AstroPaint

Sep 16, 2020

Contents:

A python package for painting the sky

You can install **AstroPaint** by running the following in the command line:

```
git clone https://github.com/syasini/AstroPaint.git
cd AstroPaint
pip install [-e] .
```

the `-e` argument will install the package in editable mode which is suitable for developement. If you want to modify the code use this option.

Important Note: If you want the sample catalogs to be cloned automatically along with the rest of the repository, make sure you have Git Large File Storage (git lfs) installed.

If you are a conda user, please consider creating a new environment before installation:

```
conda create -n astropaint python=3.7
conda activate astropaint
```

CHAPTER 1

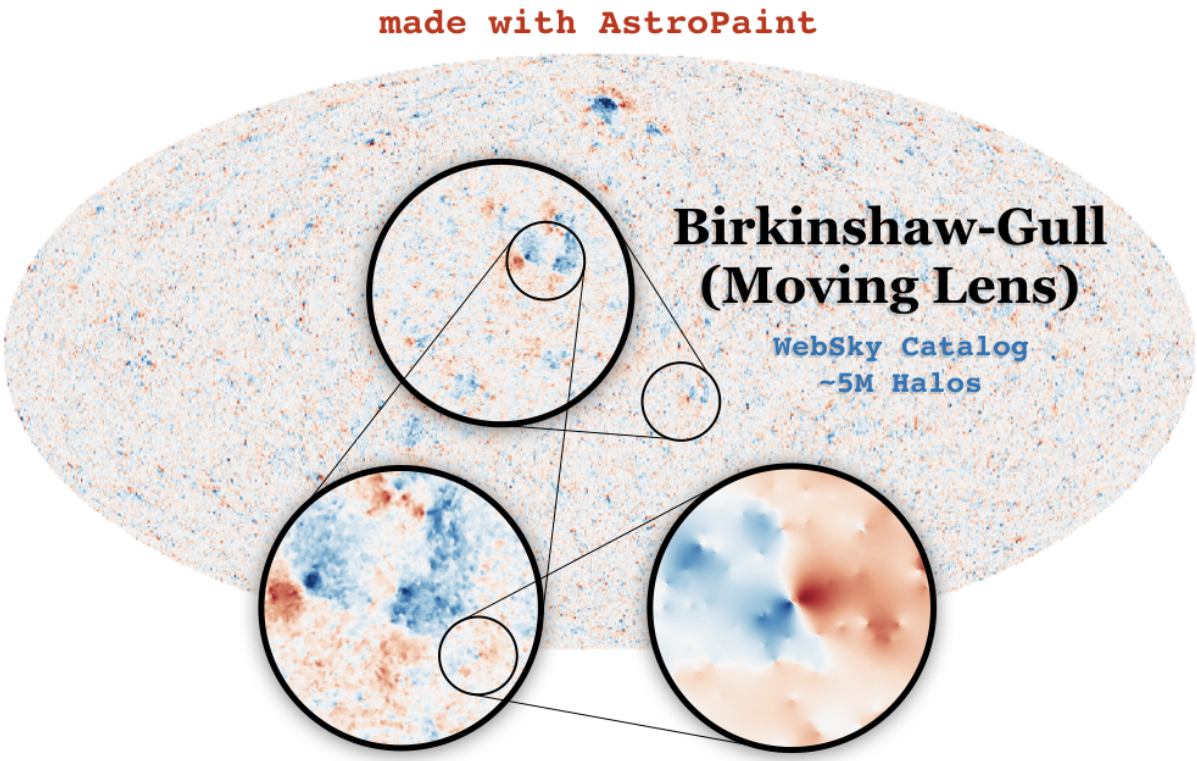
Workflow

Converting catalogs to mock maps with AstroPaint is extremely simple. Here is what an example session looks like:

```
from astropaint import Catalog, Canvas, Painter

catalog = Catalog(data=your_input_data)

canvas = Canvas(catalog, nside)

painter = Painter(template=your_radial_profile)

painter.spray(canvas)
```

That's it! Now you can check out your masterpiece using

```
canvas.show_map()
```


made with AstroPaint
Birkinshaw-Gull
(Moving Lens)
WebSky Catalog
~5M Halos

CHAPTER 2

What is AstroPaint?

AstroPaint is a python package for generating and visualizing sky maps of a wide range of astrophysical signals originating from dark matter halos or the gas that they host. AstroPaint creates a whole-sky mock map of the target signal/observable, at a desired resolution, by combining an input halo catalog and the radial/angular profile of the astrophysical effect. The package also provides a suite of tools that can facilitate analysis routines such as catalog filtering, map manipulation, and cutout stacking. The simulation suite has an Object-Oriented design and runs in parallel, making it both easy to use and readily scalable for production of high resolution maps with large underlying catalogs. Although the package has been primarily developed to simulate signals pertinent to galaxy clusters, its application extends to halos of arbitrary size or even point sources.

CHAPTER 3

Package Structure

While there is no external documentation for the code yet, you can use this chart to understand the package structure and see what methods are available so far.

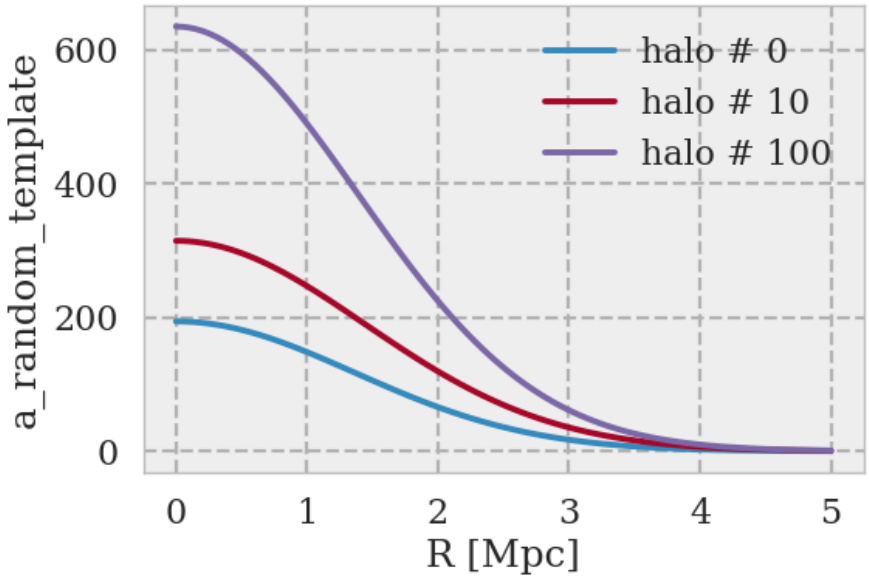
CHAPTER 4

Examples

4.1 Nonsense Template

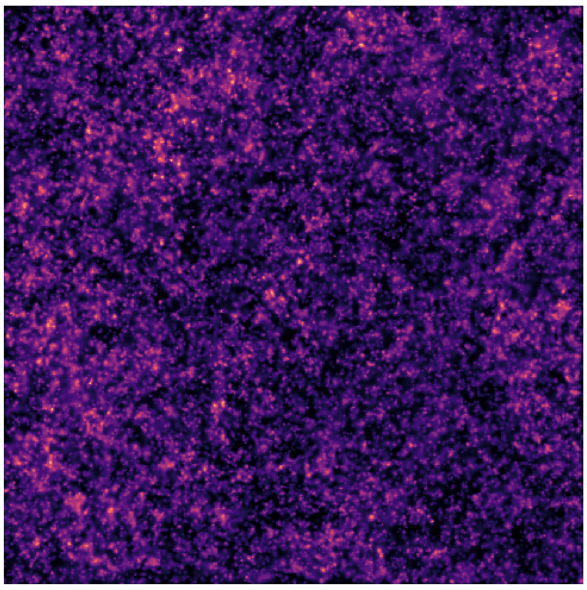
Here's an example script that paints a nonsense template on a 10 x 10 [sqr deg] patch of the `Sehgal` catalog:

```
import numpy as np
from astropaint import Catalog, Canvas, Painter

# Load the Sehgal catalog
catalog = Catalog("Sehgal")

# cutout a 10x10 sqr degree patch of the catalog
catalog.cut_lon_lat(lon_range=[0,10], lat_range=[0,10])

# pass the catalog to canvas
canvas = Canvas(catalog, nside=4096, R_times=5)

# define a nonsense template and plot it
def a_nonsense_template(R, R_200c, x, y, z):

    return np.exp(-(R/R_200c/3)**2)*(x+y+z)

# pass the template to the painter
painter = Painter(template=a_nonsense_template)

# plot the template for halos #0, #10, and #100 for R between 0 to 5 Mpc
R = np.linspace(0,5,100)
painter.plot_template(R, catalog, halo_list=[0,10,100])
```



The painter automatically extracts the parameters *R_200c* and *x,y,z* coordinates of the halo from the catalog that the canvas was initialized with. Let's spray ths canvas now:

```
# spray the template over the canvas
painter.spray(canvas)

# show the results
canvas.show_map("cartview", lonra=[0,10], latra=[0,10])
```



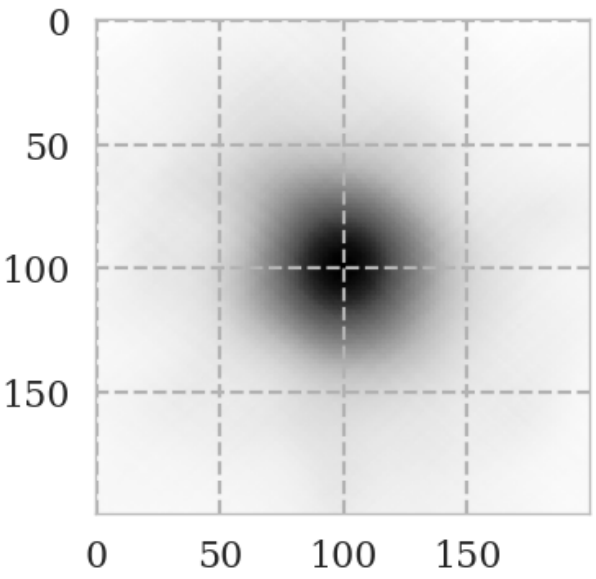
Voila!

You can use the *n_cpus* argument in the spray function to paint in parallel and speed things up! The default value *n_cpus=-1* uses all the available cpus.

4.2 Stacking

You can easily stack cutouts of the map using the following:

```
deg_range = [-0.2, 0.2] # deg
halo_list = np.arange(5000) # stack the first 5000 halos

# stack the halos and save the results in canvas.stack
stack = canvas.stack_cutouts(halo_list=halo_list, lon_range=deg_range, lat_range=deg_
↪range)

plt.imshow(canvas.stack)
```



If this is taking too long, use *parallel=True* for *parallel stacking*.

4.3 Line-Of-Sight integration of 3D profiles

AstroPaint only allows you to paint 2D (line-of-sight integrated) profiles on your catalog halos, so if you already have the analytical expression of the projected profile you want to paint, we are in business. However, not all 3D profiles can be LOS integrated analytically (e.g. generalized NFW or Einasto, etc), and integrating profiles numerically along every single LOS is generally expensive. In order to alleviate this problem, AstroPaint offers two python decorators *@LOS_integrate* and *@interpolate* which make 3D -> 2D projections effortless.

To convert a 3D profile into a 2D LOS integrated profile, all you need to do is add the *@LOS_integrate* to the definition.

For example, here's how you can turn a 3D top hat profile

```
def tophat_3D(r, R_200c):
    """Equals 1 inside R_200c and 0 outside"""

    tophat = np.ones_like(r)
    tophat[r > R_200c]=0

    return tophat
```

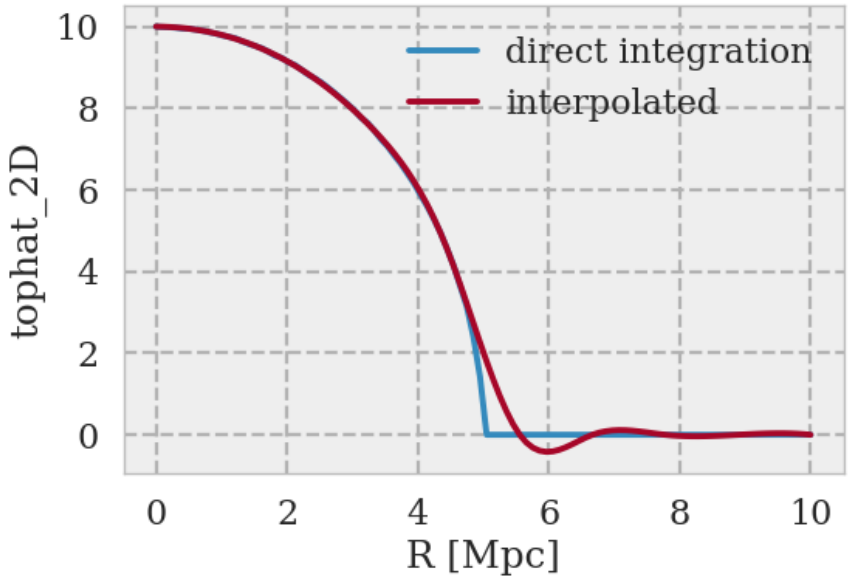
into a 2D projected one:

```
from astropaint.lib.utilities import LOS_integrate

@LOS_integrate
def tophat_2D(R, R_200c):
    """project tophat_3D along the line of sight"""

    return tophat_3D(R, R_200c)
```

This function integrates the *tophat_3D* function along every single line of sight. If you have many halos in a high resolution map, this can take forever. The trick to make this faster would be to integrate along a several LOSs and interpolate the values in between. This is what the *@interpolate* decorator does. So, a faster version of the *tophat_2D* function can be constructed as the following:

```
from astropaint.lib.utilities import interpolate

@interpolate(n_samples=20)
@LOS_integrate
def tophat_2D_interp(R, R_200c):
    """project and interpolate tophat_3D along the line of sight"""

    return tophat_3D(R, R_200c)
```

This is much faster, but the speed comes at a small price. If your 3D profile is not smooth, the interpolated 2D projection will slightly deviate from the exact integration.



You can minimize this deviation by increasing the *n_samples* argument of the *@interpolate* decorator, but that will obviously decrease the painting speed.

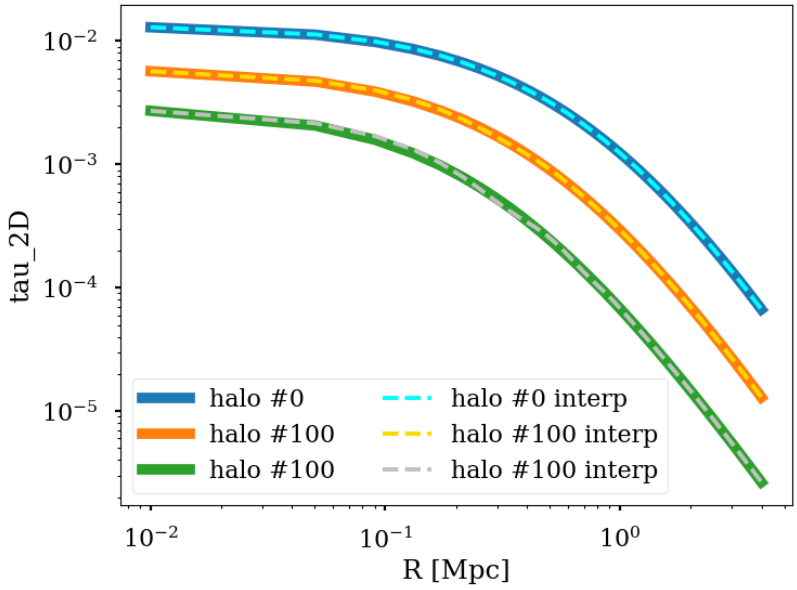
Does this plot agree with what you would expect a LOS integrated top hat profile (a.k.a. a solid sphere) to look like?

4.4 Painting Optical Depth and kSZ Profiles on the WebSky Catalog

Let's use the *Battaglia16* gas profiles to paint tau (optical depth) and kinetic Sunyaev-Zeldovich (kSZ) on the WebSky catalog halos.

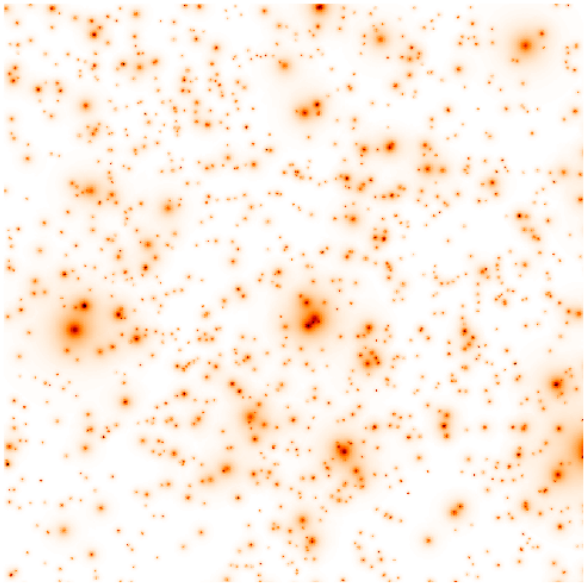
```
from astropaint.profiles import Battaglia16

 tau_painter = Painter(Battaglia16.tau_2D_interp)
```

Since the shape of the profile is smooth, we won't lose accuracy by using the interpolator.



Let's paint this on a 5x5 sqr deg patch of the WebSky catalog with a mass cut of 8E13 M_sun.

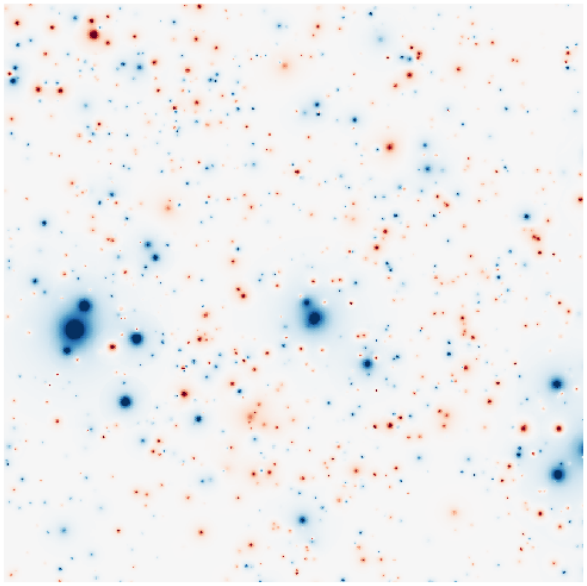
```
catalog = Catalog("websky_lite_redshift")
catalog.cut_lon_lat(lon_range=[5,10], lat_range=[5,10])
catalog.cut_M_200c(8E13)

canvas = Canvas(catalog, nside=8192, R_times=3)

tau_painter.spray(canvas)
```



The *Battaglia16.kSZ_T* function uses this tau and multiplies it by the dimensionless velocity of the halos to get the kSZ signal.

```
kSZ_painter = Painter(Battaglia16.kSZ_T)
kSZ_painter.spray(canvas)
```

And here is what it looks like:

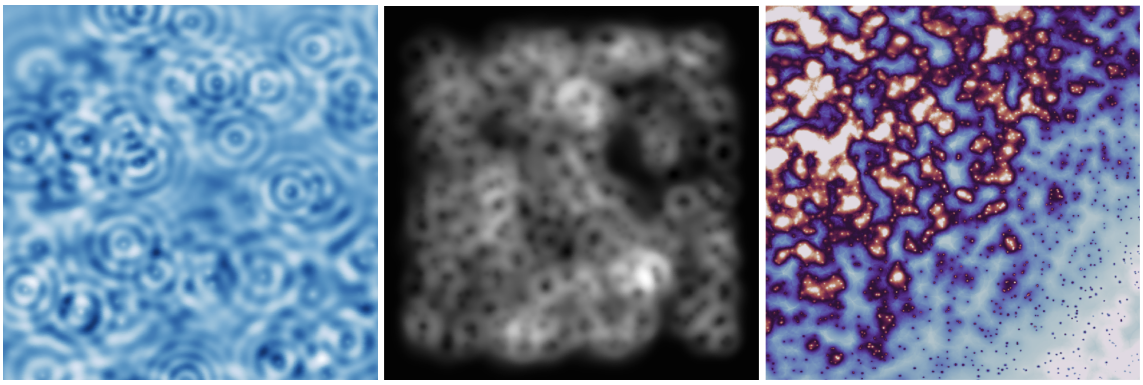


CHAPTER 5

Art Gallery

Just because AstroPaint is developed for probing new science and doing serious stuff, it doesn't mean you can't have fun with it! Check out our cool web app to get your hands dirty with some paint.

Made with AstroPaint



CHAPTER 6

How to contribute

If you would like to contribute to AstroPaint, take the following steps:

1) Fork this repository
2) Clone it on your local machine
3) Create a new branch (be as explicit as possible with the branch name)
4) Add and Commit your changes to the local branch
5) Push the branch to your forked repository
6) Submit a pull request on this repository

See this repository or Kevin Markham's step-by-step guide for more detailed instructions.

Developement happens on the `develop` branch, so make sure you are always in sync with the latest version and submit your pull requests to this branch.

6.1 AstroPaint

6.1.1 astropaint package

Subpackages

astropaint.lib package

Submodules

astropaint.lib.log module

library for logging functions and classes

exception astropaint.lib.log.**CMBAlreadyAdded**
Bases: Exception

exception astropaint.lib.log.**NoiseAlreadyAdded**
Bases: Exception

exception astropaint.lib.log.**ParameterNotFound**
Bases: KeyError

astropaint.lib.plot_configs module

astropaint.lib.transform module

library for transforming halo properties and coordinates

astropaint.lib.transform.**D_c_to_D_a**(*D_c*, *redshift*)
calculate the angular diameter distance (D_a) from comoving distance (D_c) and redshift (redshift)

astropaint.lib.transform.**D_c_to_redshift**(*D_c*, *units=Unit("Mpc")*)
calculate the redshift from comoving distance (D_c)

astropaint.lib.transform.**M_200c_to_R_200c**(*M_200c*, *redshift*)
calculate R_200c from M_200c at the given redshift see Eq. 1 in Huang et al 1701.04001

astropaint.lib.transform.**M_200c_to_c_200c**(*M_200c*, *redshift*)
calculate the concentration parameter from M_200c at the given redshift use fitting formula in Eq 19 of Child et al 2018 (1804.10199)

astropaint.lib.transform.**M_200c_to_rho_s**(*M_200c*, *redshift*, *R_200c=None*, *c_200c=None*)
calculate the NFW rho_s parameter from M_200c at the given redshift if R_200c and c_200c are not given, calculate them

astropaint.lib.transform.**M_to_tau**(*M*)

astropaint.lib.transform.**arcmin2rad**(*angle*)
convert arcmins to radians

astropaint.lib.transform.**convert_velocity_cart2sph**(*th*, *ph*, *v_x*, *v_y*, *v_z*)

astropaint.lib.transform.**convert_velocity_sph2cart**(*th*, *ph*, *v_r*, *v_th*, *v_ph*)
Calculate the cartesian velocity components from the spherical ones

astropaint.lib.transform.**fwhm2sigma**(*fwhm*, *arcmin=True*)
Convert fwhm to sigma

Parameters

- **fwhm** (*float*) – Full Width Half Maximum (fwhm)
- **arcmin** (*bool*) – if True, fwhm will be converted from arcmin to rad

Returns **sigma** – fwhm ** 2 / 8 / np.log(2)

Return type float [rad]

astropaint.lib.transform.**get_cart2sph_jacobian**(*th*, *ph*)
calculate the transformation matrix (jacobian) for spherical to cartesian coordinates at line of sight (th, ph) [radians]

see https://en.wikipedia.org/wiki/Vector_fields_in_cylindrical_and_spherical_coordinates

th is the polar angle with respect to z and ph is the azimuthal angle with respect to x

example:

th_rad = np.deg2rad(df['th'].values) ph_rad = np.deg2rad(df['ph'].values)

v_cart = np.array([df['vx'],df['vy'],df['vz']])

thph_grid = np.array([th_rad,ph_rad])

J_cart2sph = cart2sph2(th_rad,ph_rad) v_cart2sph = np.einsum('ij...,i...->j...',J_cart2sph,v_cart)

astropaint.lib.transform.**get_sph2cart_jacobian**(*th*, *ph*)
calculate the transformation matrix (jacobian) for spherical to cartesian coordinates at line of sight (th, ph) [radians]

see https://en.wikipedia.org/wiki/Vector_fields_in_cylindrical_and_spherical_coordinates

th is the polar angle with respect to z and ph is the azimuthal angle with respect to x

example: see cart2sph2

astropaint.lib.transform.**rad2arcmin**(*angle*)
convert radians to arcmins

astropaint.lib.transform.**radius_to_angsize**(*radius*, *D_a*, *arcmin=True*)
calculate the angular radius (theta) of the halo from its radius and angular diameter distance (D_a).

if arcmin == True: return the value in arcmin

*NOTE: radius and D_a must have the same units

astropaint.lib.transform.**rotate_patch**(*patch*, *angle*)
Rotate input patch by angle [deg]

astropaint.lib.transform.**taper_patch**(*patch*)
taper the patch to smooth out the edges

astropaint.lib.utils module

astropaint.lib.utils.**LOS_integrate**(*profile_3D*, **args*, ***kwargs*)
integrate along the line of sight for all 3D r's that correspond to the 2D R

astropaint.lib.utils.**combine_Nl**(*Nls*)
combine the input noise power spectra

Parameters **Nls** (*list*) – list of noise power spectra

Returns

- *combined noise power spectrum*
- *1/N_tot = 1/N1 + 1/N2 + ...*

astropaint.lib.utils.**get_CMB_Cl**(*lmax*, *lmin=0*, *mode='TT'*, *return_ell=False*, *uK=False*)
load Cl from camb generated Dl file

Parameters

- **lmax** (*int*) – max ell number
- **lmin** (*int*) – min ell number
- **mode** (*str*) – CMB mode to return (e.g. "TT", "EE", etc)
- **return_ell** (*bool*) – if True, returns the corresponding ell array as well

Returns

- *Cl [K^2]*
- *or*
- *ell, Cl [K^2]*
- **available keys in Cls** (*L, TT, EE, BB, TE*)

astropaint.lib.utils.**get_custom_B2l**(*fwhm*, *lmax*, *lmin=0*, *arcmin=True*, *return_ell=False*)

Compute the instrumental Beam power spectrum

After smoothing the map with a beam of size fwhm, the power spectrum would be suppressed by a factor

B2l= np.exp(-ell * (ell + 1) * sigma_b)

where sigma_b = fwhm ** 2 / 8 / np.log(2)

Parameters

- **[arcmin]** (*fwhm*) – beam fwhm in arcmins (or radians if arcmin=False)
- **lmax** – maximum ell mode in the power spectrum
- **lmin** – minimum ell mode in the power spectrum
- **arcmin** (*bool*) – set to True if fwhm is in arcmin
- **return_ell** (*bool*) – if True, returns the corresponding ell array as well

Returns

- *Bl^2*
- *or*
- *ell, Bl^2*

astropaint.lib.utils.**get_custom_Nl**(*sigma_n, lmax, fwhm=None, frequency=[217], lmin=0, apply_beam=False, uK=False, return_ell=False*)

get temperature and polarization noise power spectra for a custom experiment

Parameters

- **[uK-arcmin]** (*sigma_n*) – noise level in uK-arcmin can a scalar or an array for multiple channels the length must match that of fwhm
- **lmax** (*scalar*) – maximum ell mode of the power spectrum
- **[arcmin]** (*fwhm*) – beam fwhm in arcmins can be scalar or an array for multiple channels
- **lmin** (*scalar*) – minimum ell mode of the power spectrum
- **apply_beam** – if True, deconvolves the noise with beam
- **return_ell** (*bool*) – if True, returns the corresponding ell array as well

Returns

- *Nl [K^2]*
- *or*
- *ell, Nl [K^2]*

astropaint.lib.utils.**get_experiment_Nl**(*lmax, lmin=0, name='Planck', frequency=[217], apply_beam=False, uK=False, return_ell=False*)

get temperature and polarization noise power spectra for various experiments

Parameters

- **lmax** (*scalar*) – maximum ell mode of the power spectrum
- **lmin** (*scalar*) – minimum ell mode of the power spectrum
- **name** – name of the experiment string in ["Planck" , "SO" , "S4"]
- **apply_beam** – if True, deconvolves the noise with beam
- **return_ell** (*bool*) – if True, returns the corresponding ell array as well

Returns

- *Nl_TT [K^2]*
- *or*
- *ell, Nl_TT [K^2]*

astropaint.lib.utils.**interpolate**(*profile*, *n_samples=20*, *min_frac=None*, *sampling_method='linspace'*, *k=3*, *interpolator=<class 'scipy.interpolate.fitpack2.InterpolatedUnivariateSpline'>*, **args*, ***kwargs*)

interpolate the profile function instead of calculating it at every given R

Parameters

- **profile** – wrapped profile to be interpolated (implicit)
- **n_samples** (*int*) – number of points to sample in R
- **min_frac** (*float*) – fraction of points in R to sample, unless n_sample is larger if min_frac: n_samples = max(n_samples, min_frac * len(R))

  e.g. for n_sample=10, min_frac=0.1 if len(R)=200, 20 (0.1*200) points will be sampled, but if len(R)=50 10 points will be sampled
- **sampling_method** (*str in ["linspace", "logspace", "random"]*) – determines how the points are sampled
- **k** (*int*) – interpolation order
- **interpolator** (*func*) – interpolating function

Returns

Return type Interpolated profile

astropaint.lib.utils.**load_Cl_Planck2018**(*lmin=0*)

load Cl from camb generated Dl file

Parameters

- **lmax** (*max ell number*) –
- **lmin** (*min ell number*) –

Returns

- *Cls*
- **available keys in Cls** (*L, TT, EE, BB, TE*)

astropaint.lib.utils.**load_noise_yaml**()

astropaint.lib.utils.**sample_array**(*array*, *n_samples*, *method='linspace'*, *eps=0.001*)

sample an input array

astropaint.lib.utils.**timeit**(*process_name='Process'*)

Time the code in mins

Module contents

astropaint.profiles package

Submodules

astropaint.profiles.Battaglia16 module

library containing [projected] gas profiles

E. Schaan: Tau profile, from Battaglia 2016 watch typos in paper. This code is correct.

astropaint.profiles.Battaglia16.**kSZ_T**(*R*, *R_200c*, *M_200c*, *v_r*, *redshift*, *, *T_cmb=2.7251*)
kinetic Sunyaev Zeldovich effect #TODO: add reference

astropaint.profiles.Battaglia16.**ne_2D**(*R*, *R_200c*, *M_200c*, *redshift*)

2D physical electron number density profile in 1/(Mpc/h)^3 assuming fully ionized gas and primordial He abundance

Parameters

- **R** (*arraylike, float*) – 2D projected distance from the center of the halo
- **R_200c** (*float [Mpc/h]*) – comoving radius of the halo
- **M_200c** (*float [Msun/h]*) – mass of the halo
- **redshift** – redshift of the halo

astropaint.profiles.Battaglia16.**ne_3D**(*r*, *R_200c*, *M_200c*, *redshift*)
3D physical electron number density profile in 1/(Mpc/h)^3 assuming fully ionized gas and primordial He abundance

Parameters

- **r** (*arraylike, float*) – 3D distance from the center of the halo
- **R_200c** (*float [Mpc/h]*) – comoving radius of the halo
- **M_200c** (*float [Msun/h]*) – mass of the halo
- **redshift** – redshift of the halo

astropaint.profiles.Battaglia16.**rho_gas_2D**(*R*, *R_200c*, *M_200c*, *redshift*)
2D physical gas density profile in (Msun/h) / (Mpc/h)^3

Parameters

- **R** (*arraylike, float*) – 2D projected distance from the center of the halo
- **R_200c** (*float [Mpc/h]*) – comoving radius of the halo
- **M_200c** (*float [Msun/h]*) – mass of the halo
- **redshift** – redshift of the halo

astropaint.profiles.Battaglia16.**rho_gas_2D_interp**(*R*, *R_200c*, *M_200c*, *redshift*)
Interpolated version of rho_gas_2D 2D physical gas density profile in (Msun/h) / (Mpc/h)^3

Parameters

- **R** (*arraylike, float*) – 2D projected distance from the center of the halo

- **R_200c** (*float [Mpc/h]*) – comoving radius of the halo
- **M_200c** (*float [Msun/h]*) – mass of the halo
- **redshift** – redshift of the halo

astropaint.profiles.Battaglia16.**rho_gas_3D**(*r*, *R_200c*, *M_200c*, *redshift*)

3D physical gas density profile in (Msun/h) / (Mpc/h)^3

Parameters

- **r** (*arraylike, float*) – 3D distance from the center of the halo
- **R_200c** (*float [Mpc/h]*) – comoving radius of the halo
- **M_200c** (*float [Msun/h]*) – mass of the halo
- **redshift** – redshift of the halo

astropaint.profiles.Battaglia16.**tau_2D**(*R*, *R_200c*, *M_200c*, *redshift*)

Thompson scattering optical depth 2D projected profile in 1/(Mpc/h) comoving ie you get the 2D tau profile by projecting this profile along the physical (not comoving) radial coordinate assuming fully ionized gas and primordial He abundance

Parameters

- **R** (*arraylike, float*) – 2D projected distance from the center of the halo
- **R_200c** (*float [Mpc/h]*) – comoving radius of the halo
- **M_200c** (*float [Msun/h]*) – mass of the halo
- **redshift** – redshift of the halo

astropaint.profiles.Battaglia16.**tau_2D_interp**(*R*, *R_200c*, *M_200c*, *redshift*)

Interpolated version of tau_2D: Thompson scattering optical depth 2D projected profile in 1/(Mpc/h) comoving ie you get the 2D tau profile by projecting this profile along the physical (not comoving) radial coordinate assuming fully ionized gas and primordial He abundance

Parameters

- **R** (*arraylike, float*) – 2D projected distance from the center of the halo
- **R_200c** (*float [Mpc/h]*) – comoving radius of the halo
- **M_200c** (*float [Msun/h]*) – mass of the halo
- **redshift** – redshift of the halo

astropaint.profiles.Battaglia16.**tau_3D**(*r*, *R_200c*, *M_200c*, *redshift*)

Thompson scattering optical depth 3D profile in 1/(Mpc/h) comoving ie you get the 2D tau profile by projecting this profile along the physical (not comoving) radial coordinate assuming fully ionized gas and primordial He abundance

Parameters

- **r** (*arraylike, float*) – 3D distance from the center of the halo
- **R_200c** (*float [Mpc/h]*) – comoving radius of the halo
- **M_200c** (*float [Msun/h]*) – mass of the halo
- **redshift** – redshift of the halo

astropaint.profiles.NFW module

library containing [projected] halo profiles

astropaint.profiles.NFW.**BG**(*R_vec*, *c_200c*, *R_200c*, *M_200c*, *theta*, *phi*, *v_th*, *v_ph*, ***, *T_cmb=2.7251*)
Birkinshaw-Gull effect aka moving lens aka Rees-Sciama (moving gravitational potential)

astropaint.profiles.NFW.**deflection_angle**(*R*, *c_200c*, *R_200c*, *M_200c*, ***, *suppress=True*, *suppression_R=8*)
calculate the deflection angle of a halo with NFW profile Using Eq 6 in Baxter et al 2015 (1412.7521)

Parameters

- **R** – distance from the center of halo [Mpc]
- **c_200c** – halo concentration parameter
- **R_200c** – halo 200c radius in [Mpc]
- **M_200c** – halo 200c mass of halo in M_sun

Returns

Return type the deflection angle at distance R from the center of halo

astropaint.profiles.NFW.**kSZ_T**(*R*, *rho_s*, *R_s*, *v_r*, ***, *T_cmb=2.7251*)
kinetic Sunyaev Zeldovich effect #TODO: add reference

astropaint.profiles.NFW.**rho_2D**(*R*, *rho_s*, *R_s*)
3D NFW profile intgrated along the line of sight

Returns surface mass density

Return type [M_sun/Mpc^2]

astropaint.profiles.NFW.**rho_2D_bartlemann**(*R*, *rho_s*, *R_s*)
projected NFW mass profile Eq. 7 in Bartlemann 1996: https://arxiv.org/abs/astro-ph/9602053

Returns surface mass density

Return type [M_sun/Mpc^2]

astropaint.profiles.NFW.**rho_2D_interp**(*R*, *rho_s*, *R_s*)
3D NFW profile intgrated along a sampled number of line of sights and then interpolated

Returns surface mass density

Return type [M_sun/Mpc^2]

astropaint.profiles.NFW.**rho_3D**(*r*, *rho_s*, *r_s*)
Calculate the 3D NFW density profile #TODO: add reference Eq.

Parameters

- **r** – distance from the center
- **rho_s** – density at radius r_s
- **r_s** – characterisic radius R_200c/c_200c

Returns

Return type rho = 4 * rho_s * r_s ** 3 / r / (r + r_s) ** 2

astropaint.profiles.NFW.**tau_2D**(*R*, *rho_s*, *R_s*)
projected NFW tau profile Eq. 7 in Battaglia 2016 :

Returns tau

Return type [NA]

astropaint.profiles.art_gallery module

library containing fun profiles!

astropaint.profiles.art_gallery.**bacteria**(*R*, *R_200c*, *M_200c*, *x*)
Are these microbes? use with cm.Greys_r colormap

astropaint.profiles.art_gallery.**drops**(*R*, *R_200c*, *x*, *y*, *z*)
profile to emulate water droplets... use with cm.Blues colormap

astropaint.profiles.art_gallery.**twilight**(*R*, *R_200c*, *x*, *y*, *z*)
nobody knows where this profile came from... use with cm.twilight profile

astropaint.profiles.spherical module

library containing simple [projected] spherical halo profiles

astropaint.profiles.spherical.**constant_density_2D**(*R*, *constant*)
return a constant value at every input r :param R: distance from the center :type R: [Mpc] :param constant: multiplicative constant

Returns

Return type constant

astropaint.profiles.spherical.**linear_density_2D**(*R*, *intercept*, *slope*)
return a R*constant at every input R :param R: distance from the center :type R: [Mpc] :param intercept: intercept of the line :param slope: slope of the line

Returns

Return type intercept

astropaint.profiles.spherical.**solid_sphere_2D**(*R*, *M_200c*, *R_200c*)
projected mass density of uniform sphere :param r: distance from the center :type r: [Mpc] :param M_200c: total mass of the sphere :type M_200c: [M_sun] :param R_200c: total radius (edge) of the sphere :type R_200c: [Mpc]

Returns

Return type Sigma = M_200c /2/pi * sqrt(R_tot**2 - r**2)/R_tot**3

Module contents

Submodules

astropaint.paint_bucket module

library for simulating semi-analytic mock maps of CMB secondary anisotropies

class astropaint.paint_bucket.**Canvas**(*catalog*, *nside*, *mode='healpy'*, *R_times=1*, *inclusive=False*)
Bases: object

healpy or flat-sky canvas with the location of the halos to paint the signal on

Cl

class Disc(*catalog*, *nside*, *R_times*, *inclusive*)

Bases: object

R_times

catalog

center_D_a

center_ang

center_index

center_vec

gen_cent2pix_hat(*halo_list='All'*)

gen_cent2pix_mpc(*halo_list='All'*)

gen_cent2pix_mpc_vec(*halo_list='All'*)

gen_cent2pix_rad(*halo_list='All'*)

gen_center_ang(*halo_list='All'*, *snap2pixel=False*)

generate the angle (theta, phi) of the halo centers if snap2pixel is True, the angular coordinate of the halo center pixel will be returned

gen_center_ipix(*halo_list='All'*)

generate ipix of the halo centers

gen_center_vec(*halo_list='All'*)

gen_pixel_ang(*halo_list='All'*)

gen_pixel_index(*halo_list='All'*)

gen_pixel_vec(*halo_list='All'*)

generate the unit vectors pointing to the disc pixels

Returns

Return type None

inclusive

nside

pix2cent_mpc

pix2cent_rad

pix2cent_vec

pixel_ang

pixel_index

Dl

R_max

add_cmb(*Cl='LCDM'*, *mode='TT'*, *lmax=None*, *inplace=True*, *weight=1*, *overwrite=False*, **args*, ***kwargs*)

Add CMB to the pixels If Cl array is not provided, a Lambda-CDM power spectrum is loaded from disc

Parameters

- **Cl** – Input CMB Power Spectrum The default keyword 'LCDM' loads a CAMB generated power spectrum from disc

- **mode** – 'TT' for Temperature Note: 'EE' and 'BB' for polarization are currently unavailable
- **lmax** – Maximum ell mode to include in the CMB power spectrum
- **inplace** – If True, the result will be added to canvas.pixels Otherwise the generated CMB map will be returned as output
- **weight** – The multiplicative factor for the generated CMB pixels
- **args** – *args to be passed to healpy.synfast(*args)
- **kwargs** – *kwargs to be passed to healpy.synfast(**kwargs)

Returns

- *None or np.ndarray*
- *The generated CMB map has the same NSIDE as the canvas.pixels map*

add_noise (*Nl='Planck', mode='TT', frequency=[217], lmax=None, sigma_n=None, fwhm_b=None, apply_beam=False, inplace=True, weight=1, *args, **kwargs*)

Add Noise to the pixels. The Nl can be either provided directly as an array, or as a keyword using the name of an experiment. The list of all available experiments can be found in astropaint/lib/noise.yml.

Parameters

- **Nl** – Input Noise Power Spectrum. - If Nl is an array, it will be used as the noise power spectrum Nl. - If Nl is a string (e.g. 'Planck', 'SO', or 'S4') the noise configuration for the selected frequency channel will be read from the noise.yml file. Arbitrary noise configurations can be added to this file and then called here (see lib.misc.get_experiment_Nl for details). - If Nl is set to 'white', then sigma_n and fwhm_b will be used to construct a white noise power spectrum on the fly (see lib.misc.get_custom_Nl for details).
- **mode** – 'TT' for Temperature Note: 'EE' and 'BB' for polarization are currently unavailable
- **[GHz]** (*frequency*) – Frequency channels at which the noise will be calculated. If a list is provided, the final noise will be the combination of all channels.
- **lmax** – Maximum ell mode to include in the noise power spectrum
- **sigma_n** – For Nl='white', this will be used as the noise level sigma [uK-arcmin], i.e.

  w_inverse = arcmin2rad(sigma_n) ** 2 Nl = w_inverse (for each ell)
- **fwhm_b** – if apply_beam=True, this will be used as the beam Full-Width-Half-Maximum (FWHM), i.e.

  fwhm = arcmin2rad(np.asarray(fwhm)) sigma_theta = fwhm ** 2 / 8 / np.log(2) B2l = np.exp(-L * (L + 1) * sigma_theta
- **apply_beam** – If True, the Noise power spectrum will be divided by the Beam spectrum B2l
- **inplace** – If True, the result will be added to canvas.pixels Otherwise the generated noise map will be returned as output
- **weight** – The multiplicative factor for the generated CMB pixels
- **args** – *args to be passed to healpy.synfast(*args)
- **kwargs** – *kwargs to be passed to healpy.synfast(**kwargs)

Returns

- *None or np.ndarray*

- *The generated Noise map has the same NSIDE as the canvas.pixels map*

alm

almxfl (*fl*, *inplace=True*)
wrapper for healpy.almxfl multiplies the alms by the array fl (lmin=0, to lmax=3*nside+1)

beam_smooth (*fwhm_b=0.0*, *sigma_b=None*, **args*, ***kwargs*)
Smoothes canvas.pixels with a gaussian beam using healpy.sphtfunc.smoothing

Parameters

- **fwhm_b** –
- **sigma_b** –
- **args** –
- **kwargs** –

Returns

Return type None

catalog

clean ()
Clean the canvas and set all pixels to zero

Returns

Return type None

cmap

cutouts (*halo_list='all'*, *lon_range=[-1, 1]*, *lat_range=None*, *xpix=200*, *ypix=None*, *apply_func=None*, *func_kwargs={}*)
Generate cutouts of angular size lon_range x lat_range around halo center with xpix & ypix pixels on each side.

*This method uses Healpy's projector.CartesianProj class to perform the cartesian projection.

Parameters

- **halo_list** – index of halos to consider (e.g. [1,2,5,10]). goes through all the halos in the catalog when set to "all".
- **lon_range** – range of longitutes to cut around the halo center in degrees. e.g. [-1,1] cuts out 1 degree on each side of the halo. same as lon_range in healpy
- **lat_range** – range of longitutes to cut around the halo center in degrees. by default (None) it is set equal to lon_range. same as lat_range in healpy
- **xpix** – number of pixels on the x axis same as xpix in healpy
- **ypix** – number of pixels on the y axis by default (None) it is set equal to xrange same as ypix in healpy
- **apply_func** – function to apply to the patch after cutting it out. THe first argument of the function must be the input patch.

func_kwargs: dictionary or list of dictionaries dictionary of keyword arguments to pass to apply_func

Example usage:

cutouts(apply_func=linear_transform, slope=2, intercept=1)

if the func_kwargs are scalars, they will be the same for all the halos in apply_func. If they are arrays or lists, their lengths must be the same as the catalog size.

Returns

Return type generator

ell

get_Cl(*save_alm=True*, *lmax=None*)
find the power spectrum of the map (.pixels)

get_alm_from_pixels()
find the alm coefficients of the map

get_pixels_from_alm()
get the map from the alm coefficients

instantiate_discs()
instantiate the discs attribute using the Disc class Useful when the disc generators are exhausted and need to be reset

lmax

load_map_from_file(*filename=None*, *prefix=None*, *suffix=None*, *inplace=True*)
save the healpy map to file

Parameters

- **filename** (*str*) – custom file name; overrides the prefix and suffix and default file name
- **prefix** (*str*) – prefix string to be added to the beginning of the default file name
- **suffix** (*str*) – suffix string to be added to the end of default file name
- **inplace** (*bool*) – if True, canvas.pixels will be loaded with the map from file

mask_alm(*lmin=0*, *lmax=None*, *inplace=True*)
only keep alms in the range between lmin and lmax (inclusive)

npix

nside

pixels

remove_cmb()
remove cmb from the pixels

remove_noise()
Remove canvas.noise from the pixels

save_Cl_to_file(*prefix=None*, *suffix=None*, *filename=None*)
save the map power spectrum to file

Parameters

- **prefix** (*str*) – prefix string to be added to the beginning of the default file name
- **suffix** (*str*) – suffix string to be added to the end of default file name
- **filename** (*str*) – custom file name; overrides the prefix and suffix and default file name

save_map_to_file(*filename=None*, *prefix=None*, *suffix=None*, *overwrite=True*)
save the healpy map to file

Parameters

- **filename** (*str*) – custom file name; overrides the prefix and suffix and default file name
- **prefix** (*str*) – prefix string to be added to the beginning of the default file name
- **suffix** (*str*) – suffix string to be added to the end of default file name

show_discs(*projection='mollweide'*, *graticule=False*, **args*, ***kwargs*)

show_halo_centers(*projection='mollweide'*, *graticule=True*, *marker='o'*, *color=None*, *s=None*, **args*, ***kwargs*)

show_map(*projection='mollweide'*, *apply_func=None*, **args*, ***kwargs*)

stack_cutouts(*halo_list='all'*, *lon_range=[-1, 1]*, *lat_range=None*, *xpix=200*, *ypix=None*, *inplace=True*, *parallel=False*, *apply_func=None*, *func_kwargs={}*)

Stack cutouts of angular size lon_range x lat_range around halo center with xpix & ypix pixels on each side. apply_func is applied to each cutout before stacking

*This method uses Healpy's projector.CartesianProj class to perform the cartesian projection.

Parameters

- **halo_list** – index of halos to consider (e.g. [1,2,5,10]). goes through all the halos in the catalog when set to "all".
- **lon_range** – range of longitutes to cut around the halo center in degrees. e.g. [-1,1] cuts out 1 degree on each side of the halo. same as lon_range in healpy
- **lat_range** – range of longitutes to cut around the halo center in degrees. by default (None) it is set equal to lon_range. same as lat_range in healpy
- **xpix** – number of pixels on the x axis same as xpix in healpy
- **ypix** – number of pixels on the y axis by default (None) it is set equal to xrange same as ypix in healpy
- **inplace** – if True, the result is saved in canvas.stack. Otherwise the stack is returned as outpu.
- **with_ray** – if True, stacking is be performed in parallel using ray.
- **apply_func** – function to apply to the patch after cutting it out. THe first argument of the function must be the input patch.

  Example:

  def linear_transform(patch, slope, intercept): return slope * patch + intercept
- **func_kwargs** – keyword arguments to pass to apply_func

  Example usage:

  stack_cutouts(halo_list=np.arange(10), apply_func=linear_transform, inplace=True, slope=2, intercept=1)

  This will apply the function linear_transform to the first 10 halos and stacks them together in canvas.stack.

  if the func_kwargs are scalars, they will be the same for all the halos in apply_func. If they are arrays or lists, their lengths must be the same as the catalog size.

Returns

- *np.ndarray*
- *or None (if inplace=True)*

ud_grade(*nside_out*, *inplace=True*, ***kwargs*)
"wrapper for healpy.ud_grade() function changes the nside of canvas.pixels to nside_out

Parameters

- **nside_out** (*int*) – nside of the new map
- **inplace** (*bool*) – if True, change canvas.pixels, otherwise return the new map
- **kwargs** (*kwargs or dict*) – kwargs to be passeed to hp.ud_grade()

class astropaint.paint_bucket.**Catalog**(*data=None*, *calculate_redshifts=False*, *default_redshift=0*)
Bases: object

halo catalog containing halo masses, locations, velocities, and redshifts

x, y, z: [Mpc] v_x, v_y, v_z: [km/s] M_200c: [M_sun]

build_dataframe(*calculate_redshifts=False*, *default_redshift=0*)

cut_D_a(*D_min=0.0*, *D_max=inf*, *inplace=True*)
Cut the catalog according the the given angular diameter distance range

Parameters

- **[Mpc]** (*D_max*) – minimum halo angular diameter distance to keep
- **[Mpc]** – maximum halo angular diameter distance to keep

Returns

- *None*
- *catalog.data will only contain halos with angular diameter distance D_a in the range*
- *D_min < D_a < D_max*

cut_D_c(*D_min=0.0*, *D_max=inf*, *inplace=True*)
Cut the catalog according the the given comoving distance range

Parameters

- **[Mpc]** (*D_max*) – minimum halo comoving distance to keep
- **[Mpc]** – maximum halo comoving distance to keep

Returns

- *None*
- *catalog.data will only contain halos with comoving distance D_c in the range D_min < D_c <*
- *D_max*

cut_M_200c(*mass_min=0.0*, *mass_max=inf*, *inplace=True*)
Cut the catalog according the the given mass range

Parameters

- **[M_sun]** (*mass_max*) – minimum halo mass to keep
- **[M_sun]** – maximum halo mass to keep

Returns

- *None*
- *catalog.data will only contain halos with mass M in the range mass_min < M < mass_max*

cut_R_200c(*R_min=0.0*, *R_max=inf*, *inplace=True*)
Cut the catalog according the the given radius range

Parameters

- **[Mpc]** (*R_max*) – minimum halo radius to keep
- **[Mpc]** – maximum halo radius to keep

Returns

- *None*
- *catalog.data will only contain halos with radius R in the range R_min < R < R_max*

cut_R_ang_200c(*R_ang_min=0.0*, *R_ang_max=inf*, *inplace=True*)
Cut the catalog according the the given angular radius range

Parameters

- **[arcmin]** (*R_ang_max*) – minimum halo angular radius to keep
- **[arcmin]** – maximum halo angular radius to keep

Returns

- *None*
- *catalog.data will only contain halos with angular radius R_ang_200c in the range*
- *R_ang_min < R_ang_200c < R_ang_max*

cut_lon_lat(*lon_range=[0, 360], lat_range=[-90, 90], inplace=True*)
Cut the catalog according the the given longitude and latitude range

Parameters

- **[deg]** (*lat_range*) – range of longitutes to keep
- **[deg]** – rane of latitudes to keep

Returns

- *None*
- *catalog.data will only contain halos with longitutes in the range lon_range and*
- *latitudes in the range lat_range*

cut_mask(*mask*, *threshold=0.5*, *inplace=True*)
cut the catalog according to the input mask halos outside of the mask (mask<threshold) will be discarded

cut_redshift(*redshift_min=0.0*, *redshift_max=inf*, *inplace=True*)
Cut the catalog according the the given redshift range

Parameters

- **redshift_min** – minimum halo redshift to keep
- **redshift_max** – maximum halo redshift to keep

Returns

- *None*
- *catalog.data will only contain halos with redshift in the range*
- *redshift_min < redshift < redshift_max*

cut_theta_phi(*theta_range=[0, 3.141592653589793], phi_range=[0, 6.283185307179586], inplace=True*)
Cut the catalog according the the given longitude and latitude range

Parameters

- **[rad]** (*phi_range*) – range of longitutes to keep
- **[rad]** – rane of latitudes to keep

Returns

- *None*
- *catalog.data will only contain halos with theta in the range theta_range and*
- *phi in the range phi_range*

data

generate_random_box(*box_size=50, v_max=100, mass_min=100000000000000.0, mass_max=1000000000000000.0, n_tot=50000, put_on_shell=False, inplace=True*)

generate_random_shell(*shell_radius=50, v_max=100, mass_min=100000000000000.0, mass_max=1000000000000000.0, n_tot=50000, inplace=True*)

generate_test_box(*configuration=['all'], distance=100, mass=1000000000000000.0, inplace=True*)

load_from_csv(*sample_name='MICE'*)
load sample data using the name of dataset

move_to_box_center()
move the observer from (0,0,0) to the center of the box (Lx/2, Ly/2, Lz/2) to make coordinates symmetric
*Not recommended for light-cone catalogs

replicate(*mode='rotate'*)
Replicate an octant to get a whole-sky box

Parameters **mode** –

save_to_csv(*sample_name*)
load sample data using the name of dataset

class astropaint.paint_bucket.**Painter**(*template*)
Bases: object

Painter object sprays a signal over the canvas using a template

calculate_template(*R, catalog, halo_list=[0], **template_kwargs*)
calculate the 1D profile of the template as a function of R [Mpc]

plot_template(*R, catalog, halo_list=[0], axis=None, **template_kwargs*)
plot the 1D profile of the template as a function of R [Mpc]

spray(*canvas, distance_units='Mpc', n_cpus=-1, cache=False, lazy=False, lazy_grid=None, **template_kwargs*)
#TODO: add example

Parameters

- **canvas** –
- **distance_units** –
- **with_ray** –

- **cache** –
- **lazy** –
- **template_kwargs** –

template

Module contents

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

N

P

R

S

T

U