MAGPIE

Release 0.2

Krishna Naidoo

Nov 27, 2021

CONTENTS

1 Contents **3**

2 Introduction **5**

3 Tutorials and API **7**

3.1 Coords 7
3.2 Grids 10
3.3 Pixel 13
3.4 Randoms 16
3.5 Remap 22
3.6 API 23

4 Dependencies **25**

5 Installation **27**

6 Support **29**

7 Version History **31**

Index **33**

Author	Krishna Naidoo
Version	0.2.2
Repository	https://github.com/knaidoo29/magpie
Documentation	https://magpie-doc.readthedocs.io/

Warning: MAGPIE is currently in development. Functions and classes may change. Use with caution.

CHAPTER

ONE

CONTENTS

- *Introduction*
- *Tutorials and API*
- *Dependencies*
- *Installation*
- *Support*
- *Version History*

CHAPTER

TWO

INTRODUCTION

MAGPIE is a python module for remapping bins (in 1D), pixels (in 2D) and cells (in 3D) into different coordinate systems. The package will enable data to be remapped from cartesian to polar coordinates, cartesian to spherical polar coordinates and will enable rotations of these systems. When carrying out coordinate transformations we typically take the center of the pixel or cell and apply the transform without considering the surface area of the pixel (or volume of the cell). In MAGPIE this is taken into account by applying two remapping schemes. The first weights pixels to a new coordinate grid using monte-carlo integration. The second uses a higher-resolution mesh (denser than the new coordinate grid by some integer factor along each dimension) which is rebinned to the target coordinate grid. In both cases we sample the surface area or volume of the new coordinate pixels to accurately remap the data. The monte-carlo method is more accurate but scales poorly to 3D. For 2D this scheme will work very well even for moderately large datasets. The higher-resolution mesh method, while less accurate, is very fast and should be used for large data sets and all 3D transformations. In 1D these are computed exactly without requiring the approximate schemes above.

Note: MPI functionality is currently unavailable but the plan will be to implement this using mpi4py and an additional library MPIutils which will handle all of the MPI enabled functions.

CHAPTER

THREE

TUTORIALS AND API

3.1 Coords

3.1.1 Conventions

All angular coordinates are defined in radians.

Polar Coordinates

Polar coordinates are related to 2D cartesian coordinates by the relations

$$x = r\cos(\phi) + x_0,$$
$$y = r\sin(\phi) + y_0,$$

where x_0 and y_0 define the center of the polar coordinate grid. The polar coordinates r lies in the range $[0, \infty)$ while ϕ lies in the range $[0, 2\pi)$. The conversion from 2D cartesian coordinates to polar is given by

$$r = \sqrt{(x - x_0)^2 + (y - y_0)^2},$$
$$\phi = \arctan\left(\frac{y - y_0}{x - x_0}\right).$$

Note: This means $\phi = 0$ lies along the x-axis i.e. y=0.

Spherical Polar Coordinates

3.1.2 API

Conversion

magpie.coords.cart2polar

magpie.coords.**cart2polar**(*x, y[, center=[0., 0.]]*)

Returns the polar coordinates for a given set of cartesian coordinates.

Parameters

x [array] x coordinates.

y [array] y coordinates.

center [list] Center point of polar coordinate grid.

Returns

r [array] Radial coordinate.

phi [array] Phi coordinate.

magpie.coords.cart2sphere

magpie.coords.**cart2sphere**(*x, y, z[, center=[0., 0., 0.]]*)

Return polar coordinates for a given set of cartesian coordinates.

Parameters

x [array] x coordinate.

y [array] y 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].

magpie.coords.ortho2cart

magpie.coords.**ortho2cart**(*x, y*[, *r= 1., fill_value=np.nan*])

Orthographic projection to cartesian coordinates.

Parameters

x [array] X value in the orthographic projection.

y [array] Y value in the orthographic projection.

r [float] Radius of the sphere.

fill_value [float] Fill values outside the sphere with this value.

Returns

z [array] Returns the z value of the cartesian coordinates.

magpie.coords.polar2cart

magpie.coords.**polar2cart**(*r, p[, center=[0., 0.]]*)

Return cartesian coordinates for a given set of polar coordinates.

Parameters

r [array] Radial coordinate.

phi [array] Phi coordinate.

center [list] Center point of polar coordinate grid.

Returns

x [array] x coordinate.

y [array] y coordinate.

magpie.coords.sphere2cart

magpie.coords.**sphere2cart**(*r, phi, theta[, center=[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 [array] Euclidean coordinates.

Rotations

Utility

magpie.coords.usphere_area

magpie.coords.**usphere_area**(*phi_min*, *phi_max*, *theta_min*, *theta_max*)

Returns the area for a 'square' segment of a unit sphere given in spherical coordinates phi, theta.

Parameters

phi_min [float] Minimum latitudinal coordinate in radians (where phi lies [0, 2pi]).

phi_max [float] Maximum latitudinal coordinate in radians (where phi lies [0, 2pi]).

theta_min [float] Minimum longitudinal coordinate in radians (where theta lies [0, pi]).

theta_max [float] Maximum longitudinal coordinate in radians (where theta lies [0, pi]).

Returns

Area [float] Area in square radians.

magpie.coords.sphere2lonlat

magpie.coords.**sphere2lonlat**(*theta*)

Converts the spherical coordinates theta to the longitude and latitude convention (where theta lies [-pi/2., pi/2.].

Parameters

theta [array] Latitude given in the range [0., pi] where theta = 0 at the north pole.

Returns

Latitude [array] Latitude given in the range [-pi/2, pi/2].

magpie.coords.lonlat2sphere

magpie.coords.**lonlat2sphere**(*latitude*)

Converts from latitude to spherical coordinate convention.

Parameters

Latitude [array] Latitude given in the range [-pi/2, pi/2].

Returns

theta [array] Latitude given in the range [0., pi] where theta = 0 at the north pole.

3.2 Grids

3.2.1 Theory

3.2.2 Tutorial

3.2.3 API

Cartesian

magpie.grids.get_xedges

magpie.grids.**get_xedges**(*length*, *ngrid*[, *xmin=0.*])

Returns the edges of a uniform grid along one axis.

Parameters

length [float] Length of the grid.

ngrid [int] Number of grid points.

xmin [float, optional] Minimum value of the grid.

Returns

xedges [array] The edges of a uniform grid along one axis.

magpie.grids.xedges2mid

magpie.grids.**xedges2mid**(*xedges*)

Returns the mid-point of a uniform grid from the grid edges.

Parameters

xedges [array] The edges of a uniform grid along one axis.

Returns

xmid [array] The mid-point of a uniform grid.

magpie.grids.xmid2edges

magpie.grids.**xmid2edges**(*xmid*)

Returns the mid-point of a uniform grid from the grid edges.

Parameters

xmid [array] The mid-point of a uniform grid.

Returns

xedges [array] The edges of a uniform grid along one axis.

magpie.grids.grid1d

magpie.grids.**grid1d**(*length*, *ngrid*[, *xmin=0.*, *return_edges=False*])

Returns the edges of a uniform grid along one axis.

Parameters

Length [float] Length of the grid.

ngrid [int] Minimum value of the grid.

xmin [float, optional] Minimum value of the grid.

return_edges [bool, optional] If True then the edges of the uniform grid are also supplied.

Returns

xmid [array] The mid-point of a uniform grid.

xedges [array, optional] The edges of a uniform grid along one axis.

magpie.grids.grid2d

magpie.grids.**grid2d**(*lengths, ngrids[, mins=[None, None], return1d=False]*)

Returns the mid-point of a uniform grid.

Parameters

lengths [float or list[float]] Length of the grid.

ngrids [int or list[int]] Number of grid points.

mins [list[float], optional] Minimum values of the grid along each axis.

return1d [bool, optional] Returns 1d mid-points.

Returns

x2d, y2d [2darray] The uniform 2d grid.

xmid, ymid [array, optional] The mid-point of the uniform grid.

magpie.grids.grid3d

magpie.grids.**grid3d**(*lengths, ngrids[, mins=[None, None, None], return1d=False]*)
Returns the mid-point of a uniform grid.

Parameters

lengths [float or list[float]] Length of the grid.

ngrids [int or list[int]] Number of grid points.

mins [list[float], optional] Minimum values of the grid along each axis.

return1d [bool, optional] Returns 1d mid-points.

Returns

x3d, y3d, z3d [3darray] The uniform 3d grid.

xmid, ymid, zmid [array, optional] The mid-point of the uniform grid.

Polar

magpie.grids.polargrid

magpie.grids.**polargrid**(*nr*, *nphi*[, *rmin=0.*, *rmax=1.*, *phimin=0.*, *phimax=2*np.pi*, *returns1d=False*])
Constructs a polar coordinate grid.

Parameters

nr [int] Number of divisions along the r-axis.

nphi [int] Number of divisions along the p-axis.

rmin [float, optional] Minimum radial value, default=0.

rmax [float, optional] Maximum radial value, default=1.

phimin [float, optional] Minimum phi value, default=0.

phimax [float, optional] Maximum phi value, default=2pi.

return1d [bool, optional] Returns 1d mid-points.

Returns

r2d, p2d [2darray] Radial and phi grid points in 2darray.

rmid, pmid [array, optional] The mid-point of the polar grid.

magpie.grids.polarEA_grid

magpie.grids.**polarEA_grid**(*nr*[, *rmax=1.*, *base_nphi=4*])
Constructs an equal area polar grid.

Parameters

nr [int] Number of divisions along the r-axis.

rmax [float, optional] Maximum radial value, default=1.

base_nphi [int, optional] Number of pixels around r=0, default=4.

Returns

r, p [array] Radial and phi mid points for each pixel in the equal area polar grid.

magpie.grids.polarEA_area

magpie.grids.**polarEA_area**(*nr*[, *rmax=1.*, *base_nphi=4*])
Returns the area for an equal area polar grid.

Parameters

nr [int] Number of divisions along the r-axis.

rmax [float, optional] Maximum radial value, default=1.

base_nphi [int, optional] Number of pixels around r=0, default=4.

Returns

area [float] The area of pixels in the equal area polar grid.

magpie.grids.polarEA_npix

magpie.grids.**polarEA_npix**(*nr*[, *rmax=1.*, *base_nphi=4*])
Returns the number of pixels in an equal area polar grid.

Parameters

nr [int] Number of divisions along the r-axis.

base_nphi [int, optional] Number of pixels around r=0, default=4.

Returns

npix [int] Number of pixels in an equal area polar grid.

3.3 Pixel

3.3.1 Theory

3.3.2 Tutorial

3.3.3 API

Pixel Shapes

magpie.pixel.get_square

magpie.pixel.**get_square**(*xmin*, *xmax*, *ymin*, *ymax*[, *steps=40*])
Returns the coordinates of the perimeter of a rectangle.

Parameters

xmin [float] Minimum x-value.

xmax [float] Maximum x-value.

ymin [float] Minimum y-value.

ymax [float] Maximum y-value.

steps [int, optional] Number of divisions across each line segment, default = 4.

Returns

x, y [array] Perimeter coordinates of a rectangle.

magpie.pixel.get_box

`magpie.pixel.get_box`(*xmin, xmax, ymin, ymax, zmin, zmax[, steps=120, return_nearest=False, center=[0., 0., 0.]]*)

Returns 3D coordinates for a box.

Parameters

xmin [float] Minimum x value.

xmax [float] Maximum x value.

ymin [float] Minimum y value.

ymax [float] Maximum y value.

zmin [float] Minimum z value.

zmax [float] Maximum z value.

steps [int, optional] Number of divisions in each line element.

return_nearest [bool, optional] If True only the lines for the nearest or visible faces from a defined center will be outputed.

center [list, optional] Coordinate center used for return_nearest=True

Returns

x, y, z [array] Coordinates of the box edges.

magpie.pixel.get_arc

`magpie.pixel.get_arc`(*radius[, phimin=0., phimax=2.*np.pi, center=[0., 0.], steps=10]*)

Returns the coordinates of the arc of a circle, default settings will return the coordinates along a complete circle.

Parameters

radius [float] Radius of the circle.

phimin [float, optional] Minimum phi value, default=0.

phimax [float, optional] Maximum phi value, default=2pi.

center [list, optional] Central x and y coordinates for the arc.

steps [int, optional] The number of points along the arc curve.

Returns

x, y [array] Coordinates along the arc of a circle.

magpie.pixel.get_disc

magpie.pixel.**get_disc**(*[rmin=0., rmax=1., phimin=0., phimax=2.*np.pi, center=[0., 0.], steps=40]*)
Returns the coordinates of a disc segment.

Parameters

rmin [float, optional] Minimum radial value, default=0.

rmax [float, optional] Maximum radial value, default=1.

phimin [float, optional] Minimum phi value, default=0.

phimax [float, optional] Maximum phi value, default=2pi.

center [list, optional] Central x and y coordinates for the arc.

steps [int, optional] The number of points along the arc curve.

Returns

x, y [array] Coordinates along the arc of a circle.

magpie.pixel.healpix_boundary_bot

healpix_boundary_bot(*p*, *nside*[, *steps=20*, *reverse=False*])
Returns the bottom side of a healpix pixel in healpix x and y coordinates.

Parameters

p [int] Healpix pixel index.

nside [int] Healpix Nside.

steps [int, optional] Number of steps for the top function.

reverse [bool, optional] Reverse the order so the output has descending x.

Returns

x [array] X-coordinates of the bottom side of the healpix pixel.

y [array] Y-coordinates of the bottom side of the healpix pixel.

magpie.pixel.healpix_boundary_top

healpix_boundary_top(*p*, *nside*[, *steps=20*, *reverse=False*])
Returns the top side of a healpix pixel in healpix x and y coordinates.

Parameters

p [int] Healpix pixel index.

nside [int] Healpix Nside.

steps [int, optional] Number of steps for the top function.

reverse [bool, optional] Reverse the order so the output has descending x.

Returns

x [array] X-coordinates of the top side of the healpix pixel.

y [array] Y-coordinates of the top side of the healpix pixel.

magpie.pixel.healpix_boundary

healpix_boundary(*p*, *nside*[, *steps=40*, *reverse=False*])

Returns the boundary of a healpix pixel in healpix x and y coordinates, default given in clockwise directions.

Parameters

p [int] Healpix pixel index.

nside [int] Healpix Nside.

steps [int, optional] Number of steps.

reverse [bool, optional] Reverse the order so the output is anti-clockwise.

Returns

x [array] X-coordinates of the boundaries of the healpix pixel.

y [array] Y-coordinates of the boundaries of the healpix pixel.

Pixel Indices

3.4 Randoms

3.4.1 Theory

3.4.2 Tutorial

3.4.3 API

Cartesian

magpie.randoms.randoms_1d

magpie.randoms.**randoms_1d**(*size*[, *xmin=0.*, *xmax=1.*])

Returns uniform randoms in 1D.

Parameters

size [int] Number of randoms to produce.

xmin [float, optional] Minimum value.

xmax [float, optional] Maximum value.

Returns

xrands [array] Uniform randoms in 1D.

magpie.randoms.randoms_2d

magpie.randoms.**randoms_2d**(*size[, mins=[0., 0.], maxs=[1., 1.]]*)

Returns uniform randoms in 2D.

Parameters

size [int] Number of randoms to produce.

mins [float, optional] Minimum values.

maxs [float, optional] Maximum values.

Returns

xrands, yrands [array] Uniform randoms in 2D.

magpie.randoms.randoms_3d

magpie.randoms.**randoms_3d**(*size[, mins=[0., 0., 0.], maxs=[1., 1., 1.]]*)

Returns uniform randoms in 3D.

Parameters

size [int] Number of randoms to produce.

mins [float, optional] Minimum values.

maxs [float, optional] Maximum values.

Returns

xrands, yrands, zrands [array] Uniform randoms in 3D.

Polar

magpie.randoms.randoms_polar

magpie.randoms.**randoms_polar**(*size*[, *r_min=0.*, *r_max=1.*, *phi_min=0.*, *phi_max=2.*np.pi*])

Generates randoms for polar coordinates. Default will produce randoms within a unit circle. This can be specified to a ring segment, i.e. with inner radius r_min and outer radius r_max and specifying the angle of the ring segment.

Parameters

size [int] Number of randoms.

r_min [float] Minimum r.

r_max [float] Maximum r.

phi_min [float] Minimum phi.

phi_max [float] Maximum phi.

Returns

r [array] Random radial coordinates.

phi [array] Random phi coordinates.

Unit Sphere

magpie.randoms.randoms_usphere

magpie.randoms.**randoms_usphere**(*size*[, *phi_min=0.*, *phi_max=2.*np.pi*, *theta_min=0.*, *theta_max=np.pi*])
Random points on the unit sphere or more generally across the surface of a sphere. The default will give randoms on the full unit sphere.

Coordinate convention:

- phi lies in the range [0, 2pi]
- theta lies in the rang [0, pi].

Parameters

size [int] Number of randoms to generate.

phi_min [float] Minimum longitude in radians.

phi_max [float] Maximum longitude in radians.

theta_min [float] Minimum latitude in radians.

theta_max [float] Maximum longitude in radians.

Returns

phi [array] Random phi.

theta [array] Random theta.

magpie.randoms.randoms_healpix_pixel

magpie.randoms.**randoms_healpix_pixel**(*size*, *pix*, *nside*)
Returns roughly *size* number of randoms inside a HEALPix pixel.

Parameters

size [int] Average number of randoms per pixel.

p [int] Pixel identifier for healpix map.

nside [int] Nside of the healpix map.

Returns

phi [array] Random phi (latitude angle) in radians.

theta [array] Random theta (longitude angle) in radians.

Spherical

magpie.randoms.randoms_sphere_r

magpie.randoms.**randoms_sphere_r**(*size*[, *r_min=0.*, *r_max=1.*])

Random radial points for a segment of a sphere (default will give randoms within a unit sphere).

Parameters

size [int] Number of randoms to generate.

r_min [float] Minimum radius.

r_max [float] Maximum radius.

Returns

r [array] Random r.

magpie.randoms.randoms_sphere

magpie.randoms.**randoms_sphere**(*size*[, *r_min=0.*, *r_max=1.*, *phi_min=0.*, *phi_max=2*np.pi*, *theta_min=0.*, *theta_max=np.pi*])

Random points inside a sphere (default will give randoms within a unit sphere). You can specify the inner and outer radii to get randoms in a shell and the region on the sky.

Coordinate convention:

- phi lies in the range [0, 2pi]
- theta lies in the rang [0, pi].

Parameters

size [int] Number of randoms to generate.

r_min [float] Minimum radius.

r_max [float] Maximum radius.

phi_min [float] Minimum longitude in radians.

phi_max [float] Maximum longitude in radians.

theta_min [float] Minimum latitude in radians.

theta_max [float] Maximum longitude in radians.

Returns

r [array] Random r.

phi [array] Random phi.

theta [array] Random theta.

Sample PDF/CDF

magpie.randoms.pdf2cdf

magpie.randoms.**pdf2cdf**(*xmid*, *pdf*[, *return_normpdf=True*])
Calculates the CDF from a given PDF.

Parameters

xmid [array] Linearly spaced x-values given at the middle of a bin of length dx.

pdf [array] Probabilty distribution function.

return_normpdf [bool, optional] Normalise PDF is also outputed.

Returns

x [array] X-coordinates.

cdf [array] Cumulative distribution function with extreme points set 0 and 1.

normpdf [array, optional] Normalised PDF.

magpie.randoms.randoms_cdf

magpie.randoms.**randoms_cdf**(*x*, *cdf*, *size*[, *kind='cubic'*])
Generates randoms from a given cumulative distribution function.

Parameters

x [array] X-coordinates.

cdf [array] Cumulative distribution function, extreme points must be 0 and 1 i.e. cdf[0] = 0 and cdf[-1] = 1.

size [int] Size of the random sample.

kind [str, optional] Scipy CDF interpolation kind.

Returns

rands [array] Randoms drawn from sample CDF.

magpie.randoms.randoms_pdf

magpie.randoms.**randoms_pdf**(*x*, *pdf*, *size*[, *kind='cubic'*])
Generates randoms from a given probability distribution function by first calculating a CDF.

Parameters

xmid [array] Linearly spaced x-values given at the middle of a bin of length dx.

pdf [array] Probabilty distribution function.

size [int] Size of the random sample.

kind [str, optional] Scipy CDF interpolation kind.

Returns

rands [array] Randoms drawn from sample PDF.

Subsampling

magpie.randoms.shuffle

magpie.randoms.**shuffle**(*sample*)
Shuffles the ordering of a sample.

Parameters

sample [array] Input sample data.

magpie.randoms.random_draw

magpie.randoms.**random_draw**(*sample*, *size*)
Draws a random sample from an input function, the algorithm ensures there can be no repeats.

Parameters

sample [array] Input sample data.

size [int] Size of the random draws.

Returns

randsamp [array] Random subsample.

magpie.randoms.random_prob_draw

magpie.randoms.**random_prob_draw**(*sample*, *prob*, *size*)
Probabilistic draw from an input sample.

Parameters

sample [array] Input sample data.

prob [array, optional] The probability assigned to each sample.

size [int, optional] Size of the probabilistic draw.

Returns

randsamp [array] Random subsample.

magpie.randoms.stochastic_integer_weights

magpie.randoms.**stochastic_integer_weights**(*weights*)
Returns stochastic integer weights for an input weight. This is useful for point processes that require integer weights, where a non-integer weight can be achieved by superposition of many realisations.

Parameters

weights [array] Input weights.

Returns

weights_SI [array] Stochastic integer weights.

magpie.randoms.stochastic_binary_weights

magpie.randoms.**stochastic_binary_weights**(*weights*)

Returns stochastic binary integer weights for an input weight. This is useful for point processes that require binary integer weights, where a non-integer weight can be achieved by superposition of many realisations.

Parameters

weights [array] Input weights.

Returns

weights_SB [array] Stochastic binary weights.

3.5 Remap

Remapping data from one coordinate system mesh to another can play a critical role in data science. Converting one dimensional data, image-like data in two dimensions and three dimensional mesh data into another coordinate systems can be critical in enabling certain analysis pipelines to be calculated and on a more fundamental level enable better intuitive understanding of certain phenomena. Typically we assume that data provided on a grid can be defined by the cells (bins in 1D and pixels in 2D) center and map those central positions into a new coordinate system. However, this fails to take into account the pixel's volume (length in 1D and area in 2D) and can lead to some strange artifacts (such as empty pixels in the new coordinate mesh). A possible solution to this problem is to use interpolation however while interpolation works well for theoretically generated data, for measured data this makes the implicit assumption that data measured and binned in a cell is equivalent to the value at the cells center. In general this assumption is not hugely problematic but can lead to some issues particularly when the cells are integrated values across the cell. In `magpie` we take a different approach, we consider each pixels volume and map to another coordinate system mesh by taking the volume-weighted mean of overlapping pixels in the original coordinate system. This is achieved through two methods, the first uses monte carlo (MC) integration to determine the weights but for large data sets this can be slow so a second scheme approximates these weights by using a higher resolution (HR) mesh.

3.5.1 Theory

Remapping

Remapping data d (defined in coordinate system X) to a new coordinate system Y is calculated by taking the weighted mean,

$$\tilde{d}(Y_j) = \sum_{i=1}^{N} w(X_i, Y_j) d(X_i),$$

where $\tilde{d}$ is d mapped onto coordinate system Y and i and j indicate pixels in X and Y respectively. The weights $w(X, Y)$ are given by

$$w(X, Y) = \frac{A(X \cap Y)}{A(Y)},$$

where $A(Y)$ is the area (length in 1D and volume in 3D) of pixel Y and $A(X \cap Y)$ is the area of the overlapping region for pixels X and Y. For variance we can use propagation of errors to determine the variance on the remapped coordinate system although note we assume covariant terms are zero. The remapped variance is given by

$$\tilde{\Delta d}^2(Y_j) = \sum_{i=1}^{N} U(X_i, Y_j) w^2(X_i, Y_j) \Delta d^2(X_i).$$

Note, the variance for a subset of X called s is given by the relation

$$\Delta d^2(s) = \frac{A(s)}{A(X)} \Delta d^2(X),$$

this assumes that the variance observed in a bin is actually the standard error on the mean for that bin. The above relation shows that dividing by the area gives a normalised variance,

$$\delta d^2(X) = \frac{1}{A(X)} \Delta d^2(X),$$

and it follows for $\tilde{\delta d}(Y) \equiv \delta d(X)$ for constant variance that $U(X_i, Y_j) = A(X_i)/A(Y_j)$ which means,

$$\tilde{\Delta d}^2(Y_j) = \sum_{i=1}^{N} \frac{A(X_i)}{A(Y_j)} w^2(X_i, Y_j) \Delta d^2(X_i).$$

Weight Calculation

Monte-Carlo Method

Weights are computed by monte carlo integration – random points are distributed equally in each pixel Y and the weights are simply the normalised counts which fall into pixels in X.

Higher-Resolution Method

A higher resolution mesh, determined by some input integer factor N_H, is constructed. This means for the new coordinate system Y there are N_H HR pixels in each of the coordinate dimensions. The weights are simply given by

$$w(X_i, Y_j) = \frac{1}{A(Y_j)} \sum_{k=1}^{N_H^D} A(y_k) S(y_k, X_i)$$

where D is the dimension of the coordinate system, y indicates pixels in the higher resolution mesh and $S(y_k, X_i) = 1$ if the pixel center for y is within X_i and 0 otherwise.

3.5.2 Tutorial

3.5.3 API

3.6 API

3.6.1 Coords

3.6.2 Grids

3.6.3 Randoms

3.6.4 Remap

CHAPTER

FOUR

DEPENDENCIES

MAGPIE is being developed in Python 3.9 but should work on all versions >=3.4. MAGPIE is written mostly in python but with some heavy computation carried out in Fortran. Compiling the Fortran source code will require the availability of a fortran compiler such as gfortran or ifort.

The following Python modules are required:

- numpy
- scipy
- healpy

For testing you will require nose or pytest .

CHAPTER

FIVE

INSTALLATION

MAGPIE can be installed in a variety of ways – using `pip`, `conda` or by directly cloning the repository. If you are having trouble installing MAGPIE or running MAGPIE we recommend using the conda install as this will setup the environment.

1. Using `pip`:

```
pip install magpie-pkg
```

2. Using `conda`:

```
conda install -c knaidoo29 magpie-pkg
```

3. By cloning the github repository:

```
git clone https://github.com/knaidoo29/magpie.git
cd magpie
python setup.py build
python setup.py install
```

Once this is done you should be able to call MAGPIE from python:

```
import magpie
```

CHAPTER

SIX

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

SEVEN

VERSION HISTORY

- **Version 0.2:**
  - Restructured layout and the incorporation of documentation and unit testing for eventual first release.
  - Randoms in a variety of coordinate systems: cartesian, polar, spherical and from an input PDF/CDF and subsampling and stochastic weights.
  - Cartesian coordinate remapping in 1D, 2D and 3D.
- **Version 0.1:**
  - Coordinate transformations between cartesian, polar and spherical polar coordinates.
  - Rebinning in 1D (computed exactly), in 2D and 3D via monte-carlo weighted remapping and a higher-resolution mesh.
  - Randoms in cartesian, polar and spherical polar coordinates.
  - Rotation transformations.
  - Polar coordinate utilities and integration for polar grid to radial profiles.
  - Plotting routine for orthographic projection of unit sphere data.

INDEX

B

built-in function
    healpix_boundary(), 16
    healpix_boundary_bot(), 15
    healpix_boundary_top(), 15
    magpie.coords.cart2polar(), 7
    magpie.coords.cart2sphere(), 8
    magpie.coords.lonlat2sphere(), 10
    magpie.coords.ortho2cart(), 8
    magpie.coords.polar2cart(), 8
    magpie.coords.sphere2cart(), 9
    magpie.coords.sphere2lonlat(), 9
    magpie.coords.usphere_area(), 9
    magpie.grids.get_xedges(), 10
    magpie.grids.grid1d(), 11
    magpie.grids.grid2d(), 11
    magpie.grids.grid3d(), 12
    magpie.grids.polarEA_area(), 13
    magpie.grids.polarEA_grid(), 12
    magpie.grids.polarEA_npix(), 13
    magpie.grids.polargrid(), 12
    magpie.grids.xedges2mid(), 10
    magpie.grids.xmid2edges(), 11
    magpie.pixel.get_arc(), 14
    magpie.pixel.get_box(), 14
    magpie.pixel.get_disc(), 15
    magpie.pixel.get_square(), 13
    magpie.randoms.pdf2cdf(), 20
    magpie.randoms.random_draw(), 21
    magpie.randoms.random_prob_draw(), 21
    magpie.randoms.randoms_1d(), 16
    magpie.randoms.randoms_2d(), 17
    magpie.randoms.randoms_3d(), 17
    magpie.randoms.randoms_cdf(), 20
    magpie.randoms.randoms_healpix_pixel(), 18
    magpie.randoms.randoms_pdf(), 20
    magpie.randoms.randoms_polar(), 17
    magpie.randoms.randoms_sphere(), 19
    magpie.randoms.randoms_sphere_r(), 19
    magpie.randoms.randoms_usphere(), 18
    magpie.randoms.shuffle(), 21
    magpie.randoms.stochastic_binary_weights(), 22
    magpie.randoms.stochastic_integer_weights(), 21

H

healpix_boundary()
    built-in function, 16
healpix_boundary_bot()
    built-in function, 15
healpix_boundary_top()
    built-in function, 15

M

magpie.coords.cart2polar()
    built-in function, 7
magpie.coords.cart2sphere()
    built-in function, 8
magpie.coords.lonlat2sphere()
    built-in function, 10
magpie.coords.ortho2cart()
    built-in function, 8
magpie.coords.polar2cart()
    built-in function, 8
magpie.coords.sphere2cart()
    built-in function, 9
magpie.coords.sphere2lonlat()
    built-in function, 9
magpie.coords.usphere_area()
    built-in function, 9
magpie.grids.get_xedges()
    built-in function, 10
magpie.grids.grid1d()
    built-in function, 11
magpie.grids.grid2d()
    built-in function, 11
magpie.grids.grid3d()
    built-in function, 12
magpie.grids.polarEA_area()
    built-in function, 13
magpie.grids.polarEA_grid()
    built-in function, 12

magpie.grids.polarEA_npix()
    built-in function, 13
magpie.grids.polargrid()
    built-in function, 12
magpie.grids.xedges2mid()
    built-in function, 10
magpie.grids.xmid2edges()
    built-in function, 11
magpie.pixel.get_arc()
    built-in function, 14
magpie.pixel.get_box()
    built-in function, 14
magpie.pixel.get_disc()
    built-in function, 15
magpie.pixel.get_square()
    built-in function, 13
magpie.randoms.pdf2cdf()
    built-in function, 20
magpie.randoms.random_draw()
    built-in function, 21
magpie.randoms.random_prob_draw()
    built-in function, 21
magpie.randoms.randoms_1d()
    built-in function, 16
magpie.randoms.randoms_2d()
    built-in function, 17
magpie.randoms.randoms_3d()
    built-in function, 17
magpie.randoms.randoms_cdf()
    built-in function, 20
magpie.randoms.randoms_healpix_pixel()
    built-in function, 18
magpie.randoms.randoms_pdf()
    built-in function, 20
magpie.randoms.randoms_polar()
    built-in function, 17
magpie.randoms.randoms_sphere()
    built-in function, 19
magpie.randoms.randoms_sphere_r()
    built-in function, 19
magpie.randoms.randoms_usphere()
    built-in function, 18
magpie.randoms.shuffle()
    built-in function, 21
magpie.randoms.stochastic_binary_weights()
    built-in function, 22
magpie.randoms.stochastic_integer_weights()
    built-in function, 21