VELOCIraptor Python Tools

Release 0.16.1

Apr 12, 2023

Contents

VELOCIraptor-Python is a python toolkit for reading VELOCIraptor outputs, with a particular focus on integration with the SWIFT cosmological simulation code.

It allows for automatic association of symbolic units (using unyt) with halo properties, extraction of individual haloes to a SWIFTsimIO dataset, and includes tools to help create both publication-ready and quick figures.

CHAPTER 1

Installation

1.1 Requirements

The VELOCIraptor library requires:

- `unyt` and its dependencies
- `h5py` and its dependencies
- `python3.6` or above
- Note that for development, we suggest that you have `pytest` and `black` installed. To create plots, you will need the plotting framework `matplotlib`.

1.2 Installation

The package can be installed from PyPI, and is named *velociraptor*,

```
pip3 install velociraptor
```

Alternatively you can install the library from the GitHub repository by cloning it and running:

```
pip3 install -e .
```

CHAPTER 2

Basic Usage

At its most fundamental level, this library was designed to make working with the VELOCIraptor catalogues easier. It does this by abstracting away the HDF5 file into a python object, where each array has units associated with it.

2.1 Registered Quantities

Within the library, there are sets of registration functions that turn the velociraptor data into python data with units, associated with an object. Each of these registration functions acts on different classes of properties. We describe the available registration functions (these are not entirely complete!) below:

- `metallicity`: properties that start with `Zmet`
- `ids`: properties that are to do with IDs, such as Halo IDs or the most bound particle ID.
- `energies`: properties starting with `E`
- `stellar_age`: the `t_age` property
- `rotational_support`: the `kappa` properties that describe rotational support
- `star_formation_rate`: properties starting with `SFR`
- `masses`: properties starting with `M` or `Mass`, e.g. `M_200crit`
- `eigenvectors`: shape properties
- `radii`: properties starting with `R`, that are various characteristic radii
- `temperature`: properties starting with `T` such as the temperature of the halo
- `veldisp`: velocity dispersion quantities
- `structure_type`: the structure type properties
- `velocities`: velocity properties
- `positions`: various position properties, such as `Xc`
- `concentration`: concentration of the halo, contains `cNFW`

- `rvmax_quantities`: properties measured inside `RVmax`
- `angular_momentum`: various angular momentum quantities starting with `L`
- `projected_apertures`: several projected apertures and the quantities associated with them
- `apertures`: properties measured within apertures
- `element_mass_fractions`: element mass fractions within the halo
- `fail_all`: a registration function that fails all tests, development only.

To extract properties, you need to instantiate a `velociraptor.VelociraptorCatalogue`. You can do this by:

```
from velociraptor import load

data = load("/path/to/catalogue.properties")

masses_200crit = data.masses.m_200crit
masses_200crit.convert_to_units("kg")
```

Here, we have the values of `M_200crit` stored in kgs, correctly applied based on the unit metadata in the file.

There is also full unit information available in the `data.units` object, with an `astropy` cosmology object provided as `data.units.cosmology`.

Subsets of the catalogue can be read by providing a mask, for instance to read only values for the 3rd object in the catalogue (indexed from 0):

2.2 Creating your first plot

If, for example, we wish to create a mass function of these values, we can use the tools,

```
from velociraptor.tools import create_mass_function
from velociraptor.labels import get_full_label, get_mass_function_label
from unyt import Mpc

# Convert to stellar masses because that 'makes sense'
masses_200crit.convert_to_units("msun")

box_volume = (25 * Mpc)**3

# Set the edges of our halo masses,
lowest_halo_mass = 1e9 * unyt.msun
highest_halo_mass = 1e14 * unyt.msun

bin_centers, mass_function, error = tools.create_mass_function(
   halo_masses, lowest_halo_mass, highest_halo_mass, box_volume
)
```

We now have a halo mass function, but the fun doesn't end there - we can get pretty labels *automatically* out of the python tools:

```
mass_label = get_full_label(masses_200crit)
mf_label = get_mass_function_label("200crit", mass_function)
```

If you want to try this out yourself, you can use the example scripts available in the repository. Currently, we have scripts that create a HMF, SMF, and a galaxy-size stellar-mass plot.

CHAPTER 3

Extracting particles

Using the `velociraptor.particles.VelociraptorParticles` class, it is possible to find which particles belong to a given halo.

3.1 Extracting a Halo

To extract the information from the `.catalog_groups` files in VELOCIraptor you will need to use an instance of *velociraptor.particles.particles.VelociraptorGroups*, easily generated from `velociraptor.particles.particles.load_groups()`. Pass this the path to the `.catalog_groups` file, and associate the catalogue with it. This will allow you to extract haloes using the *velociraptor.particles.particles.VelociraptorGroups.extract_halo()* method, as follows:

```
from velociraptor.particles import load_groups
from velociraptor import load

catalogue = load("{path_to_file}.properties")
groups = load_groups("{path_to_groups}.catalog_groups", catalogue=catalogue)

particles, unbound_particles = groups.extract_halo(halo_index=100)
```

Note that the `halo_index` is the position of the halo in the group catalogue, not the halo's unique id.

This returns two objects, `particles`, and `unbound_particles`, corresponding to both the bound and unbound component of your halo respectively. Each of these contains the information required to extract just those particles from a snapshot (this is made much easier by using the SWIFT integration, shown below). Note that use of a masked `catalogue` is not supported.

The instances of *velociraptor.particles.particles.VelociraptorParticles* have several useful attributes,

- `mass_{200crit,200mean,bn98,fof}` - masses in various apertures and groups
- `r_{200crit,200mean,bn98,size}` - radii in various apertures and groups
- `x, y, z` - center of mass co-ordinates for the halo

- **If available:**
  - `x_gas, y_gas, z_gas` - gas CoM co-ordinates
  - `x_star, y_star, z_star` - star CoM co-ordinates
  - `x_mbp, y_mbp, z_mbp` - position of the most bound particle.

3.2 SWIFTsimIO Integration

We also provide functionality to quickly (by using spatial metadata in the snapshots) extract the regions around haloes, and the specific particles in each halo itself. To do this, you will need to use the tools in *velociraptor.swift*, in particular the *velociraptor.swift.swift.to_swiftsimio_dataset()* function. It is used as follows:

```
data, mask = to_swiftsimio_dataset(
    particles,
    "/path/to/snapshot.hdf5",
    generate_extra_mask=True
)

# The dataset that is returned is only spatially masked. It only contains
# particles that are within the same top-level cell as the region that the
# halo overlaps with, but it can be accessed as if it is just a regular
# `swiftsimio` dataset. For instance
gas_densities = data.gas.densities
redshift = data.metadata.z
hydro_info = data.metadata.hydro_info

# The extra mask allows for you to find only the particles that are classed
# as being part of the FoF group (in this case only the bound particles).
# To select the gas densities of particles in the group, for example,
# perform the following:
gas_densities_only_fof = data.gas.densities[mask.gas]
# Or the dark matter co-ordinates
dm_coordinates_only_fof = data.dark_matter.coordinates[mask.dark_matter]

# All of the swiftsimio features are available, so for instance you can
# generate a py-sphviewer instance out of these
from swiftsimio.visualisation.sphviewer import SPHViewerWrapper
sphviewer = SPHViewerWrapper(data.gas)
sphviewer.quickview(xsize=1024,ysize=1024,r="infinity")
```

The spatial mask and extra mask can be obtained individually by using the functions *velociraptor.swift.swift.generate_spatial_mask()* and *velociraptor.swift.swift.generate_bound_mask()*, respectively.

To see these functions in action, you can check out the examples available in `examples/swift_integration*.py` in the repository.

CHAPTER 4

Plotting Tools

As well as the ability to extract data out of catalogues in a more efficient way, the VELOCIraptor python toolkit also contains useful tools for creating plots in a programatic way. These tools are available in the *velociraptor.tools* module.

4.1 Labels

You can automatically generate labels for a given dataset using the *velociraptor.tools.labels.get_full_label()* function, as follows:

```
from velociraptor import load
from velociraptor.tools.labels import get_full_label

data = load("...properties")
m = data.masses.mass_200crit
label = get_full_label(m)
```

This `label` is a LaTeX string containing a description of the dataset and the symbolic units of whatever you converted it to.

You can also generate labels for mass functions:

```
from velociraptor.tools.labels import get_mass_function_label
import unyt

label = get_mass_function_label("*", unyt.Mpc**(-3))
```

which produces `d$n(M_*)$/d$log_{10}M_*$ [Mpc$^{-3}$]`.

4.2 Lines

One frequent activity is to generate binned lines for a figure, for instance in a plot of x against y you may wish to have the mean or median of y in equally spaced bins of x.

We provide two functions, *velociraptor.tools.lines.binned_mean_line()* and *velociraptor.tools.lines.binned_median_line()*, to perform this task for mean and median bins respectively. They both return scatter, with the mean line returning the standard deviation and the median line by default returning the 16-86 percentile scatter.

```
from velociraptor import load
from velociraptor.tools.lines import binned_median_line
from numpy import logspace
from unyt import unyt_array, Solar_Mass

data = load("...properties")

bins = unyt_array(np.logspace(10, 15, 25), units=Solar_Mass)

centers, median, deviation = binned_mean_line(
   x=data.apertures.mass_30_kpc.to(Solar_Mass),
   y=data.apertures.rhalfmass_30_kpc,
   x_bins=bins
)
```

4.3 Mass Functions

Generating mass functions can be tricky, especially if you wish to plot them on 'true' axes (i.e. plotting mass on the x axis, not log(mass)). We provide functionality to create the mass function through *velociraptor.tools.mass_functions.create_mass_function()*. This returns the bin centers, mass function, and poisson scatter in that bin. If you wish to use your own mass bins, you can use the alternative *velociraptor.tools.mass_functions.create_mass_function_given_bins()*.

```
from velociraptor import load
from velociraptor.tools.mass_functions import create_mass_function
from unyt import Solar_Mass

data = load("...properties")
masses = data.masses.mass_200mean
box_volume = data.units.comoving_box_volume

centers, mf, scatter = create_mass_function(
   masses=masses,
   lowest_mass=1e8*Solar_Mass,
   highest_mass=1e15*Solar_Mass,
   box_volume=box_volume,
   n_bins=25
)
```

We also provide an adaptive mass function plotting code; this allows for variable bin widths and is useful in the case where you have very few data points. It is called through *velociraptor.tools.mass_functions.create_adaptive_mass_function()* and has the same parameters as `create_mass_function` except that `n_bins` is now `base_n_bins`. The algorithm works as follows:

1. Attempt to use the standard binning scheme based on fixed-width bins, by trawling the data from left (low M_*) to right (high M_*).
2. If you have passed more than n items by the time you get to the next bin (standard is 0.2 dex in M_*)$, create the bin as normal and plot the point as the median mass value.
3. If not, continue until you have at least n items in the bin. Once you do, call this a new 'bin' with the right edge of the bin being the value you just found. Place the point at the median value of M_* in this bin.
4. The highest mass value within a given bin becomes the right edge of that bin and hence the left edge of the next bin.
5. Once you have reached the end of the data, attempt to make one final bin with the leftovers. If there is only one item in the final bin, we extend the previous bin to include it.

By default $n = 3$.

CHAPTER 5

Observational Data

The functionality in this library is generally for dealing with data that comes out of VELOCIraptor, and as such can only interface with a limited number of simulation datasets. Frequently, it is useful to over-plot data from, for example, other simulation groups, or even (_gasp_) observational data. Here, "observational data" actually refers to comparison data of any kind.

Dealing with this observational data is usually a mess of scattered `.csv` files. Here, we describe an interface that allows for structured metadata to be created, and for easily usable files containing this observational data to be saved to disk and recovered for future use. The overarching abstraction here is a python object (of type *velociraptor.observations.objects.ObservationalData*) that can be written to or read from file with HDF5 as the backing format.

This data can also be used with the *velociraptor.autoplotter* functionality.

5.1 File Format

The file format used for the `velociraptor.observational_data` functionality is described below. These files are simple HDF5 files that can be opened using `h5py` or some other module, and so are easily portable to different systems. They are designed to be both human and machine 'readable' to enable maximal re-use throughout many projects.

You are free to write your own reader/writer for this data, but the other sections in this documentation showcase python tools already written to perform such tasks.

5.1.1 HDF5 Fields

The required fields in a file are given here:

```
file_name.hdf5
├cosmology
│  └9 attributes:
│    ├H0: H0 in km/s/Mpc
```

(continues on next page)

(continued from previous page)

```
|   ├Neff: Effective number of neutrino species
|   ├Ob0: Omega baryons: density of baryonic matter in units of the critical density␣
↪at z=0.
|   ├Ode0: Omega dark energy: density of dark energy in units of the critical density␣
↪at z=0.
|   ├Om0: Omega matter: density of matter in units of the critical density at z=0.
|   ├Tcmb0: CMB temperature at z=0
|   ├m_nu: Masses of neutrino species (size 3)
|   ├m_nu_units: Neutrino mass units
|   └name: Name of this cosmology (e.g. WMAP7)
├metadata
|  └6 attributes:
|   ├bibcode: 'Bibcode - available from e.g. ADS'
|   ├citation: 'Short citation (e.g. X et al. 2019)'
|   ├comment: 'Free-text comment describing e.g. IMF choices'
|   ├name: 'Short name that is easily recognised'
|   ├plot_as: 'points' or 'line' - plot as either markers or a line
|   └redshift: 0.0 - floating point number
├x
|  ├2 attributes:
|  |  ├comoving: True/False
|  |  └description: 'Short description of axis, e.g. galaxy stellar mass'
|  └values    [float64: N]
|     └2 attributes:
|        ├unit_registry: Generated by unyt automatically
|        └units: 'String units, e.g. Msun'
└y
   ├2 attributes:
   |  ├comoving: True
   |  └description: 'Short description of axis, e.g. galaxy stellar mass'
   ├scatter  [float64: N (or 2XN), OPTIONAL] - scatter in this dimension.
   |  └2 attributes:
   |     ├unit_registry: Generated by unyt automatically
   |     └units: 'String units, e.g. Msun'
   └values   [float64: N]
      └2 attributes:
         ├unit_registry: Generated by unyt automatically
         └units: 'String units, e.g. Msun'
```

5.2 Using "Observational Data" Files

The observational data files created using *velociraptor* can be opened into python objects through the use of the API available in *velociraptor.observations*.

To load a file, you should use the *velociraptor.observations.load_observations()*, which will return an list of objects of type *velociraptor.observations.objects.ObservationalData*. Each instance has several useful properties, and one key method, *velociraptor.observations.objects.ObservationalData.plot_on_axes()*. This method allows you to provide a `matplotlib.pyplot.Axes` object to automatically plot the `x` and `y` fields on those axes. Those arrays, `x` and `y`, are instances of the `unyt.unyt_array` class, and as such you can use the `matplotlib_support` environment within `unyt` to automatically convert units to be consistent on the axes.

5.2.1 Example

Below we create a plot of only the information out of a given ObservationalData instance (and file) after loading it.

```
from velociraptor.observations import load_observations
import matplotlib.pyplot as plt

obs = load_observations("path/to/file.hdf5")[0]

fig, ax = plt.subplots()
ax.loglog()

obs.x.convert_to_units("kpc")
obs.y.convert_to_units("msun")

obs.plot_on_axes(ax)

fig.savefig("out.png")
```

You can provide multiple filenames, and possibly multi-redshift datasets, along with a redshift bracket to only return datasets that overlap with the given redshift range:

```
from velociraptor.observations import load_observation
import matplotlib.pyplot as plt
import unyt

observations = load_observations(
   ["path/to/file_1.hdf5", "path/to/file_2.hdf5", "path/to/file_3.hdf5"],
   [0.0, 0.5] # Plot observations from z=0.0 to z=0.5
)[0]

fig, ax = plt.subplots()
ax.loglog()

with unyt.matplotlib_support:
   for obs in observations:
      obs.plot_on_axes(ax)

fig.savefig("out.png")
```

5.3 Creating Data Files

There is functionality built into *velociraptor* to enable easy creation of data files. To do this, you will need to fill an *velociraptor.observations.objects.ObservationalData* instance, and then call the *velociraptor.observations.objects.ObservationalData.write()* method to save it out to file. There are several association functions that you will need to call to register various metadata. An example converting a TSV file to a *velociraptor*-compatible file is shown below.

5.3.1 General Rules

To ensure consistency between data files, follow the following suggestions:

- Never use 'log' data on either axis; e.g. always include mass, not log(mass) even if the original paper used logarithmic quantities.

- Always include all requested metadata (it doesn't take that long, but is very useful)!
- Always remove all h-factors. Every file should be h-free.
- Include a comment describing important aspects for the data, e.g. for a stellar mass function include the assumed IMF.

5.3.2 Example

The input TSV file:

```
#Crain et al 2009 (GIMIC)
#Assuming Chabrier IMF and cosmology of
# Omega_l = 0.75, Omega_0 = 0.045, h = 0.73
#
#GSMF weighted over all density regions
#log(M) [Msun] Phi [(h^-1 Mpc)^-3]
7.513259     -0.266989
7.765294     -0.297700
8.028524     -0.493567
8.280272     -0.704415
8.531949     -0.960297
8.783625     -1.216179
9.035397     -1.412015
9.298460     -1.712962
9.527172     -1.998804
9.767462     -2.209621
10.019115    -2.480514
10.271149    -2.511225
10.523422    -2.391822
10.775695    -2.272418
11.004742    -2.348101
11.267685    -2.724105
11.508166    -2.814830
11.771062    -3.220858
12.011042    -3.626822
12.251666    -3.627479
12.490953    -4.468774
```

Conversion file:

```
from velociraptor.observations.objects import ObservationalData
from astropy.cosmology import WMAP7 as cosmology
import unyt
import numpy as np
import os

input_filename = "Crain2009_GSMF.txt"
delimiter = "\t"

output_filename = "crain_2009.hdf5"
output_directory = "gsmf"

if not os.path.exists(output_directory):
   os.mkdir(output_directory)

processed = ObservationalData()
```

(continues on next page)

(continued from previous page)

```
raw = np.loadtxt(input_filename, delimiter=delimiter)

comment = f"Assuming Chabrier IMF. h-corrected for SWIFT using cosmology: {cosmology.
↪name}."
citation = "Crain et al. 2009 (GIMIC)"
bibcode = "2009MNRAS.399.1773C"
name = "GSMF from GIMIC"
plot_as = "line"
redshift = 0.0
redshift_lower = 0.0
redshift_upper = 0.2
h = cosmology.h

log_M = raw.T[0]
M = 10 ** (log_M) * unyt.Solar_Mass / h
Phi = (10**raw.T[1] * (h ** 3)) * unyt.Mpc ** (-3)

processed.associate_x(M, scatter=None, comoving=True, description="Galaxy Stellar Mass
↪")
processed.associate_y(Phi, scatter=None, comoving=True, description="Phi (GSMF)")
processed.associate_citation(citation, bibcode)
processed.associate_name(name)
processed.associate_comment(comment)
processed.associate_redshift(redshift, redshift_lower, redshift_upper)
processed.associate_plot_as(plot_as)
processed.associate_cosmology(cosmology)

output_path = f"{output_directory}/{output_filename}"

if os.path.exists(output_path):
   os.remove(output_path)

processed.write(filename=output_path)
```

5.3.3 Multi-Redshift Data

Data from a single paper that has been collected at multiple redshifts (or a single simulation, with multiple snapshots) should be stored in a multi-redshift file. This will allow the most appropriate redshift from the data to be plotted automatically when using the pipeline.

The `velociraptor.observations.MultiRedshiftObservationalData` class acts as a container for multiple instances of the `velociraptor.observations.ObservationalData` object, each for a single redshift. However, the comments and cosmology are stored at the top level. Extending the example above to handle the multiple redshift case:

```
from velociraptor.observations.objects import (
   ObservationalData,
   MultiRedshiftObservationalData,
)
from astropy.cosmology import WMAP7 as cosmology
import unyt
import numpy as np
import os

input_filenames = ["Crain2009_GSMF_z0.txt", "Crain2009_GSMF_z1.txt"]
```

(continues on next page)

(continued from previous page)

```
input_redshifts = [[0.0, 0.5], [0.5, 1.5]]
delimiter = "\t"

output_filename = "Crain_2009.hdf5"
output_directory = "gsmf"
comment = f"Assuming Chabrier IMF. h-corrected for SWIFT using cosmology: {cosmology.
↪name}."
citation = "Crain et al. 2009 (GIMIC)"
bibcode = "2009MNRAS.399.1773C"
name = "GSMF from GIMIC"

if not os.path.exists(output_directory):
   os.mkdir(output_directory)

multi_z = MultiRedshiftObservationalData()
multi_z.associate_citation(citation, bibcode)
multi_z.associate_name(name)
multi_z.associate_comment(comment)
multi_z.associate_cosmology(cosmology)
multi_z.associate_maximum_number_of_returns(1)

for filename, redshifts in zip(input_filenames, input_redshifts):
   processed = ObservationalData()
   raw = np.loadtxt(filename, delimiter=delimiter)

   plot_as = "line"
   redshift = 0.5 * sum(redshifts)
   redshift_lower, redshift_upper = redshifts
   h = cosmology.h

   log_M = raw.T[0]
   M = 10 ** (log_M) * unyt.Solar_Mass / h
   Phi = (10**raw.T[1] * (h ** 3)) * unyt.Mpc ** (-3)

   processed.associate_x(M, scatter=None, comoving=True, description="Galaxy Stellar␣
↪Mass")
   processed.associate_y(Phi, scatter=None, comoving=True, description="Phi (GSMF)")
   processed.associate_redshift(redshift, redshift_lower, redshift_upper)
   processed.associate_plot_as(plot_as)

   multi_z.associate_dataset(processed)

output_path = f"{output_directory}/{output_filename}"

if os.path.exists(output_path):
   os.remove(output_path)

multi_z.write(filename=output_path)
```

In this example, note that the following items are stored at the top level:

- Citation
- Name
- Comment
- Cosmology

as the object is an abstraction for a single piece of academic work. Below this, at the individual dataset level, we have

- Actual data (e.g. x, y, associated with a single redshift)
- Redshift (with bracketing)
- Plotting commands (as some redshifts may have a very small number of objects, hence being better plotted as points, whereas some redshifts may require binning to a line).

Finally, we have the new `associate_maximum_number_of_returns` function. This determines the maximum number of returned datasets from the `load_datasets` function. This is useful in cases where you have a large number of individual datasets that cover very small ranges in redshift, and you may only wish to plot one of them at a time on a given figure.

CHAPTER 6

AutoPlotter

The `velociraptor.auto_plotter` sub-module can be used to automatically generate large amounts of figures from VELOCIraptor catalogues without writing any python code.

CHAPTER 7

API Documentation

7.1 velociraptor package

The velociraptor module.

More information is available in the documentation.

velociraptor.**load**(*filename: str, disregard_units: bool = False, registration_file_path: Union[List[str], str, None] = None, mask: slice = Ellipsis*) → velociraptor.catalogue.catalogue.Catalogue

Loads a velociraptor catalogue, producing a Catalogue object.

Parameters

- **filename** (*str*) – The filename of your VELOCIraptor catalogue file (i.e. the path to the .properties file).
- **disregard_units** (*bool, optional*) – If `True`, then disregard any additional units in the VELOCIraptor catalogues, and instead base everything on the 'base' units of velocity, length, and mass. In this case metallicities are left dimensionless. If you are using EAGLE data, you should set this to False, as the star formation rate units are presented in non-internal units.
- **registration_file_path** (*Union[List[str], str], optional*) – The filename of the derived quantities script(s) to register additional properties with the catalogue. This is an advanced feature. See the documentation for more details.
- **mask** (*Union[None, NDArray[bool], int], optional*) – If a boolean array is provided, it is used to mask all catalogue arrays. If an int is provided, catalogue arrays are masked to the single corresponding element.

Returns The Catalogue object that describes your .properties file.

Return type *Catalogue*

7.1.1 Subpackages

velociraptor.autoplotter package

Autoplotter module. Contains the functionality to use the library to automatically make plots of variable x against variable y, by reading from a yaml file.

This yaml file has the following format:

```
output_filename:
  # Global plot quantities
  type: scatter / histogram / cumulative_histogram / 2dhistogram / massfunction /␣
↪adaptivemassfunction
  select_structure_type: number of substructure to select (e.g. 10 for centrals)
  exclude_structure_type: number of substructure to exclude (e.g. 10 for non-
↪centrals). Cannot be a number equal to select_structure_type.
  selection_mask: "quantity.name" - a boolean quantity to select the items you wish␣
↪to plot.
  number_of_bins: number of bins in the (background) histogram or massfunc
  min_num_points_highlight: Minimum number of data points (with the largest values of␣
↪x) to highlight in mean- and median-line plots (default: 10)
  reverse_cumsum: reverse the cumulative sum in cumulative histogram plots? Choose␣
↪true or false
  comment: "An extra string to be placed on the plot"
  legend_loc: Location of legend (string). One of:
         "upper right",
         "upper left",
         "lower left",
         "lower right",
         "right",
         "lower center",
         "upper center",
         "center",
  redshift_loc: Location of z=x a=y string. Same vailidity as legend_loc.
  comment_loc: Location of extra comment string. Same validity as legend_loc
  # Horizontal quantity
  x:
    quantity: "quantity.name" (e.g. masses.mass_200crit)
    units: units to plot in (e.g. Solar_Mass)
    start: horizontal axis start (in units above)
    end: horizontal axis to end (in units above)
    log: true/false (true by default), do you want a log axis?
    label_override: override the label with your own choice
    # if hist or massfunc, in units above
    start_bin: number to start binning at (default is start)
    end_bin: number to end binning at (default is end)
    # Shade below or above a value, to show you do not trust this region
    shade:
      below: number (in same units)
      above: number
  # Vertical quantity - only need units for massfunction
  y:
    quantity: "quantity.name" (e.g. masses.mass_200crit)
    units: units to plot in (e.g. Solar_Mass)
    start: vertical axis start (in units above)
    end: vertical axis to end (in units above)
    log: true/false (true by default), do you want a log axis?
    label_override: override the label with your own choice
```

(continues on next page)

(continued from previous page)

```
    # Shade below or above a value, to show you do not trust this region
    shade:
      below: number (in same units)
      above: number
  # Lines that you can plot on top - line_type can be median or mean
  line_type:
    plot: true/false (true by default) actually plot this?
    log: true/false (true by default) use log bins?
    scatter: "none", "errorbar", "errorbar_both", or "shaded". Defaults to shaded
    number_of_bins: number of bins for median line, different from number_of_bins␣
↪above
    adaptive: use adaptive binning?
    start: value to start binning at
      value: start value
      units: start value units
    end:
      value: end value
      units: end value units
    # Restriction of binning vertically
    lower:
      value: lower value for median/mean line
      units: lower value units
    upper:
      value: upper value for median/mean line
      units: upper value units
  # Generic metadata for plot
  metadata:
    title: title for the plot, should you want one
    caption: caption for the plot
    section: section the plot 'lives in' for pipeline usage.
    observational_data_bracket_width: 0.1 - the width in log(a) of the window to plot␣
↪observational
    data (defined below) within.
  # Comparison data to plot on top
  observational_data:
    - filename: filename of data to plot on this plot (use ObservationalData written␣
↪to disk)
    - filename: second filename for observational data
    - filename: ... you can plot as many of these as you'd like
```

Submodules

velociraptor.autoplotter.box_size_correction module

velociraptor.autoplotter.compare module

velociraptor.autoplotter.lines module

velociraptor.autoplotter.metadata module

velociraptor.autoplotter.objects module

velociraptor.autoplotter.plot module

velociraptor.catalogue package

Submodules

velociraptor.catalogue.catalogue module

Main objects for the velociraptor reading library.

This is based upon the reading routines in the SWIFTsimIO library.

exception velociraptor.catalogue.catalogue.**CatalogueTypeError**
Bases: Exception

class velociraptor.catalogue.catalogue.**Catalogue**(*type: str*)
Bases: object

get_quantity(*quantity_name: str*)
Get a quantity from the catalogue.

Parameters **quantity_name** (*str*) – Full path to the quantity.

register_derived_quantities(*registration_file_path: Union[List[str], str]*) → None
Register any required derived quantities. These will be available through *catalogue.derived_quantities.{your_names}*.

For more information on this feature, see the documentation of *velociraptor.catalogue.derived.DerivedQuantities*.

Parameters **registration_file_path** (*Union[List[str], str]*) – Path to the python source file(s) that contain(s) the code to register the additional derived quantities.

velociraptor.catalogue.derived module

Objects required to register derived quantities. Enables the extension of the *catalogue* object.

class velociraptor.catalogue.derived.**DerivedQuantities**(*registration_file_path: Union[List[str], str], catalogue*)
Bases: object

Derived quantities class. Contains methods to open (a) python source file(s) that create(s) derived quantities as follows:

The source file will have access to:

- numpy (imported as np)
- unyt (imported as unyt)

You should write your derived quantities as follows:

```
self.derived_quantitiy_name = catalogue.type.field_name * 0.5
self.derived_quantitiy_name_two = (
    catalogue.type.field_name / catalogue.type.field_name_two
)
```

For example, to register the specific star formation rate in apertures:

```
for aperture_size in [5, 10, 30, 50, 100]:
    stellar_mass = catalogue.get_quantity(
        f"apertures.mass_star_{aperture_size}_kpc"
    )

    star_formation_rate = catalogue.get_quantity(
        f"apertures.sfr_gas_{aperture_size}_kpc"
    )

    ssfr = star_formation_rate / stellar_mass
    # Don't forget to set the name or your plot will be un-labeled!
    ssfr.name = f"Specific SFR ({aperture_size} kpc)"

    setattr(
        self, f"specific_sfr_gas_{aperture_size}_kpc", ssfr
    )
```

The path to this file(s) should be passed to the __init__ method of this class. Note that you should only register quantities that you do in fact intend to use, as these are not lazily loaded in the same way as the properties that are built into catalogues. For example, in the above registration both the star formation rate and stellar mass are loaded from the VELOCIraptor catalogue file, a behaviour which may not be ideal if the catalogue is very large.

velociraptor.catalogue.registration module

Default registration functions.

If you add one, don't forget to add it to global_registration_functions at the end of the file.

velociraptor.catalogue.registration.**registration_fail_all**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits*) -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Basic registration function showing function signature that is required and automatically fails all tests against itself.

Function signature:

- field_path: the name of the field
- unit_system: a VelociraptorUnits instance that contains all unit information that is available from the velociraptor catalogue

Return signature:

- field_units: the units that correspond to field_path.
- name: A fancy (possibly LaTeX'd) name for the field.
- snake_case: A correct snake_case name for the field.

velociraptor.catalogue.registration.**registration_apertures**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> *(<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registers aperture values by searching them with regex.

velociraptor.catalogue.registration.**registration_projected_apertures**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> *(<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registers aperture values by searching them with regex.

velociraptor.catalogue.registration.**registration_energies**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> *(<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registers all energy related quantities (those beginning with E).

velociraptor.catalogue.registration.**registration_ids**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> *(<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registers all quantities related to particle ids and halo ids (those beginning or ending with ID).

velociraptor.catalogue.registration.**registration_rotational_support**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> *(<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registers rotational support quantities (those beginning with K). Note that this corresponds to kappa in Sales+2010 _not_ K.

velociraptor.catalogue.registration.**registration_angular_momentum**(*field_path: str, unit_system: velociraptor.units.VelociraptorUnits)* -> *(<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registers values starting with L, those that represent angular momenta.

velociraptor.catalogue.registration.**registration_masses**(*field_path: str, unit_system: velociraptor.units.VelociraptorUnits)* -> *(<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registration for all mass-based quantities. (Start with M)

velociraptor.catalogue.registration.**registration_rvmax_quantities**(*field_path: str, unit_system: velociraptor.units.VelociraptorUnits)* -> *(<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registration for all quantities measured within RVmax (Start with RVmax)

velociraptor.catalogue.registration.**registration_radii**(*field_path: str, unit_system: velociraptor.units.VelociraptorUnits)* -> *(<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registration for all radii quantities (start with **R_**)

velociraptor.catalogue.registration.**registration_star_formation_rate**(*field_path: str, unit_system: velociraptor.units.VelociraptorUnits)* -> *(<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registers star formation rate quantities. (Start with SFR)

velociraptor.catalogue.registration.**registration_temperature**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Registers temperature based quantites (Those beginning with T).

velociraptor.catalogue.registration.**registration_structure_type**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Registers the StructureType field.

velociraptor.catalogue.registration.**registration_velocities**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Registers velocity quantities (those starting with V).

velociraptor.catalogue.registration.**registration_positions**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Registers all positon based quantities (those beginning with X, Y, or Z).

velociraptor.catalogue.registration.**registration_concentration**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Registers concentration values (those beginning with c).

velociraptor.catalogue.registration.**registration_metallicity**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Registers metallicity-based quantities (those beginning with Zmet)

velociraptor.catalogue.registration.**registration_eigenvectors**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Registers eigenvector quantities (those beginning with eig).

velociraptor.catalogue.registration.**registration_veldisp**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Registers velocity dispersion quantities (those beginning with veldisp).

velociraptor.catalogue.registration.**registration_stellar_age**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Registers the stellar ages properties (currently tage_star).

velociraptor.catalogue.registration.**registration_element_mass_fractions**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Registers the element mass fraction properties.

Hopefully this is changed in the future as this is a mess.

velociraptor.catalogue.registration.**registration_dust_mass_fractions**(*field_path: str, unit_system: velociraptor.units.VelociraptorUnits) -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registers the dust mass fraction properties.

Hopefully this is changed in the future as this is a mess.

velociraptor.catalogue.registration.**registration_number**(*field_path: str, unit_system: velociraptor.units.VelociraptorUnits) -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registers the number of particles in each halo (n_{bh, gas, star} and npart).

velociraptor.catalogue.registration.**registration_gas_H_and_He_masses**(*field_path: str, unit_system: velociraptor.units.VelociraptorUnits) -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registers the masses in Hydrogen & Helium within apertures

velociraptor.catalogue.registration.**registration_gas_diffuse_element_masses**(*field_path: str, unit_system: velociraptor.units.VelociraptorUnits) -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registers the masses in Hydrogen & Helium within apertures

velociraptor.catalogue.registration.**registration_dust_masses_from_table**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits*) -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Registers the dust mass fraction properties.

velociraptor.catalogue.registration.**registration_gas_hydrogen_species_masses**(*field_path: str*, *unit_system: velociraptor.units.Velocirapto* -> (<class 'unyt.unit_object.Un <class 'str'>, <class 'str'>)

Registers the masses in hydrogen species within apertures.

velociraptor.catalogue.registration.**registration_cold_dense_gas_properties**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorU* -> (<class 'unyt.unit_object.Unit' <class 'str'>, <class 'str'>)

Registers the mass of cold (T < 10^4.5 K), dense (nH > 0.1 cm^-3) gas in apertures.

velociraptor.catalogue.registration.**registration_log_element_ratios_times_masses**(*field_path: str, unit_system: velociraptor.units.Veloc -> (<class 'unyt.unit_obj <class 'str'>, <class 'str'>)*

Registers the log10(Fe/H) times mass and log10(O/H) times mass within apertures for two particle floor values

velociraptor.catalogue.registration.**registration_lin_element_ratios_times_masses**(*field_path: str, unit_system: velociraptor.units.Veloc -> (<class 'unyt.unit_obj <class 'str'>, <class 'str'>)*

Registers the Fe/H times mass and O/H times mass within apertures for two particle floor values

velociraptor.catalogue.registration.**registration_dust_masses**(*field_path: str, unit_system: velociraptor.units.VelociraptorUnits) -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registers the masses in dust within apertures

velociraptor.catalogue.registration.**registration_stellar_luminosities**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Registers the luminosities within apertures

velociraptor.catalogue.registration.**registration_hydrogen_phase_fractions**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Registers the phase fractions for hydrogen.

velociraptor.catalogue.registration.**registration_black_hole_masses**(*field_path: str*, *unit_system: velociraptor.units.VelociraptorUnits)* -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Sub-grid black hole property registrations.

velociraptor.catalogue.registration.**registration_stellar_birth_densities**(*field_path: str, unit_system: velociraptor.units.VelociraptorUnits*) -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Stellar birth density registrations.

velociraptor.catalogue.registration.**registration_snii_thermal_feedback_densities**(*field_path: str, unit_system: velociraptor.units.Veloc*) -> (<class 'unyt.unit_obj <class 'str'>, <class 'str'>)

SNII thermal feedback density registrations.

velociraptor.catalogue.registration.**registration_species_fractions**(*field_path: str, unit_system: velociraptor.units.VelociraptorUnits*) -> (<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)

Registers the species mass fraction properties.

Hopefully this is changed in the future as this is a mess.

velociraptor.catalogue.registration.**registration_spherical_overdensities**(*field_path: str, unit_system: velociraptor.units.VelociraptorUnits* -> *(<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registers SO aperture values by searching them with regex.

velociraptor.catalogue.registration.**registration_element_masses_in_stars**(*field_path: str, unit_system: velociraptor.units.VelociraptorUnits* -> *(<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registers element masses contained in stars

velociraptor.catalogue.registration.**registration_snia_rates**(*field_path: str, unit_system: velociraptor.units.VelociraptorUnits)* -> *(<class 'unyt.unit_object.Unit'>, <class 'str'>, <class 'str'>)*

Registers the SNIa rates within apertures

velociraptor.catalogue.soap_catalogue module

class velociraptor.catalogue.soap_catalogue.**SWIFTUnitsMockup**(*units: Dict[KT, VT]*)

Bases: object

Takes a dict with internal swift units and generates a unyt system compatible with swiftsimio.

mass
unit for mass used

Type float

length
unit for length used

Type float

time
unit for time used

Type float

current
unit for current used

Type float

temperature
unit for temperature used

Type float

class velociraptor.catalogue.soap_catalogue.**CatalogueElement**(*file_name: pathlib.Path*, *name: str*)

Bases: object

Abstract class for catalogue elements. These map to specific objects in the SOAP output file.

The SOAP output file is a tree structure with HDF5 groups that contain either more HDF5 groups or HDF5 datasets. Each group/dataset has a name that corresponds to its path in the SOAP file.

class velociraptor.catalogue.soap_catalogue.**CatalogueDataset**(*file_name: pathlib.Path*, *name: str*, *handle: h5py._hl.files.File*)

Bases: *velociraptor.catalogue.soap_catalogue.CatalogueElement*

Representation of a SOAP dataset.

A dataset has unit metadata and values that are only read if the dataset is actually used.

set_value(*value: unyt.array.unyt_array*, *group: velociraptor.catalogue.soap_catalogue.CatalogueGroup*)
Setter for the dataset values.

Parameters

- **value** (–) – unyt.unyt_array New values for the dataset.
- **group** (–) – CatalogueGroup Group this dataset belongs to. Only provided for property() compatibility (since we want the dataset to be a property of the CatalogueGroup object).

del_value(*group: velociraptor.catalogue.soap_catalogue.CatalogueGroup*)
Deleter for the dataset values.

Parameters **group** (–) – CatalogueGroup Group this dataset belongs to. Only provided for property() compatibility (since we want the dataset to be a property of the CatalogueGroup object).

get_value(*group: velociraptor.catalogue.soap_catalogue.CatalogueGroup*) → unyt.array.unyt_array
Getter for the dataset values. Performs lazy reading: if the value has not been read before, it is read from the SOAP catalogue file. Otherwise, a buffered value is used.

Parameters **group** (–) – CatalogueGroup Group this dataset belongs to. Only provided for property() compatibility (since we want the dataset to be a property of the CatalogueGroup object).

Returns the dataset values as a unyt.unyt_array.

class velociraptor.catalogue.soap_catalogue.**CatalogueDerivedDataset**(*file_name: pathlib.Path, name: str, terms: List[velociraptor.catalogue.soap_*

Bases: *velociraptor.catalogue.soap_catalogue.CatalogueElement*, abc.ABC

Representation of a derived SOAP dataset.

A derived dataset is a dataset that can be trivially derived from another (set of) dataset(s). It is similar to a dataset registered using a registration function, but still supports lazy evaluation. Its main purpose is to guarantee full compatibility between the SOAP and VR catalogues in the pipeline.

set_value(*value: unyt.array.unyt_array*, *group: velociraptor.catalogue.soap_catalogue.CatalogueGroup*)
Setter for the dataset values.

Parameters

- **value** (–) – unyt.unyt_array New values for the dataset.
- **group** (–) – CatalogueGroup Group this dataset belongs to. Only provided for property() compatibility (since we want the dataset to be a property of the CatalogueGroup object).

del_value(*group: velociraptor.catalogue.soap_catalogue.CatalogueDataset*)
Deleter for the dataset values.

Parameters **group** (–) – CatalogueGroup Group this dataset belongs to. Only provided for property() compatibility (since we want the dataset to be a property of the CatalogueGroup object).

get_value(*group: velociraptor.catalogue.soap_catalogue.CatalogueDataset*) → unyt.array.unyt_array
Getter for the dataset values. Performs lazy evaluation: if the value has not been computed before, all the datasets that are required to compute it are obtained (and might be lazily read at this point), and the child class specific computation method is called. Otherwise, a buffered value is returned.

Parameters **group** (–) – CatalogueGroup Group this dataset belongs to. Only provided for property() compatibility (since we want the dataset to be a property of the CatalogueGroup object).

Returns the dataset values as a unyt.unyt_array.

compute_value(**values*) → unyt.array.unyt_array
Subclass specific computation method for the derived dataset. Should be implemented by each subclass.

Parameters ***values** (–) – unyt.unyt_array Datasets that are required for the calculation.

Returns the computed values as a unyt.unyt_array.

class velociraptor.catalogue.soap_catalogue.**VelocityDispersion**(*file_name: pathlib.Path, name: str, terms: List[velociraptor.catalogue.soap_catalog*

Bases: *velociraptor.catalogue.soap_catalogue.CatalogueDerivedDataset*

CatalogueDerivedDataset that computes the 1D velocity dispersion from the full 3D velocity dispersion matrix.

compute_value(*velocity_dispersion_matrix: unyt.array.unyt_array*) → unyt.array.unyt_array
Calculate the 1D velocity dispersion from the velocity dispersion matrix.

The velocity dispersion matrix in SOAP consists of the 6 non-trivial elements of

V_{ij} = Sigma_p (v_{p,i} - <v>_i)**2 * (v_{p,j} - <v>_j)**2,

where v_p is the particle velocity and <v> is a reference velocity.

Since this is a symmetric matrix, SOAP only outputs (in this order) V_XX, V_YY, V_ZZ, V_XY, V_XZ, V_YZ

The 1D velocity dispersion output by VR is sqrt(V_XX + V_YY + V_ZZ)

Parameters **velocity_dispersion_matrix** (–) – unyt.unyt_array SOAP velocity dispersion matrix.

Returns the 1D velocity dispersion as a unyt.unyt_array.

class velociraptor.catalogue.soap_catalogue.**CatalogueGroup**(*file_name: pathlib.Path, str, name: handle: h5py._hl.files.File*)
Bases: *velociraptor.catalogue.soap_catalogue.CatalogueElement*

Representation of an HDF5 group in the SOAP catalogue.

A CatalogueGroup contains other groups or datasets. Datasets are exposed as new attributes for the CatalogueGroup object, so that things like

so.v200_crit.totalmass

map directly to the CatalogueDataset::get_value() function that retrieves
SOAP_catalogue["SO/200_crit/TotalMass"][:]

Note that attribute names cannot start with a number, so we use the convention that numeric group names are preceded by a 'v' (for value).

velociraptor.catalogue.soap_catalogue.**dynamically_register_properties**(*group: velociraptor.catalogue.soap_catalogue.*
Trick an object into thinking it is of a different class that has additional properties, based on the properties contained in the group.properties dictionary.

Concrete example: suppose 'group' enters this method as group.elements = [CatalogueDataset<a>, CatalogueDataset<b>] group.properties= {"a_name": (CatalogueDataset<a>, CatalogueDataset<a>.value),

"b_name": (CatalogueDataset<a>, CatalogueDataset<a>.value)},

where we have used CatalogueDataset<x>.value

as a shorthand for

property(CatalogueDataset<x>.get_value, CatalogueDataset<x>.set_value, CatalogueDataset<x>.del_value)

After this method acts on 'group', it will look like this: group.a_name = CatalogueDataset<a>.get_value group.b_name = CatalogueDataset<b>.get_value

class velociraptor.catalogue.soap_catalogue.**SOAPCatalogue**(*file_name: pathlib.Path*)

Bases: *velociraptor.catalogue.catalogue.Catalogue*

Catalogue specialisation for a SOAP catalogue.

print_fields()
Debugging function used to output the fields that were actually accessed sicne the catalogue was opened.

get_SOAP_quantity(*quantity_name: str*) → unyt.array.unyt_array
Get the quantity with the given name from the catalogue.

Quantities should be addressed using their full path in the catalogue file, with the convention that the name is fully written in small caps and that '/' is replaced with '.'. Since attribute names cannot start with a digit, we additionally add a 'v' in between '.' and any digit.

Parameters **quantity_name** (–) – str Full path to the quantity in the SOAP catalogue.

Returns the corresponding quantity as a unyt.unyt_array.

get_quantity(*quantity_name: str*) → unyt.array.unyt_array
Get the quantity with the given name from the catalogue.

This version uses a fallback mechanism to deal with quantities that are addressed using the wrong name, i.e. quantities for which the old VR catalogue name is used. We first try to use the parent class get_quantity() version that assumes all datasets are simply exposed as attributes. If this fail, we use the SOAP catalogue version above. If that fails too, we try to find a SOAP equivalent for the given quantity name in the VR_to_SOAP translator function. If that fails to, we bail out with a NotImplementedError.

Parameters **quantity_name** (–) – str Full path to the quantity in the SOAP catalogue.

Returns the corresponding quantity as a unyt.unyt_array.

velociraptor.catalogue.translator module

Routines that provide translation of velociraptor quantities into something a little more human readable, or to internal quantities.

velociraptor.catalogue.translator.**VR_to_SOAP**(*particle_property_name: str*) → str
Convert a VR property name into its SOAP counterpart (if one exists).

Parameters **particle_property_name** (–) – str VR-like property name.

Returns the SOAP-like equivalent of the same name, if one exists.

velociraptor.catalogue.translator.**typo_correct**(*particle_property_name: str*)
Corrects for any typos in field names that may exist.

velociraptor.catalogue.translator.**get_aperture_unit**(*unit_name: str*, *unit_system: velociraptor.units.VelociraptorUnits*)
Converts the velociraptor strings to internal velociraptor units from the naming convention in the velociraptor files.

velociraptor.catalogue.translator.**get_particle_property_name_conversion**(*name: str*, *ptype: str*)
Takes an internal velociraptor particle property and returns a fancier name for use in plot legends. Typically used for the complex aperture properties.

velociraptor.catalogue.velociraptor_catalogue module

Main objects for the velociraptor reading library.

This is based upon the reading routines in the SWIFTsimIO library.

class velociraptor.catalogue.velociraptor_catalogue.**VelociraptorFieldMetadata**(*filename, path: str, registration_functions: Dict[str, Callable], units: velociraptor.units.Velocirap*

Bases: object

Metadata for a velociraptor field. Pass it a field path and a filename, and it will:

- Use the registration functions to find the correct units and "fancy name".
- Assign a proper snake_case_name to the dataset.

valid = False

name = ''

snake_case = ''

register_field_properties()
Registers the field properties using the registration functions.

velociraptor.catalogue.velociraptor_catalogue.**generate_getter**(*filename*, *name: str*, *field: str*, *full_name: str*, *unit*)

Generates a function that:

a) If self._'name' exists, return it

b) If not, open *filename*

c) Reads filename[*field*]

d) Set self._'name'

e) Return self._'name'.

Takes:

- filename, the filename of the hdf5 file
- name, the snake_case name of the property
- field, the field in the hdf5 file corresponding to this property
- full_name, the fancy printing name for this quantity (registered to array.name)
- unit, the unyt unit corresponding to this value

velociraptor.catalogue.velociraptor_catalogue.**generate_setter**(*name: str*)
Generates a function that sets self._name to the value that is passed to it.

velociraptor.catalogue.velociraptor_catalogue.**generate_deleter**(*name: str*)
Generates a function that destroys self._name (sets it back to None).

velociraptor.catalogue.velociraptor_catalogue.**generate_sub_catalogue**(*filename, registration_name: str, registration_function: Callable, units: velociraptor.units.VelociraptorUnits, field_metadata: List[velociraptor.catalogue.velo mask: slice = Ellipsis*)
Generates a sub-catalogue object with the correct properties set. This is required as we can add properties to a class, but _not_ to an object dynamically.

So, here, we initialise the metadata, create a _copy_ of the __VelociraptorSubCatlaogue class, and then add all of our properties to that _class_ before instantiating it with the metadata.

class velociraptor.catalogue.velociraptor_catalogue.**VelociraptorCatalogue**(*filename: str, disregard_units: bool = False, extra_registration_functior Union[None, Dict[str, Callable]] = None, mask: slice = Ellipsis*)
Bases: *velociraptor.catalogue.catalogue.Catalogue*

A velociraptor dataset, containing information that has correct units and are easily accessible through snake_case names.

get_units()

Gets the units instance from the file properties.

extract_properties_from_units()
Use the self.units object to extract interesting parameters that should be visible from the top-level.

centrals

satellites

velociraptor.fitting_formulae package

Submodules

velociraptor.fitting_formulae.smhmr module

Fitting formulae for the stellar mass-halo mass relation.

velociraptor.fitting_formulae.smhmr.**moster**(*catalogue: velociraptor.catalogue.catalogue.Catalogue*, *mass_range: Union[unyt.array.unyt_array, List[unyt.array.unyt_quantity]] = [unyt_quantity(10000., 'Msun'), unyt_quantity(1.e+16, 'Msun')]*, *n_eval: int = 256*)
The moster retaion (from Moster+ 2013). Original code provided by Matthieu Schaller. Takes:

- catalogue, your catalogue object
- **mass_range, a length 2 array with the lowest and highest halo mass** you would like the relation to be evaluated between (default: 1e4 msun to 1e16 msun)
- **n_eval, the number of function evaluations (equally spaced in log (Mhalo))** (default: 256)

Returns:

- halo_mass, the halo masses at which the relation is evaluated at
- stellar_mass, the stellar masses matching with the above halo masses

velociraptor.fitting_formulae.smhmr.**moster_raw**(*z*, *Mhalo*)
Stellar mass-halo mass relation from Moster+2013.

Provided by Matthieu Schaller.

velociraptor.fitting_formulae.smhmr.**behroozi**(*catalogue: velociraptor.catalogue.catalogue.Catalogue*, *mass_range: Union[unyt.array.unyt_array, List[unyt.array.unyt_quantity]] = [unyt_quantity(10000., 'Msun'), unyt_quantity(1.e+16, 'Msun')]*, *n_eval: int = 256*)
The behroozi fit to the SMHMR (from Behroozi+ 2013). Original code provided by Matthieu Schaller. Takes:

- catalogue, your catalogue object
- **mass_range, a length 2 array with the lowest and highest halo mass** you would like the relation to be evaluated between (default: 1e4 msun to 1e16 msun)
- **n_eval, the number of function evaluations (equally spaced in log (Mhalo))** (default: 256)

Returns:

- halo_mass, the halo masses at which the relation is evaluated at
- stellar_mass, the stellar masses matching with the above halo masses

velociraptor.fitting_formulae.smhmr.**behroozi_raw**(*z*, *Mhalo*)
Stellar mass-halo mass relation from Behroozi +2013.

Provided by Matthieu Schaller.

velociraptor.fitting_formulae.smhmr.**behroozi_2019_raw**(*z*, *Mhalo*)
Stellar mass-halo mass relation from Behroozi +2019.

The data is taken from https://www.peterbehroozi.com/data.html

This function is a median fit to the raw data for centrals (i.e. excluding satellites)

The stellar mass is the true stellar mass (i.e. w/o observational corrections)

The fitting function does not include the intrahalo light contribution to the stellar mass

The halo mass is the peak halo mass that follows the Bryan & Norman (1998) spherical overdensity definition

Provided by Evgenii Chaikin.

velociraptor.observations package

Sub-module for adding observational data to plots.

Includes the ObservationalData object and helper functions to convert data to this new format.

velociraptor.observations.**load_observation**(*filename: str*)
Load an observation from file filename. This should be in the standard velociraptor format.

Deprecated in favour of *load_observations()*

Parameters **filename** (*str*) – Filename of the observational dataset that you wish to load. Should probably end in .hdf5. See the documentation for *velociraptor.observations.objects.ObservationalData* for more information.

Returns Observational data instance read from file.

Return type *velociraptor.observations.objects.ObservationalData*

velociraptor.observations.**load_observations**(*filenames: Union[str, Iterable[str]]*, *redshift_bracket: List[float] = [0.0, 1000.0]*)

Parameters

- **filename** (*str, Iterable[str]*) – Filename(s) of the observational dataset that you wish to load. Should probably end in .hdf5. See the documentation for *velociraptor.observations.objects.ObservationalData* and *velociraptor.observations.objects.MultiRedshiftObservationalData* for more information.
- **redshift_bracket** (*str*) – Redshift bracket to overlap with. If any of the observations in the file overlap with this bracket, they are returned. By default, this bracket is 0.0 to 1000.0, so will encompass all reasonable observations present in the file.

Returns Observational data instances read from file that overlap with your specified redshift bracket.

Return type List[*velociraptor.observations.objects.ObservationalData*]

Submodules

velociraptor.observations.objects module

Objects for observational data plotting.

Tools for adding in extra (e.g. observational) data to plots.

Includes an object container and helper functions for creating and reading files.

velociraptor.observations.objects.**save_cosmology**(*handle: h5py._hl.files.File*, *cosmology: astropy.cosmology.core.Cosmology*)

Save the (astropy) cosmology to a HDF5 dataset.

Parameters

- **handle** (*h5py.File*) – h5py file handle to save the cosmology to. This is performed by creating a cosmology group and setting attributes.
- **cosmology** (*astropy.cosmology.Cosmology*) – The Astropy cosmology instance to save to the HDF5 file. This is performed by extracting all of the key variables and saving them as either floating point numbers or strings.

Notes

This process can be reversed by using load_cosmology.

velociraptor.observations.objects.**load_cosmology**(*handle: h5py._hl.files.File*)

Save the (astropy) cosmology to a HDF5 dataset.

Parameters **handle** (*h5py.File*) – h5py file handle to read the cosmology from.

Returns Astropy cosmology instance extracted from the HDF5 file.

Return type astropy.cosmology.Cosmology

class velociraptor.observations.objects.**ObservationalData**

Bases: object

Observational data object. Contains routines for both writing and reading HDF5 files containing the observations, as well as plotting.

name

Name of the observation for users to identifty

Type str

x_units

Units for horizontal axes

Type unyt_quantity

y_units

Units for vertical axes

Type unyt_quantity

x

Horizontal data points

Type unyt_array

y
Vertical data points

Type unyt_array

x_scatter
Scatter in horizontal direction. Can be None, or an unyt_array of shape 1XN (symmetric) or 2XN (non-symmetric) such that it can be passed to plt.errorbar easily.

Type Union[unyt_array, None]

y_scatter
Scatter in vertical direction. Can be None, or an unyt_array of shape 1XN (symmetric) or 2XN (non-symmetric) such that it can be passed to plt.errorbar easily.

Type Union[unyt_array, None]

x_comoving
Whether or not the horizontal values are comoving (True) or physical (False)

Type bool

y_comoving
Whether or not the vertical values are comoving (True) or physical (False)

Type bool

x_description
Default label for horizontal axis (without units), also a description of the variable.

Type str

y_description
Default label for horizontal axis (without units), also a description of the variable.

Type str

filename
Filename that the data was read from, or was written to.

Type str

comment
A free-text comment describing the data, including e.g. which cosmology and IMF it is calibrated to.

Type str

citation
Short citation for data, e.g. Author et al. (Year) (Project), such as Baldry et al. (2012) (GAMA)

Type str

bibcode
Bibcode for citation, this can be found on the NASA ADS.

Type str

redshift
Redshift at which the data is collected at. If a range, use the mid-point.

Type float

redshift_lower
Lowest redshift at which the data is collected at. Used to determine whether it should be plotted on a given figure.

Type float

redshift_upper
Highest redshift at which the data is collected at. Used to determine whether it should be plotted on a given figure.

Type float

plot_as
Whether the data should be plotted as points (typical for observations) or as a line (typical for simulation data). Allowed values:

- points
- line

Type Union[str, None]

cosmology
Astropy cosmology that the data has been corrected to.

Type Cosmology

plot_as = None

load(*filename: str*, *prefix: Optional[str] = None*)
Loads the observations from file.

Parameters

- **filename** (*str*) – The filename to load the data from. Probably should end in .hdf5.
- **prefix** (*str, optional*) – An optional prefix to all fields that enables multiple observational datasets to be saved in a single file. Not for general use, only within *MultiRedshiftObservationalData*. If a prefix is used, cosmology is not saved.

write(*filename: str*, *prefix: Optional[str] = None*)
Writes the observations to file.

Parameters

- **filename** (*str*) – The filename to write the data to. Probably should end in .hdf5.
- **prefix** (*str, optional*) – An optional prefix to all fields that enables multiple observational datasets to be saved in a single file. Not for general use, only within *MultiRedshiftObservationalData*. If a prefix is used, cosmology is not saved.

associate_x(*array: unyt.array.unyt_array*, *scatter: Optional[unyt.array.unyt_array]*, *comoving: bool*, *description: str*)
Associate an x quantity with this observational data instance.

Parameters

- **array** (*unyt_array*) – The array of (horizontal) data points, including units.
- **scatter** (*Union[unyt_array, None]*) – The array of scatter (1XN or 2XN) in the horizontal co-ordinates with associated units.
- **comoving** (*bool*) – Whether or not the horizontal values are comoving.
- **description** (*str*) – Short description of the data, e.g. Stellar Masses

associate_y(*array: unyt.array.unyt_array*, *scatter: Optional[unyt.array.unyt_array]*, *comoving: bool*, *description: str*)
Associate an y quantity with this observational data instance.

Parameters

- **array** (*unyt_array*) – The array of (vertical) data points, including units.
- **scatter** (*Union[unyt_array, None]*) – The array of scatter (1XN or 2XN) in the vertical co-ordinates with associated units.
- **comoving** (*bool*) – Whether or not the vertical values are comoving.
- **description** (*str*) – Short description of the data, e.g. Stellar Masses

associate_citation(*citation: str*, *bibcode: str*)
Associate a citation with this observational data instance.

Parameters

- **citation** (*str*) – Short citation, formatted as follows: Author et al. (Year) (Project), e.g. Baldry et al. (2012) (GAMA)
- **bibcode** (*str*) – Bibcode for the paper the data was extracted from, available from the NASA ADS or publisher. E.g. 2012MNRAS.421..621B

associate_name(*name: str*)
Associate a name with this observational data instance.

Parameters **name** (*str*) – Short name to describe the dataset.

associate_comment(*comment: str*)
Associate a comment with this observational data instance.

Parameters **comment** (*str*) – A free-text comment describing the data, including e.g. which cosmology and IMF it is calibrated to.

associate_redshift(*redshift: float*, *redshift_lower: Optional[float] = None*, *redshift_upper: Optional[float] = None*)
Associate the redshift that the observations were taken at with this observational data instance.

Parameters

- **redshift** (*float*) – Redshift at which the data is collected at. If a range, use the mid-point.
- **redshift_lower** (*Optional[float]*) – Lower bound for this set of observations. Used to determine if plotting is viable, and should always be present in the case where a multiple redshift dataset is used.
- **redshift_upper** (*Optional[float]*) – Upper bound for this set of observations. Used to determine if plotting is viable, and should always be present in the case where a multiple redshift dataset is used.

associate_plot_as(*plot_as: str*)
Associate the 'plot_as' field - this should either be line or points.

Parameters **plot_as** (*str*) – Either points or line

associate_cosmology(*cosmology: astropy.cosmology.core.Cosmology*)
Associate a cosmology with this dataset that it has been corrected for. This should be an astropy cosmology instance.

Parameters **cosmology** (*astropy.cosmology.Cosmology*) – Astropy cosmology instance describing what cosmology the data has been corrected to.

plot_on_axes(*axes: matplotlib.axes._axes.Axes*, *errorbar_kwargs: Optional[dict] = None*)
Plot this set of observational data as an errorbar().

Parameters

- **axes** (*plt.Axes*) – The matplotlib axes to plot the data on. This will either plot the data as a line or a set of errorbar points, with the short citation (self.citation) being included in the legend automatically.
- **errorbar_kwargs** (*dict*) – Optional keyword arguments to pass to plt.errorbar.

class velociraptor.observations.objects.**MultiRedshiftObservationalData**

Bases: object

Multi-redshift version of *ObservationalData* class, which should be used instead of the *ObservationalData* class for reading files, and writing multiple redshift files.

Essentially, this contains multiple instances of *ObservationalData*, and allows for multiple redshift data to be read transparently.

code_version = '0.16.1'

maximum_number_of_returns = 1024

get_datasets_overlapping_with(*redshifts: List[float] = [0.0, 1000.0]*) → List[velociraptor.observations.objects.ObservationalData]

Gets individual redshift datasets overlapping with the specified redshift range. The check is performed inclusively, so if you ask for overlaps with [0.25, 0.75], and an observation has a redshift range of [0.75, 1.25], it will be included. Note that maximum_number_of_returns modifies the behaviour of this function, and the maximum length of the returned list will be the same as that attribute.

Parameters **redshifts** (*List[float]*) – Redshifts to check for overlaps with. This defaults to the range [0.0, 1000.0], and hence should overlap with all reasonable datasets.

Notes

You can access this ability in a slightly more user-friendly way using the *velociraptor.observations.load_observations()* function.

Datasets are returned in order of their scale-factor proximity to the centre of the range specified in redshifts.

associate_dataset(*dataset: velociraptor.observations.objects.ObservationalData*)

Associate an individual redshift dataset with this object.

Parameters **dataset** (ObservationalData) – Instance of ObservationalData that may or may not be completed; comments are handled at the top level and are over-written. In total, citation, bibcode, name, comment, and cosmology are shared.

associate_citation(*citation: str*, *bibcode: str*)

Associate a citation with this observational data instance.

Parameters

- **citation** (*str*) – Short citation, formatted as follows: Author et al. (Year) (Project), e.g. Baldry et al. (2012) (GAMA)
- **bibcode** (*str*) – Bibcode for the paper the data was extracted from, available from the NASA ADS or publisher. E.g. 2012MNRAS.421..621B

associate_name(*name: str*)

Associate a name with this observational data instance.

Parameters **name** (*str*) – Short name to describe the dataset.

associate_comment(*comment: str*)

Associate a comment with this observational data instance.

Parameters **comment** (*str*) – A free-text comment describing the data, including e.g. which cosmology and IMF it is calibrated to.

associate_cosmology (*cosmology: astropy.cosmology.core.Cosmology*)
Associate a cosmology with this dataset that it has been corrected for. This should be an astropy cosmology instance.

Parameters **cosmology** (*astropy.cosmology.Cosmology*) – Astropy cosmology instance describing what cosmology the data has been corrected to.

associate_maximum_number_of_returns (*maximum_number_of_returns: int*)
Associate a maximum number of returned datasets with this object. This number will give the maximum number of datasets that are returned in a call to `load_datasets`. This is particularly useful in the case where you want to provide a large number of fits and only want to show a single curve on the figure.

Parameters **maximum_number_of_returns** (*int*) – The maximum number of datasets to plot simultaneously.

write (*filename: str*)
Writes all of the datasets currently present in the object to a HDF5 file.

Parameters **filename** (*str*) – File to be written to, including the .hdf5.

load (*filename: str*)
Reads all of the datasets from an associated file to the object.

Parameters **filename** (*str*) – File to be read from, including the .hdf5.

velociraptor.particles package

Velociraptor particles interaction functions. These allow users to extract the particles belonging to, e.g. a single halo.

velociraptor.particles.**load_groups** (*filename*, *catalogue: Optional[velociraptor.catalogue.catalogue.Catalogue] = None*) → velociraptor.particles.particles.VelociraptorGroups
Load the groups file, passed by its filename to this function. Passing the velociraptor catalogue to the catalogue argument allows for significantly more information to be extracted, and is highly recommended.

Submodules

velociraptor.particles.particles module

Objects for the particles files. This includes:

- catalog_groups
- catalog_particles
- catalog_parttypes

and their unbound variants.

The objects defined here are as follows:

VelociraptorGroups: This depends on the Catalogue object and allows for users to find information about where in each file each group belongs.

VelociraptorParticles: This depends on VelociraptorGroups and itself allows for the particles in a _single group_ to be loaded. Along with this, we also load the catalog_parttypes file for those selected particles.

class velociraptor.particles.particles.**VelociraptorGroups**(*filename, catalogue: Optional[velociraptor.catalogue.catalogue.Catalog*

Bases: object

Groups object for velociraptor. Contains the information about where the groups start and stop in the catalog_particles file.

extract_halo(*halo_index: int*, *filenames: Optional[Dict[str, str]] = None*)

Get a halo particles object for a given index into the catalogue (NOT the halo unique id). Filenames is either a dictionary with the following structure:

{ "particles_filename": "...", "parttypes_filename": "...", "unbound_particles_filename": "...", "unbound_parttypes_filename": "...",

}

or None, in which case we guess what the filename should be from the filename of the groups that has already been passed.

class velociraptor.particles.particles.**VelociraptorParticles**(*particles_filename*, *parttypes_filename*, *offset: int*, *group_size: int*, *groups_instance: velociraptor.particles.particles.VelociraptorGroups*)

Bases: object

Velociraptor particles object, holds information on a single halo's particles, including which IDs and particle types are in that halo. This provides extra post-processing options, such as splitting the IDs by particle type.

register_halo_attributes(*catalogue: velociraptor.catalogue.catalogue.Catalogue*, *halo_index: int*)

Registers useful halo attributes to this object (such as the mass and radii of the halo).

velociraptor.swift package

Tools for swiftsimio integration.

Submodules

velociraptor.swift.swift module

SWIFTsimIO integration.

This allows users to load the contents of various haloes as *swiftsimio* datasets in a computationally efficient way.

velociraptor.swift.swift.**generate_spatial_mask**(*particles: velociraptor.particles.particles.VelociraptorParticles*, *snapshot_filename*) → swiftsimio.mask

Determines the mask defining the spatial region containing the particles of interest.

This uses *r_max* to define the extent of the region, so you will need to instantiate the VelociraptorParticles instance with an associated catalogue to use this feature, as it requires the knowledge of *r_max*.

Takes two arguments:

- particles, the VelociraptorParticles instance,

- snapshot_filename, the path to the associated SWIFT snapshot.

It returns:

- **mask, an object containing masks for all available datasets in the** swift dataset.

velociraptor.swift.swift.**generate_bound_mask**(*data: swiftsimio.reader.SWIFTDataset*, *particles: velociraptor.particles.particles.VelociraptorParticles*) → collections.namedtuple

Determines the mask defining the particles bound to the object of interest.

Takes two arguments:

- data, a SWIFTDataset, which may be spatially masked,
- particles, the VelociraptorParticles instance.

It returns:

- **mask, an object containing masks for all available datasets in the** swift dataset.

velociraptor.swift.swift.**to_swiftsimio_dataset**(*particles: velociraptor.particles.particles.VelociraptorParticles*, *snapshot_filename*, *generate_extra_mask: bool = False*) → Union[swiftsimio.reader.SWIFTDataset, Tuple[swiftsimio.reader.SWIFTDataset, collections.namedtuple]]

Loads a VelociraptorParticles instance for one halo into a *swiftsimio* masked dataset.

Initially, this uses *r_max* to perform a spatial mask, and then returns the *swiftsimio* dataset and a secondary mask that may be used to extract only the particles that are part of the FoF group.

You will need to instantiate the VelociraptorParticles instance with an associated catalogue to use this feature, as it requires the knowledge of *r_max*.

Takes three arguments:

- particles, the VelociraptorParticles instance,
- snapshot_filename, the path to the associated SWIFT snapshot.
- **generate_extra_mask, whether or not to generate the secondary** mask object that allows for the extraction of particles that are present only in the FoF group.

It returns:

- data, the swiftsimio dataset
- **mask, an object containing masks for all available datasets in the** swift dataset. The initial masking is performed on a spatial only basis, and this is required to only extract the particles in the FoF group as identified by velociraptor. This is only provided if generate_extra_mask has a truthy value.

velociraptor.tools package

Tools sub-module for generating information for plotting.

Submodules

velociraptor.tools.adaptive module

Tools for creating adaptive bins based on an input array.

velociraptor.tools.adaptive.**adaptive_bin_hash**(*values*, *lowest_value*, *highest_value*, *base_n_bins*, *minimum_in_bin*, *logarithmic*, *stretch_final_bin*)

Hash for adaptive binning. Note that this can raise AttributeError in the case where the array is unhashable.

velociraptor.tools.adaptive.**create_adaptive_bins**(*values: unyt.array.unyt_array*, *lowest_value: unyt.array.unyt_quantity*, *highest_value: unyt.array.unyt_quantity*, *base_n_bins: int = 25*, *minimum_in_bin: int = 3*, *logarithmic: bool = True*, *stretch_final_bin: bool = False*)

Creates a set of adaptive bins based on the input values.

Parameters

- **values** (*unyt.unyt_array*) – The array that you want to create a mass function of (usually this is for example halo masses or stellar masses).
- **lowest_value** (*unyt.unyt_quantity*) – the lowest value edge of the bins
- **highest_value** (*unyt.unyt_quantity*) – the highest value edge of the bins
- **base_n_bins** (*unyt.unyt_array, optional*) – The number of equal width bins across the range to use in the case where no adaptive sampling is required. This returns the minimal allowed bin width. Default: 25.
- **minimum_in_bin** (*int, optional*) – The number of objects in a bin for it to be classed as valid. Bins with a number of objects smaller than this are not returned. Bins are stretched so that they have at least this many items in them. Default: 3.
- **logarithmic** (*bool, optional*) – Whether or not to use logarithmically spaced bins. Default: True.
- **stretch_final_bin** (*bool, optional*) – Stretch the final bin to include all values. If `False`, some values may fall outside of all of the bins.

Returns

- **bin_centers** (*unyt.unyt_array*) – The centers of the bins (taken to be the median of the items in the bin).
- **bin_edges** (*unyt.unyt_array, optional*) – Bin edges that were used in the binning process.

Notes

Caches the output as this procedure can be very expensive, and will be repeated several times.

velociraptor.tools.histogram module

Tools for creating histograms. Uses the same API as mass_functions.

velociraptor.tools.histogram.**create_histogram_given_bins**(*masses: unyt.array.unyt_array*, *bins: unyt.array.unyt_array*, *box_volume: unyt.array.unyt_quantity*, *minimum_in_bin: int = 1*, *cumulative: bool = False*, *reverse: bool = False*)

Creates a mass function (with equal width bins in log M) for you to plot.

Parameters

- **masses** (*unyt.unyt_array*) – The array that you want to create a mass function of (usually this is for example halo masses or stellar masses).
- **bins** (*unyt.unyt_array*) – The mass bin edges to use.
- **unyt.unyt_quantity** (*box_volume*) – The volume of the box such that we can return `n / volume` (unused).
- **minimum_in_bin** (*int, optional*) – The number of objects in a bin for it to be classed as valid. Bins with a number of objects smaller than this are not returned. By default this parameter takes a value of 1.
- **cumulative** (*bool, optional*) – Whether to make the histogram cumulative. The default value is false.
- **reverse** (*bool, optional*) – Whether to reverse the cumulative sum, i.e. do the sum from high to low values. The default value is false. Relevant only if cumulative is true.

Returns

- **bin_centers** (*unyt.unyt_array*) – The centers of the bins (taken to be the linear mean of the bin edges).
- **histogram** (*unyt.unyt_array*) – The value of the mass function at the bin centers.
- *None* – Final return is None, as there are no errors associated with this.

velociraptor.tools.histogram.**create_histogram**(*masses: unyt.array.unyt_array*, *lowest_mass: unyt.array.unyt_quantity*, *highest_mass: unyt.array.unyt_quantity*, *box_volume: unyt.array.unyt_quantity*, *n_bins: int = 25*, *minimum_in_bin: int = 1*, *return_bin_edges: bool = False*, *cumulative: bool = False*, *reverse: bool = False*)

Creates a mass function (with equal width bins in log M) for you to plot.

Parameters

- **masses** (*unyt.unyt_array*) – The array that you want to create a mass function of (usually this is for example halo masses or stellar masses).
- **lowest_mass** (*unyt.unyt_quantity*) – the lowest mass edge of the bins
- **highest_mass** (*unyt.unyt_quantity*) – the highest mass edge of the bins
- **bins** (*unyt.unyt_array*) – The mass bin edges to use.

- **unyt.unyt_quantity** (*box_volume*) – The volume of the box such that we can return `n / volume`.
- **minimum_in_bin** (*int, optional*) – The number of objects in a bin for it to be classed as valid. Bins with a number of objects smaller than this are not returned. By default this parameter takes a value of 1.
- **cumulative** (*bool, optional*) – Whether to make the histogram cumulative. The default value is false.
- **reverse** (*bool, optional*) – Whether to reverse the cumulative sum, i.e. do the sum from high to low values. The default value is false. Relevant only if cumulative is true.
- **return_bin_edges** (*bool, optional*) – Return the bin edges used in the binning process? Default is False.

Returns

- **bin_centers** (*unyt.unyt_array*) – The centers of the bins (taken to be the linear mean of the bin edges).
- **mass_function** (*unyt.unyt_array*) – The value of the mass function at the bin centers.
- *None* – Final return is None, as there are no errors associated with this.
- **bin_edges** (*unyt.unyt_array, optional*) – Bin edges that were used in the binning process.

velociraptor.tools.labels module

Tools for generating labels from catalogue datasets.

velociraptor.tools.labels.**get_full_label**(*dataset: unyt.array.unyt_array*)

Get the full label for one of our VelociraptorCatalogue datasets. This will get the automatically generated name and concatenate it with the _current_ untis for that dataset.

velociraptor.tools.labels.**get_mass_function_label_no_units**(*mass_function_sub_label: str*)

Gets a fancy mass-function label such as:

d$n(M_*)$/d$log_{10}M_*$ [Mpc^{-3}]

(this would be for an input of "*" and unyt.Mpc**3).

velociraptor.tools.labels.**get_mass_function_label**(*mass_function_sub_label: str*, *mass_function_units: unyt.unit_object.Unit*)

Gets a fancy mass-function label such as:

d$n(M_*)$/d$log_{10}M_*$ [Mpc^{-3}]

(this would be for an input of "*" and unyt.Mpc**3).

velociraptor.tools.labels.**get_luminosity_function_label_no_units**(*luminosity_function_sub_label: str*)

Gets a fancy luminosity-function label such as:

d$n(M)$/dM [Mpc^{-3} mag^{-1}]

(this would be for an input of "*" and unyt.Mpc**3).

velociraptor.tools.lines module

Tools to generate various lines from datasets.

velociraptor.tools.lines.**binned_mean_line**(*x: unyt.array.unyt_array*, *y: unyt.array.unyt_array*, *x_bins: unyt.array.unyt_array*, *minimum_in_bin: int = 3*, *return_additional: bool = False*, *minimum_additional_points: int = 0*)

Gets a mean (y) line, binned in the x direction.

Parameters

- **x** (*unyt.unyt_array*) – Horizontal values, to be binned.
- **y** (*unyt.unyt_array*) – Vertical values, to have the mean calculated in the x-bins
- **x_bins** (*unyt.unyt_array*) – Horizontal bin edges. Must have the same units as x.
- **minimum_in_bin** (*int, optional*) – Minimum number of items in a bin to return that bin. If a bin has fewer values than this, it is excluded from the return values. Default: 3.
- **return_additional** (*bool, optional*) – Boolean. If true, makes the function return x and y data points that lie in the bins where the number of data points is smaller than minimum_in_bin, and any points that are higher than the highest bin edge. Default: false.
- **minimum_additional_points** (*int, optional*) – Minimum number of additional data points with the highest values of x to return. Has to be used with return_additional=True. If set to N, then at least N additional data points will always be present in the plot, regardless of how the adaptive binning is done. The adaptive binning is stopped at the lowest value of x among the additional data points so that these points and the mean line do not overlap.

Returns

- **bin_centers** (*unyt.unyt_array*) – The centers of the bins (taken to be the linear mean of the bin edges).
- **means** (*unyt.unyt_array*) – Vertical mean values within the bins.
- **standard_deviation** (*unyt.unyt_array*) – Standard deviation within the bins, to be shown as scatter.
- **additional_x** (*unyt.unyt_array, optional*) – x data points from the bins where the number of data points is smaller than minimum_in_bin
- **additional_y** (*unyt.unyt_array, optional*) – y data points from the bins where the number of data points is smaller than minimum_in_bin

Notes

The return types are such that you can pass this directly to *plt.errorbar*, as follows:

```
plt.errorbar(
    *binned_mean_line(x, y, x_bins, 10)
)
```

velociraptor.tools.lines.**binned_median_line**(*x: unyt.array.unyt_array, y: unyt.array.unyt_array, x_bins: unyt.array.unyt_array, percentiles: List[int] = [16, 84], minimum_in_bin: int = 3, return_additional: bool = False, minimum_additional_points: int = 0*)

Gets a median (y) line, binned in the x direction.

Parameters

- **x** (*unyt.unyt_array*) – Horizontal values, to be binned.
- **y** (*unyt.unyt_array*) – Vertical values, to have the median calculated in the x-bins
- **x_bins** (*unyt.unyt_array*) – Horizontal bin edges. Must have the same units as x.
- **percentiles** (*List[int], optional*) – Percentiles to return as the positive and negative errors. By default these are 16 and 84th percentiles.
- **minimum_in_bin** (*int, optional*) – Minimum number of items in a bin to return that bin. If a bin has fewer values than this, it is excluded from the return values. Default: 3.
- **return_additional** (*bool, optional*) – Boolean. If true, makes the function return x and y data points that lie in the bins where the number of data points is smaller than minimum_in_bin, and any points that are higher than the highest bin edge. Default: false.
- **minimum_additional_points** (*int, optional*) – Minimum number of additional data points with the highest values of x to return. Has to be used with return_additional=True. If set to N, then at least N additional data points will always be present in the plot, regardless of how the adaptive binning is done. The adaptive binning is stopped at the lowest value of x among the additional data points so that these points and the median line do not overlap.

Returns

- **bin_centers** (*unyt.unyt_array*) – The centers of the bins (taken to be the linear mean of the bin edges).
- **medians** (*unyt.unyt_array*) – Vertical median values within the bins.
- **deviations** (*unyt.unyt_array*) – Deviation from median vertically using the `percentiles` defined above. This has shape 2xN.
- **additional_x** (*unyt.unyt_array, optional*) – x data points from the bins where the number of data points is smaller than minimum_in_bin
- **additional_y** (*unyt.unyt_array, optional*) – y data points from the bins where the number of data points is smaller than minimum_in_bin

Notes

The return types are such that you can pass this directly to *plt.errorbar*, as follows:

```
plt.errorbar(
    *binned_median_line(x, y, x_bins, 10)
)
```

velociraptor.tools.luminosity_functions module

Tools for creating luminosity functions!

velociraptor.tools.luminosity_functions.**create_luminosity_function_given_bins**(*luminosities: unyt.array.unyt_ar bins: unyt.array.unyt_ar box_volume: unyt.array.unyt_qu minimum_in_bin: int = 3*)

Creates a luminosity function (with equal width bins magnitudes) for you to plot.

Parameters

- **luminosities** (*unyt.unyt_array*) – The array that you want to create a luminosity function of (usually this is for example the stellar luminosities).
- **bins** (*unyt.unyt_array*) – The magnitude bin edges to use.
- **unyt.unyt_quantity** (*box_volume*) – The volume of the box such that we can return `n / volume`.
- **minimum_in_bin** (*int, optional*) – The number of objects in a bin for it to be classed as valid. Bins with a number of objects smaller than this are not returned. By default this parameter takes a value of 3.

Returns

- **bin_centers** (*unyt.unyt_array*) – The centers of the bins (taken to be the linear mean of the bin edges).
- **luminosity_function** (*unyt.unyt_array*) – The value of the luminosity function at the bin centers.
- **error** (*unyt.unyt_array*) – Scatter in the luminosity function (Poisson errors).

velociraptor.tools.luminosity_functions.**create_luminosity_function**(*luminosities: unyt.array.unyt_array, lowest_magnitude: unyt.array.unyt_quantity, highest_magnitude: unyt.array.unyt_quantity, box_volume: unyt.array.unyt_quantity, n_bins: int = 25, minimum_in_bin: int = 3, return_bin_edges: bool = False*)

Creates a luminosity function (with equal width bins in log M) for you to plot.

Parameters

- **luminosities** (*unyt.unyt_array*) – The array that you want to create a luminosity function of (usually this is for example stellar luminosities).
- **lowest_magnitude** (*unyt.unyt_quantity*) – the lowest magnitude edge of the bins
- **highest_magnitude** (*unyt.unyt_quantity*) – the highest magnitude edge of the bins
- **box_volume** (*unyt.unyt_quantity*) – The volume of the box such that we can return `n / volume`.
- **n_bins** (*unyt.unyt_array*) – The number of equal log-width bins across the range to use.
- **minimum_in_bin** (*int, optional*) – The number of objects in a bin for it to be classed as valid. Bins with a number of objects smaller than this are not returned. By default this parameter takes a value of 3.
- **return_bin_edges** (*bool, optional*) – Return the bin edges used in the binning process? Default is False.

Returns

- **bin_centers** (*unyt.unyt_array*) – The centers of the bins (taken to be the linear mean of the bin edges).
- **luminosity_function** (*unyt.unyt_array*) – The value of the luminosity function at the bin centers.
- **error** (*unyt.unyt_array*) – Scatter in the luminosity function (Poisson errors).
- **bin_edges** (*unyt.unyt_array, optional*) – Bin edges that were used in the binning process.

velociraptor.tools.mass_functions module

Tools for creating mass functions!

velociraptor.tools.mass_functions.**create_mass_function_given_bins**(*masses: unyt.array.unyt_array*, *bins: unyt.array.unyt_array*, *box_volume: unyt.array.unyt_quantity*, *minimum_in_bin: int = 3*)

Creates a mass function (with equal width bins in log M) for you to plot.

Parameters

- **masses** (*unyt.unyt_array*) – The array that you want to create a mass function of (usually this is for example halo masses or stellar masses).
- **bins** (*unyt.unyt_array*) – The mass bin edges to use.
- **unyt.unyt_quantity** (*box_volume*) – The volume of the box such that we can return `n / volume`.
- **minimum_in_bin** (*int, optional*) – The number of objects in a bin for it to be classed as valid. Bins with a number of objects smaller than this are not returned. By default this parameter takes a value of 3.

Returns

- **bin_centers** (*unyt.unyt_array*) – The centers of the bins (taken to be the linear mean of the bin edges).
- **mass_function** (*unyt.unyt_array*) – The value of the mass function at the bin centers.
- **error** (*unyt.unyt_array*) – Scatter in the mass function (Poisson errors).

velociraptor.tools.mass_functions.**create_mass_function**(*masses: unyt.array.unyt_array*, *lowest_mass: unyt.array.unyt_quantity*, *highest_mass: unyt.array.unyt_quantity*, *box_volume: unyt.array.unyt_quantity*, *n_bins: int = 25*, *minimum_in_bin: int = 3*, *return_bin_edges: bool = False*)

Creates a mass function (with equal width bins in log M) for you to plot.

Parameters

- **masses** (*unyt.unyt_array*) – The array that you want to create a mass function of (usually this is for example halo masses or stellar masses).
- **lowest_mass** (*unyt.unyt_quantity*) – the lowest mass edge of the bins
- **highest_mass** (*unyt.unyt_quantity*) – the highest mass edge of the bins
- **box_volume** (*unyt.unyt_quantity*) – The volume of the box such that we can return `n / volume`.
- **n_bins** (*unyt.unyt_array*) – The number of equal log-width bins across the range to use.
- **minimum_in_bin** (*int, optional*) – The number of objects in a bin for it to be classed as valid. Bins with a number of objects smaller than this are not returned. By default this parameter takes a value of 3.
- **return_bin_edges** (*bool, optional*) – Return the bin edges used in the binning process? Default is False.

Returns

- **bin_centers** (*unyt.unyt_array*) – The centers of the bins (taken to be the linear mean of the bin edges).
- **mass_function** (*unyt.unyt_array*) – The value of the mass function at the bin centers.
- **error** (*unyt.unyt_array*) – Scatter in the mass function (Poisson errors).
- **bin_edges** (*unyt.unyt_array, optional*) – Bin edges that were used in the binning process.

velociraptor.tools.mass_functions.**create_adaptive_mass_function**(*masses: unyt.array.unyt_array*, *lowest_mass: unyt.array.unyt_quantity*, *highest_mass: unyt.array.unyt_quantity*, *box_volume: unyt.array.unyt_quantity*, *base_n_bins: int = 25*, *minimum_in_bin: int = 3*, *return_bin_edges: bool = False*)

Creates a mass function (with equal width bins in log M) for you to plot.

Parameters

- **masses** (*unyt.unyt_array*) – The array that you want to create a mass function of (usually this is for example halo masses or stellar masses).
- **lowest_mass** (*unyt.unyt_quantity*) – the lowest mass edge of the bins
- **highest_mass** (*unyt.unyt_quantity*) – the highest mass edge of the bins
- **box_volume** (*unyt.unyt_quantity*) – The volume of the box such that we can return `n / volume`.
- **base_n_bins** (*unyt.unyt_array*) – The number of equal log-width bins across the range to use in the case where no adaptive sampling is required. This returns the minimal allowed bin width.
- **minimum_in_bin** (*int, optional*) – The number of objects in a bin for it to be classed as valid. Bins with a number of objects smaller than this are not returned. By default this parameter takes a value of 3.
- **return_bin_edges** (*bool, optional*) – Return the bin edges used in the binning process? Default is False.

Returns

- **bin_centers** (*unyt.unyt_array*) – The centers of the bins (taken to be the linear mean of the bin edges).
- **mass_function** (*unyt.unyt_array*) – The value of the mass function at the bin centers.
- **error** (*unyt.unyt_array*) – Scatter in the mass function (Poisson errors).
- **bin_edges** (*unyt.unyt_array, optional*) – Bin edges that were used in the binning process.

7.1.2 Submodules

velociraptor.exceptions module

Manually defined exceptions.

exception velociraptor.exceptions.**RegistrationDoesNotMatchError**(*message=''*)

Bases: `Exception`

Raised when our registration function does not match the current variable name passed to it.

exception velociraptor.exceptions.**AutoPlotterError**(*message*)
Bases: Exception

exception velociraptor.exceptions.**ObservationalDataError**(*message*)
Bases: Exception

velociraptor.regex module

Caching functions for regexes.

velociraptor.regex.**cached_regex**

velociraptor.units module

All classes that are involved in the handling of the units, including the master VelociraptorUnits class.

class velociraptor.units.**VelociraptorUnits**(*filename: str*, *disregard_units: bool = False*)
Bases: object

Generates a unyt system that can then be used with the velociraptor data.

You are probably looking for the following attributes:

- VelociraptorUnits.length
- VelociraptorUnits.mass
- VelociraptorUnits.metallicity (relative to solar)
- VelociraptorUnits.age
- VelociraptorUnits.velocity
- VelociraptorUnits.star_formation_rate

This will allow you to extract variables in the correct units. This object also holds the current scale factor and redshift through the a and z variables. Finally, it contains whether or not the current unit system is comoving (VelociraptorUnits.comoving) and whether or not the underlying simulation was cosmological (VelociraptorUnits.cosmological).

get_unit_dictionary()
Gets the unit library from the header information in the file. These are a mix of units, so we just read them all – and allow the people who define the registration functions to figure out how to use them.

load_cosmology(*handle: h5py._hl.files.File*) → astropy.cosmology.core.Cosmology
Save the (astropy) cosmology to a HDF5 dataset.

Parameters **handle** (*h5py.File*) – h5py file handle for the catalogue file.

Returns Astropy cosmology instance extracted from the HDF5 file. Also sets self.cosmology.

Return type astropy.cosmology.Cosmology

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

v

Index

D

E

F

G

S

T

V

W

X

Y