = np.where((the >= np.deg2rad(90.-15.)) & (the <= np.deg2rad(90.+15.)))[0]
map0[cond] = 0.

rot = hp.Rotator(coord=['C', 'E'])

map0 = rot.rotate_map_pixel(map0)

cond = np.where(map0 > 0.5)[0]
map0[cond] = 1.
cond = np.where(map0 <= 0.5)[0]
map0[cond] = 0.
map0 = map0.astype('int')

# ------------------------------------------------------------------------

hp.mollview(map0, title='Stage V - like footprint')
plt.show()
```

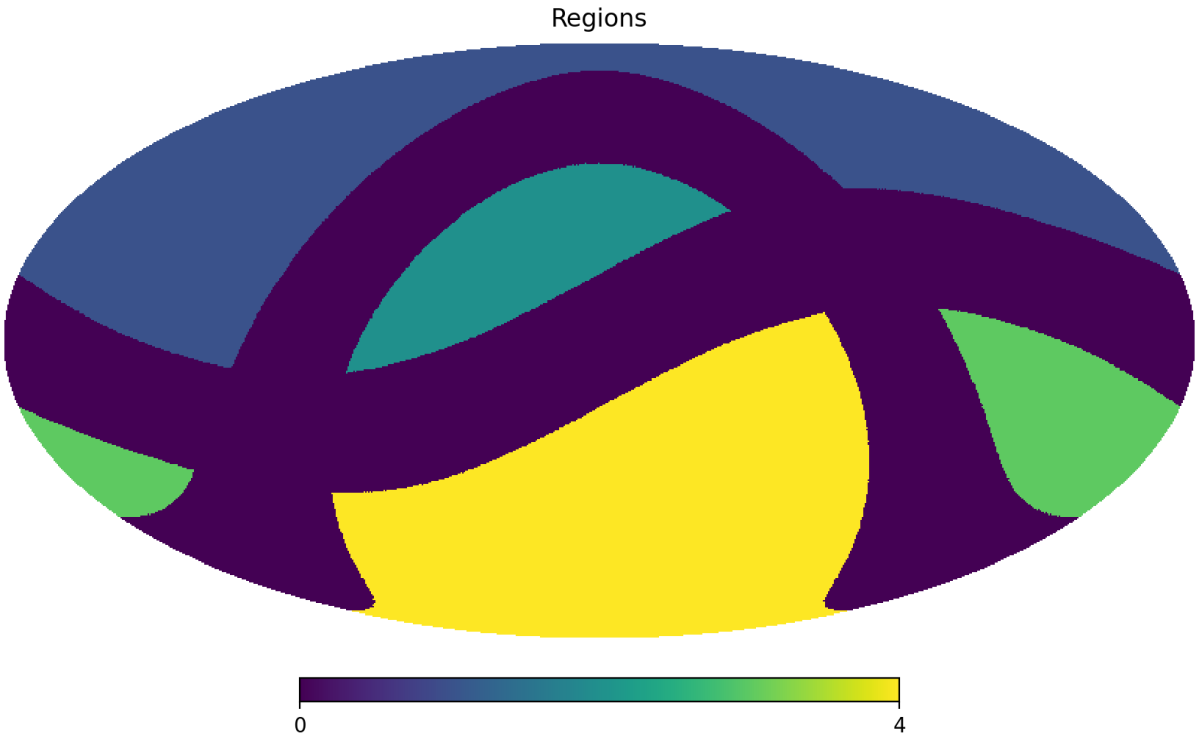


We can now use *SkySegmentor*'s inbuilt *unionfinder* function to find contiguous regions in the footprint. This uses an adapted version of the grid-based Hoshen-Kopelman union finding algorithm altered to work on the HEALPix grid.

Note

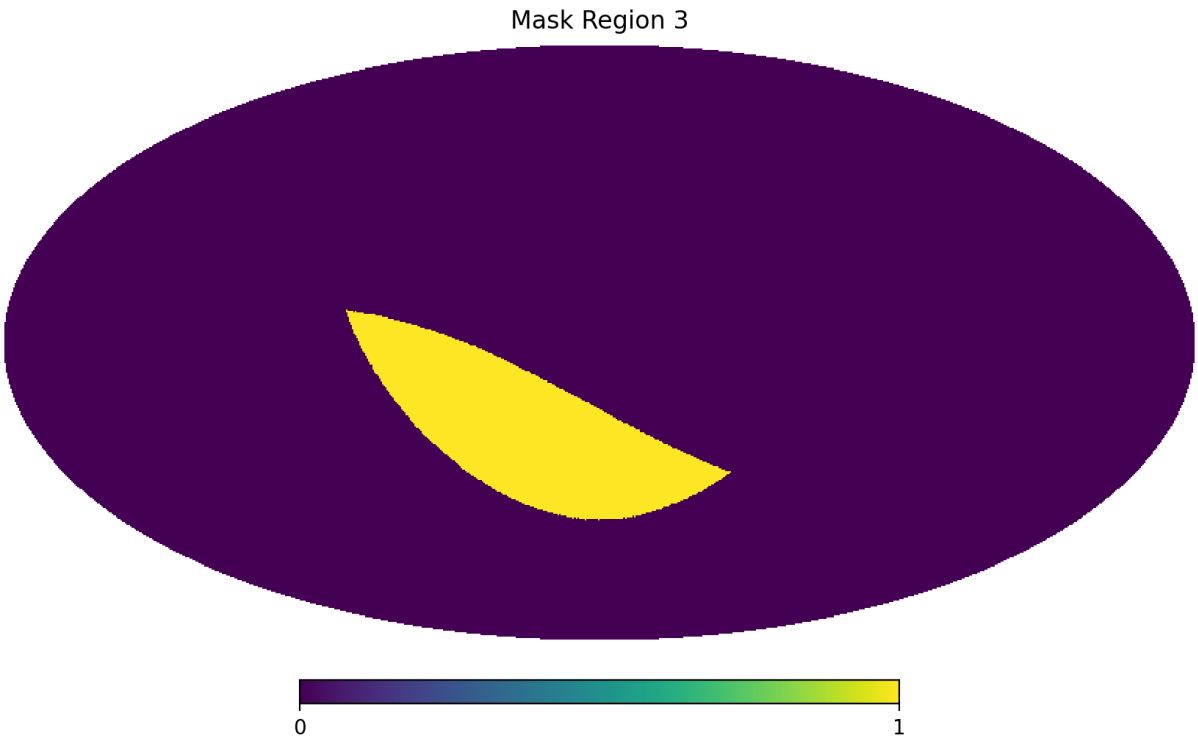
groupmap = 0 is used to represent masked out region.

```
groupmap = skysegmentor.unionfinder(map0)
```



Let's focus on region 1, by constructing a mask for region 1 only:

```
mask = np.zeros(hp.nside2npix(nside))
cond = np.where(groupmap == 3)[0]
mask[cond] = 1.
```



We will now partition this mask in the following section.

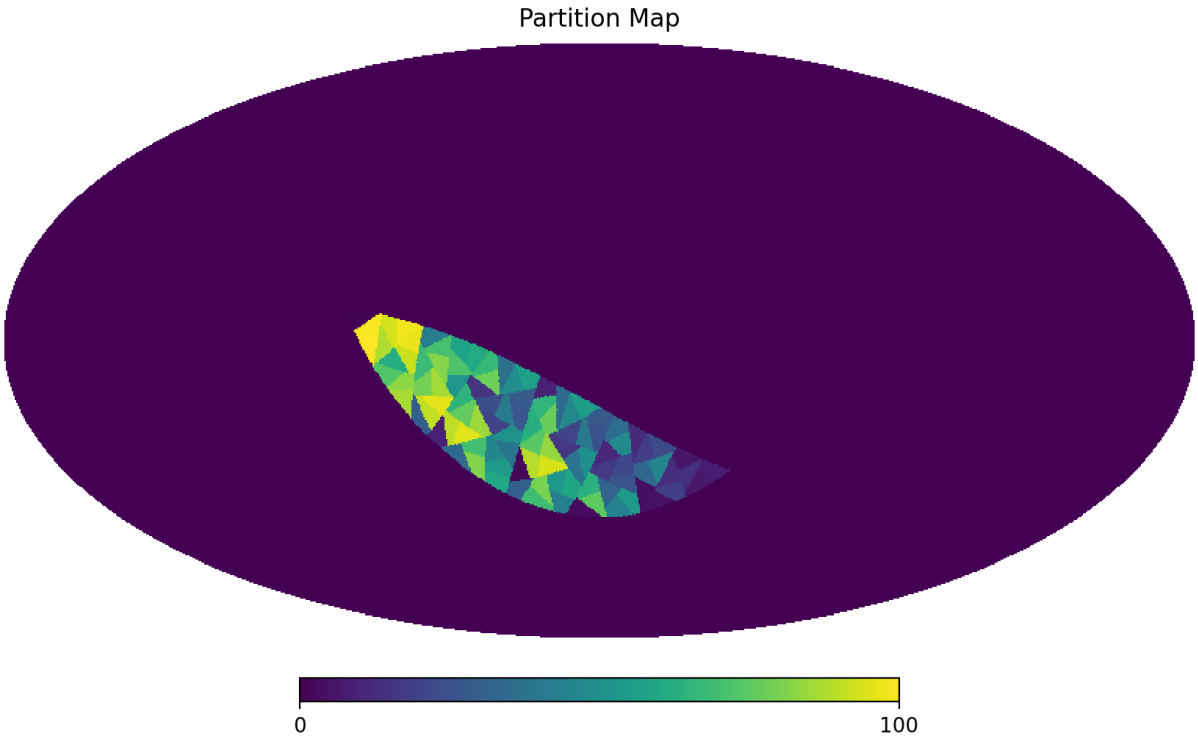
Partitioning a HEALPix map

We can partitioning a HEALPix mask into *Npartitions* by running:

```
Npartitions = 100
partitionmap = skysegmentor.segmentmapN(mask, Npartitions)

# and plotting the partition map
hp.mollview(partitionmap, title='Partition Map', rot=180)
```

Note

partitionmap = *0* is used to represent masked out region.



Note

This is assuming *Npartitions* is smaller than the number of pixels in the mask and thus not limited by the pixel scale of the map.

Partitioning a HEALPix map by weight

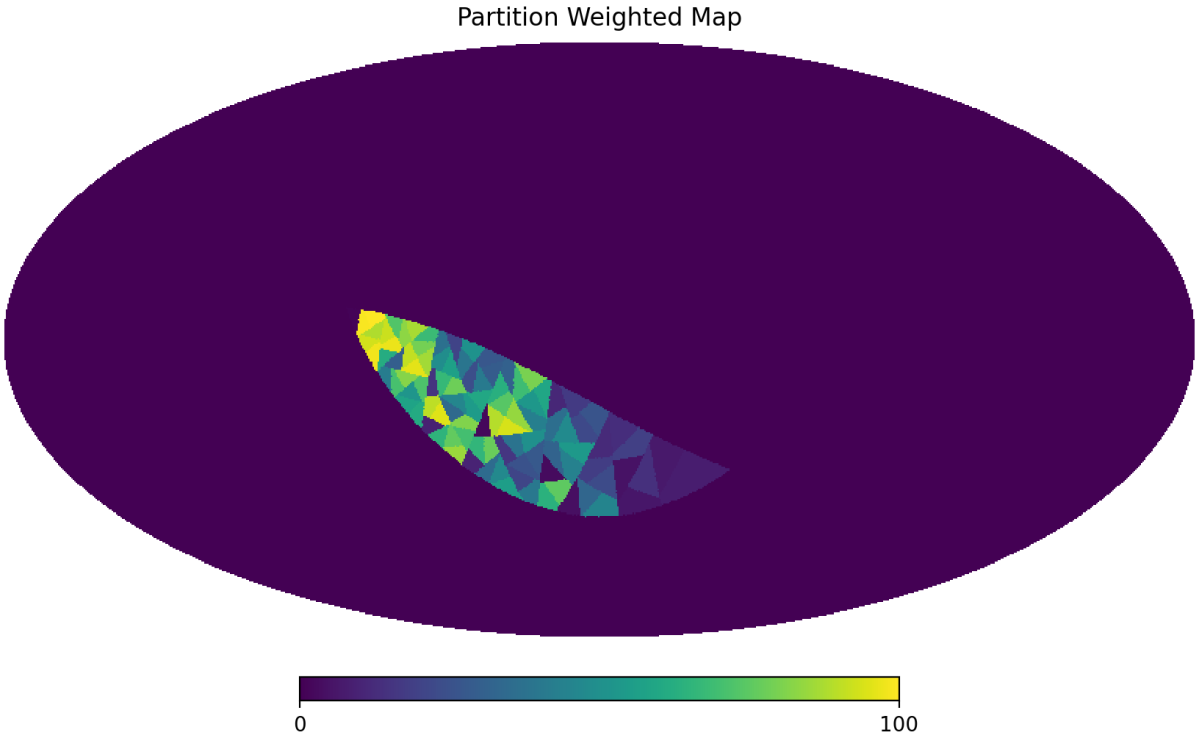
Let's now partition the same map but now with a weight map:

```
Npartitions = 100

# in this example the weight map will be proportional to the latitude
weightmap = np.zeros(hp.nside2npix(nside))
pixID = np.where(mask == 1.)[0]
the, phi = hp.pix2ang(nside, pixID)
weightmap[pixID] = (phi - phi.min())/(phi.max() - phi.min())

# you must ensure the weightmap has the correct footprint, which can be done by
# multiplying the weightmap by the mask

weightmask = weightmap * mask

partitionmap = skysegmentor.segmentmapN(weightmask, Npartitions)

# and plotting the weighted partition map
```

(continues on next page)

(continued from previous page)

```
hp.mollview(partitionmap, title='Partition Weighted Map', rot=180)
```



Partitioning a set of points

Points on the sphere in skysegmentor are alway defined in phi (longitude) and theta (latitude), both defined in radians where phi lies in the range [0, 2 pi] and theta [0, pi] where *theta=0* ` is the north pole. To convert astronomical RA and Dec to phi and theta simply do:

```
phi = np.deg2rad(ra) # convert RA to phi
the = np.deg2rad(90.-dec) # convert Dec to theta
```

and to convert back

```
ra = np.rad2deg(phi) # convert phi to RA
dec = 90. - np.rad2deg(the) # convert theta to Dec
```

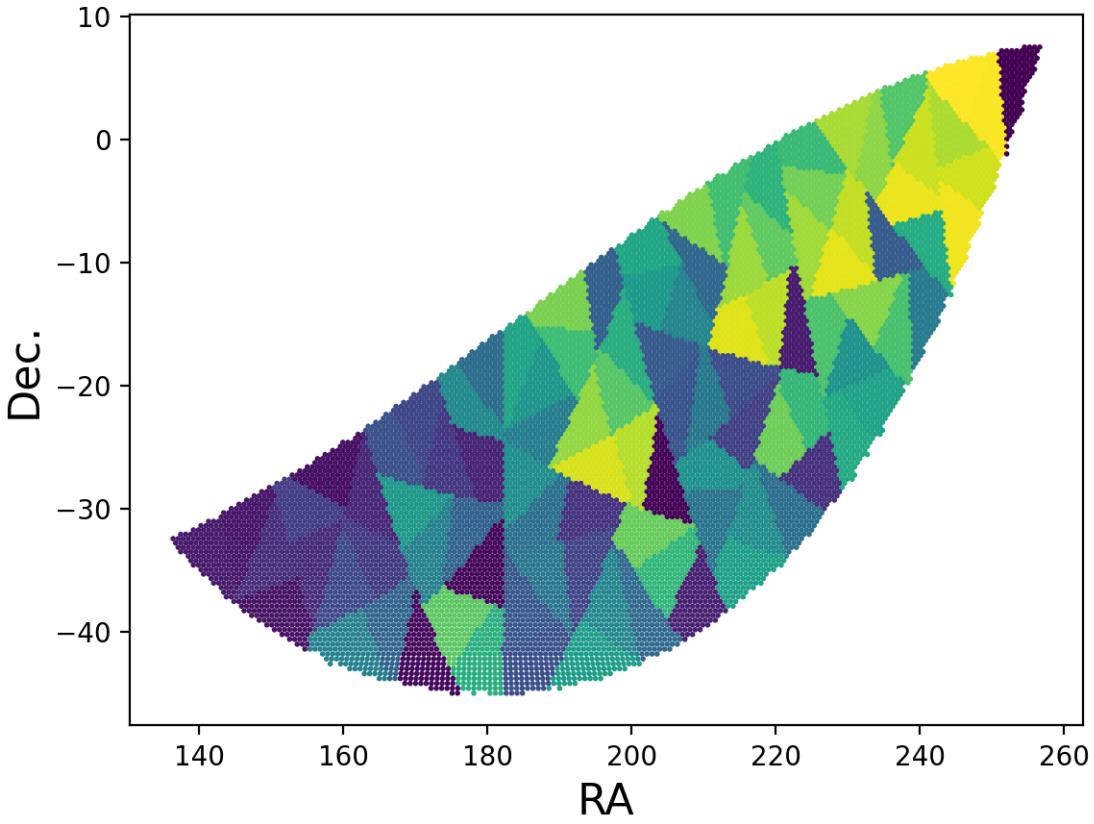
Let's now partition a set of points

```
Npartitions = 100

# let's construct a set of points
pixID = np.where(mask == 1.)[0]
the, phi = hp.pix2ang(nside, pixID)

partitionID = skysegmentor.segmentpointsN(phi, the, Npartitions)
```

(continues on next page)

(continued from previous page)

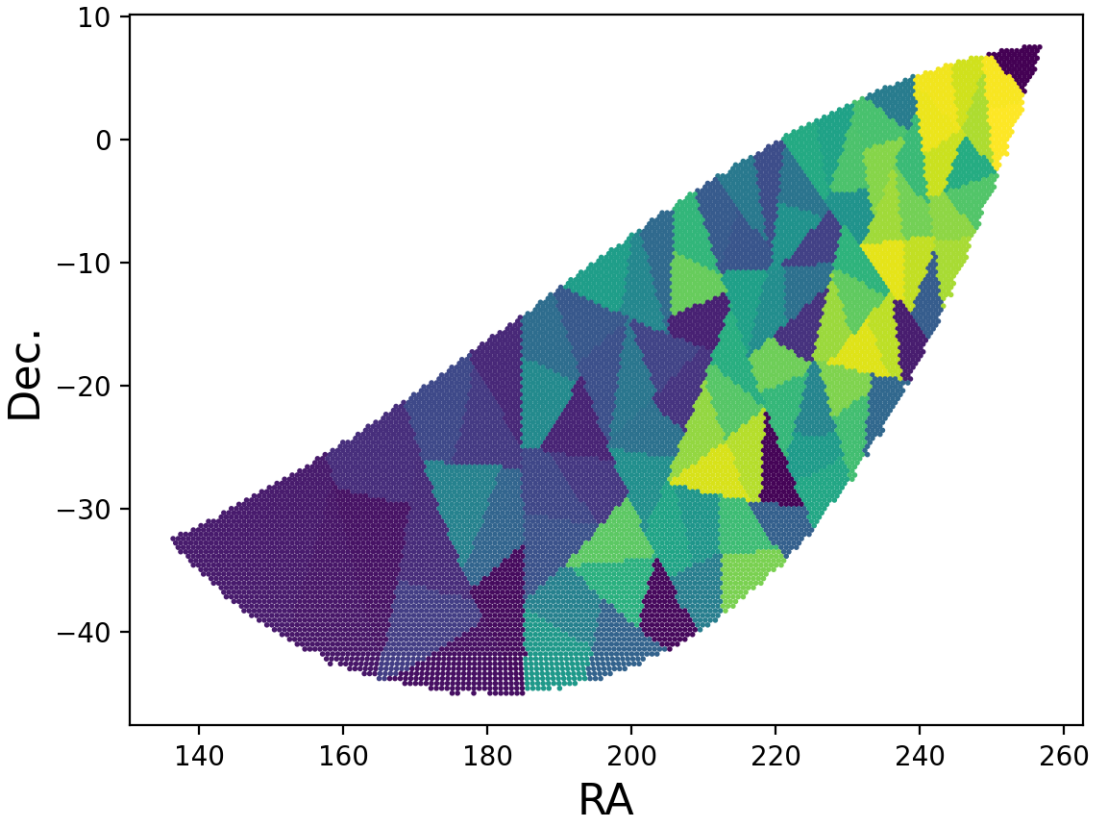
```
Npartitions = 100

# let's construct a set of points, this part can be replaced by your points ---
nside2 = 128

pixID = np.arange(hp.nside2npix(nside2))
the, phi = hp.pix2ang(nside2, pixID)
pixID = hp.ang2pix(nside, the, phi)
cond = np.where(mask[pixID] == 1.)[0]
the, phi = the[cond], phi[cond]

# ------------------------------------------------------------------------------

# If your points are in ra and dec simply convert to phi, theta as shown above
# and repeated here:

# phi = np.deg2rad(ra) # convert RA to phi
# the = np.deg2rad(90.-dec) # convert Dec to theta

# ------------------------------------------------------------------------------

# Now let's partition the points themselves!

partitionID = skysegmentor.segmentpointsN(phi, the, Npartitions)

# converting back to ra and dec for plotting.

ra = np.rad2deg(phi) # convert phi to RA
dec = 90. - np.rad2deg(the) # convert theta to Dec

# plot points and partitionIDs

plt.scatter(ra, dec, c=partitionID, s=1.)
plt.xlabel(r'RA', fontsize=16)
plt.ylabel(r'Dec.', fontsize=16)
plt.show()
```



Partitioning a set of weighted points

If the points have weights, then this can simply be added as

```
Npartitions = 100

# let's construct some weights
weights = (phi - phi.min()) / (phi.max() - phi.min())

partitionID = skysegmentor.segmentpointsN(phi, the, Npartitions, weights=weights)

ra = np.rad2deg(phi) # convert phi to RA
dec = 90. - np.rad2deg(the) # convert theta to Dec

# plot points and partitionIDs

plt.scatter(ra, dec, c=partitionID, s=1)
plt.xlabel(r'RA', fontsize=16)
plt.ylabel(r'Dec.', fontsize=16)
plt.show()
```


10
0
−10
−20
−30
−40
Dec.
140
160
180
200
220
240
260
RA

CHAPTER

SIX

API

6.1 API

6.1.1 coords

Coordinate transformations.

skysegmentor.**cart2sphere**(*x*, *y*, *z*, *center=[0.0, 0.0, 0.0]*)

Return polar coordinates for a given set of cartesian coordinates.

Parameters

- **x** (*array*) – x coordinate.
- **y** (*array*) – y coordinate.
- **z** (*array*) – z coordinate.
- **center** (*list*) – Center point of polar coordinate grid.

Returns

- **r** (*array*) – Radial coordinates.
- **phi** (*array*) – Phi coordinates [0, 2pi].
- **theta** (*array*) – Theta coordinates [0, pi].

skysegmentor.**sphere2cart**(*r*, *phi*, *theta*, *center=[0.0, 0.0, 0.0]*)

Converts spherical polar coordinates into cartesian coordinates.

Parameters

- **r** (*array*) – Radial distance.
- **phi** (*array*) – Longitudinal coordinates (radians = [0, 2pi]).
- **theta** (*array*) – Latitude coordinates (radians = [0, pi]).
- **center** (*list*) – Center point of spherical polar coordinate grid.

Returns

x, y, z – Euclidean coordinates.

Return type

array

skysegmentor.**distusphere**(*phi1*, *theta1*, *phi2*, *theta2*)

Compute angular (great-arc) distance between two points on a unit sphere.

Parameters

- **phi1** (*float or array*) – Location of first points on the unit sphere.
- **theta1** (*float or array*) – Location of first points on the unit sphere.
- **phi2** (*float or array*) – Location of second points on the unit sphere.
- **theta2** (*float or array*) – Location of second points on the unit sphere.

Returns

dist – Angular great-arc distance.

Return type

float or array

6.1.2 groupfinder

Group finding on a HEALPix map.

skysegmentor.**unionfinder**(*binmap*)

Group or label assignment on a healpix grid using the HoshenKopelman algorithm.

Parameters

binmap (*array*) – Binary healpix map.

Returns

groupID – Labelled healpix map.

Return type

array

6.1.3 maths

Simple vector and matrix maths.

skysegmentor.**vector_norm**(*a*)

Returns the magnitude a vector.

Parameters

a (*array*) – Vector a.

skysegmentor.**vector_dot**(*a*, *b*)

Returns the vector dot product.

Parameters

- **a** (*array*) – Vector a.
- **b** (*array*) – Vector b.

skysegmentor.**vector_cross**(*a*, *b*)

Returns the vector cross product.

Parameters

- **a** (*array*) – Vector a.
- **b** (*array*) – Vector b.

Returns
s – Cross product vector.

Return type
array

skysegmentor.**matrix_dot_3by3**(*mat1*, *mat2*)

Dot product of 2 3x3 matrices.

Parameters

- **mat1** (*array*) – Flattened 3by3 matrices.
- **mat2** (*array*) – Flattened 3by3 matrices.

Returns
mat3 – Matrix dot product output.

Return type
array

6.1.4 partition

Map and point partitioning related functions.

skysegmentor.**get_partition_IDs**(*partition*)

Returns the total weights of each partition.

Parameters
partition (*array*) – Partition IDs on a map. Unfilled partitions are assigned partition 0.

Returns
partition_IDs – Unique partition IDs, including zero.

Return type
array

skysegmentor.**total_partition_weights**(*partition*, *weights*)

Returns the total weights of each partition.

Parameters

- **partition** (*array*) – Partition IDs on a map. Unfilled partitions are assigned partition 0.
- **weights** (*array*) – A weight assigned to each element of the partition array.

Returns

- **partition_IDs** (*array*) – Unique partition IDs, including zero.
- **partition_weights** (*array*) – The total weight for each partition.

skysegmentor.**remove_val4array**(*array*, *val*)

Removes a given value from an array.

Parameters

- **array** (*array*) – Data vector.
- **val** (*float*) – Value to be removed from an array.

skysegmentor.**fill_map**(*pixID*, *nside*, *val=1.0*)

Fill a Healpix map with a given value val at given pixel locations.

Parameters

- **pixID** (*int array*) – Pixel index.
- **nside** (*int*) – HEalpix map nside.
- **val** (*float, optional*) – Value to fill certain pixels with.

skysegmentor.**find_map_barycenter**(*bnmap*, *wmap=None*)

Determines the barycenter of center of mass direction of the input binary map.

Parameters

- **bnmap** (*array*) – binary map.
- **wmap** (*array, optional*) – The weights.

Returns

phic, thec – The center

Return type

float

skysegmentor.**find_points_barycenter**(*phi*, *the*, *weights=None*)

Determines the barycenter of center of mass direction of the input point dataset.

Parameters

- **phi** (*array*) – Angular coordinates.
- **the** (*array*) – Angular coordinates.
- **weights** (*array, optional*) – Weights for points.

Returns

phic, thec – The center

Return type

float

skysegmentor.**get_map_border**(*bnmap*, *wmap=None*, *res=[200, 100]*)

Determines the outer border of binary map region.

Parameters

- **bnmap** (*array*) – binary map.
- **wmap** (*array, optional*) – The weights.
- **res** (*list, optional*) – Resolution of spherical cap grid where [phiresolution, thetaresolution] to find region border.

Returns

phi_border, the_border – Approximate border region.

Return type

array

skysegmentor.**get_points_border**(*phi*, *the*, *weights=None*, *res=100*)

Determines the outer border of binary map region.

Parameters

- **phi** (*array*) – Angular coordinates.
- **the** (*array*) – Angular coordinates.
- **weights** (*array, optional*) – Weights for points.
- **res** (*int, optional*) – Resolution of spherical cap grid for phiresolution to find region border.

Returns
phi_border, the_border – Approximate border region.

Return type
array

skysegmentor.**get_map_most_dist_points**(*bnmap*, *wmap=None*, *res=[100, 50]*)

Returns the most distant points on a binary map.

Parameters

- **bnmap** (*int array*) – Binary healpix map.
- **wmap** (*array, optional*) – The weights.
- **res** (*list, optional*) – Resolution of spherical cap grid where [phiresolution, thetaresolution] to find region border.

Returns
p1, t1, p2, t2 – Angular coordinates (phi, theta) for the most distant points (1 and 2) on the binary map.

Return type
float

skysegmentor.**get_points_most_dist_points**(*phi*, *the*, *weights=None*, *res=100*)

Returns the most distant points from a set of points.

Parameters

- **phi** (*array*) – Angular coordinates.
- **the** (*array*) – Angular coordinates.
- **weights** (*array, optional*) – Weights for points.
- **res** (*int, optional*) – Resolution of spherical cap grid for phiresolution to find region border.

Returns
p1, t1, p2, t2 – Angular coordinates (phi, theta) for the most distant points (1 and 2) on the binary map.

Return type
float

skysegmentor.**weight_dif**(*phi_split*, *phi*, *weights*, *balance=1*)

Compute the difference between the weights on either side of phi_split.

Parameters

- **phi_split** (*float*) – Longitude split.
- **phi** (*array*) – Longitude coordinates.
- **weights** (*array*) – Weights corresponding to each longitude coordinates.

- **balance** (*float, optional*) – A multiplication factor assigned to weights below phi_split.

skysegmentor.**find_dphi**(*phi*, *weights*, *balance=1*)

Determines the splitting longitude required for partitioning, either with 1-to-1 weights on either side or unbalanced weighting if balance != 1.

Parameters

- **hi** (*array*) – Longitude coordinates.
- **weights** (*array*) – Weights corresponding to each longitude coordinates.

Returns
dphi – Splitting longitude.

Return type
float

skysegmentor.**segmentmap2**(*weightmap*, *balance=1*, *partitionmap=None*, *partition=None*, *res=[100, 50]*)

Segment a map with weights into 2 equal (unequal in balance != 1).

Parameters

- **weightmap** (*array*) – Healpix weight map.
- **balance** (*float, optional*) – Balance of the weights for the partitioning.
- **partitionmap** (*int array, optional*) – Partitioned map IDs.
- **partition** (*int, optional*) – A singular partition to be partitioned in two pieces.
- **res** (*list, optional*) – Resolution of spherical cap grid where [phiresolution, thetaresolution] to find region border.

Returns
partitionmap – Partitioned map IDs.

Return type
int array

skysegmentor.**segmentpoints2**(*phi*, *the*, *weights=None*, *balance=1*, *partitionID=None*, *partition=None*, *res=100*)

Segments a set of points with weights into 2 equal (unequal in balance != 1).

Parameters

- **phi** (*array*) – Angular positions.
- **the** (*array*) – Angular positions.
- **weights** (*array, optional*) – Angular position weights.
- **balance** (*float, optional*) – Balance of the weights for the partitioning.
- **partitionID** (*int array, optional*) – Partitioned map IDs.
- **partition** (*int, optional*) – A singular partition to be partitioned in two pieces.
- **res** (*float, optional*) – Resolution of spherical cap phiresolution to find region border.

Returns
partitionID – Partitioned map IDs.

Return type
int array

skysegmentor.**segmentmapN**(*weightmap*, *Npartitions*, *res=[100, 50]*)

Segment a map with weights into equal Npartition sides.

Parameters

- **weightmap** (*array*) – Healpix weight map.
- **Npartitions** (*int*) – Number of partitioned regions
- **res** (*list, optional*) – Resolution of spherical cap grid where [phiresolution, thetaresolution] to find region border.

Returns
partitionmap – Partitioned map IDs.

Return type
int array

skysegmentor.**segmentpointsN**(*phi*, *the*, *Npartitions*, *weights=None*, *res=100*)

Segments a set of points with weights into equal Npartition sides.

Parameters

- **weightmap** (*array*) – Healpix weight map.
- **Npartitions** (*int*) – Number of partitioned regions
- **res** (*list, optional*) – Resolution of spherical cap grid where [phiresolution, thetaresolution] to find region border.

Returns
partitionmap – Partitioned map IDs.

Return type
int array

6.1.5 rotate

Functions for rotations on the unit sphere.

skysegmentor.**rotate3d_Euler**(*x*, *y*, *z*, *angles*, *axes='zyz'*, *center=[0.0, 0.0, 0.0]*)

Rotates points in 3D cartesian coordinates by Euler angle around specified axes.

Parameters

- **x** (*float or array*) – Cartesian coordinates.
- **y** (*float or array*) – Cartesian coordinates.
- **z** (*float or array*) – Cartesian coordinates.
- **angles** (*array*) – Euler angles.
- **axes** (*str, optional*) – Euler angle axes, default z-y-z rotations.
- **center** (*list[float], optional*) – Center of rotation, default=[0., 0., 0.].
- **k** (*array, optional*) – If k is specified, points are rotated around a unit vector k.

Returns
xrot, yrot, zrot – Rotated x, y and z coordinates.

Return type
float or array

skysegmentor.**rotate_usphere**(*phi*, *the*, *angles*)

Rotates spherical coordinates by Euler angles performed along the z-axis, then y-axis and then z-axis.

Parameters

- **phi** (*float or array*) – Spherical angular coordinates.
- **the** (*float or array*) – Spherical angular coordinates.
- **angles** (*list*) – Euler angles defining rotations about the z-axis, y-axis then z-axis.

skysegmentor.**midpoint_usphere**(*phi1*, *phi2*, *the1*, *the2*)

Finds the spherical angular coordinates of the midpoint between two points on a unit sphere.

Parameters

- **phi1** (*float*) – Longitude coordinates for both points.
- **phi2** (*float*) – Longitude coordinates for both points.
- **the1** (*float*) – Latitude coordinates for both points.
- **the2** (*float*) – Latitude coordinates for both points.

Returns

midphi, midthe – Midpoint along the longitude phi and latitude theta.

Return type

float

skysegmentor.**rotate2plane**(*c1*, *c2*)

Finds the rotation angles to place the two coordinates c1 and c2 along a latitude = pi/2 (i.e. equator of sphere) and with a midpoint of longitude = pi.

Parameters

- **c1** (*float*) – Coordinates of two points where c1 = [phi1, theta1] and c2 = [phi2, theta2].
- **c2** (*float*) – Coordinates of two points where c1 = [phi1, theta1] and c2 = [phi2, theta2].

Returns

a1, a2, a3 – Euler angles of rotation.

Return type

lists

skysegmentor.**forward_rotate**(*phi*, *the*, *a1*, *a2*, *a3*)

Applies a forward rotation of spherical angular coordinates phi and theta using the forward Euler angles of rotation a1, a2 and a3.

Parameters

- **phi** (*float or array*) – Spherical angular coordinates.
- **the** (*float or array*) – Spherical angular coordinates.
- **a1** (*lists*) – Euler angles of rotation.
- **a2** (*lists*) – Euler angles of rotation.
- **a3** (*lists*) – Euler angles of rotation.

skysegmentor.**backward_rotate**(*phi*, *the*, *a1*, *a2*, *a3*)

Applies a backward rotation of spherical angular coordinates phi and theta using the forward Euler angles of rotation a1, a2 and a3.

Parameters

- **phi** (*float or array*) – Spherical angular coordinates.
- **the** (*float or array*) – Spherical angular coordinates.
- **a1** (*lists*) – Euler angles of rotation.
- **a2** (*lists*) – Euler angles of rotation.
- **a3** (*lists*) – Euler angles of rotation.

6.1.6 utils

Utility functions.

skysegmentor.**isscalar**(*x*)

More general isscalar function to prevent 0 dimensional numpy arrays from being misidentified as arrays even though they are actually scalar variables.

CHAPTER

SEVEN

CITING

> **Warning**
>
> These are placeholders to be replaced upon publication on ArXiv.

If you use `SkySegmentor` in a publication please cite:

- NASA ADS:
- ArXiv:

BibTex:

```
@ARTICLE{Naidoo2025,
    author = {{Euclid Collaboration} and {Naidoo}, K. and {Ruiz-Zapatero}, J.
    and {Tessore}, N. and {Joachimi}, B. and {Loureiro}, A. and others ...},
    title = "{Euclid preparation: TBD. Accurate and precise data-driven
    angular power spectrum covariances}"
}
```

and include a link to the SkySegmentor documentation page: https://skysegmentor.readthedocs.io/

CHAPTER

EIGHT

SUPPORT

If you have any issues with the code or want to suggest ways to improve it please open a new issue (here) or (if you don't have a github account) email krishna.naidoo.11@ucl.ac.uk.

CHAPTER

NINE

VERSION HISTORY

- **Version 0.0.0**:

INDEX