galaxy-chop

Release 0.2

Valeria Cristiani

Dec 15, 2021

- USER GUIDE

GalaxyChop is a Python package that tackles the dynamical decomposition problem by using clustering techniques in phase space for stellar galactic components.

It runs in numerical N-body simulations populated with semi-analytical models and full hydrodynamical simulations, such as Illustris TNG and EAGLE.

CHAPTER

ONE

MOTIVATION

Galaxies are self-gravitating complex stellar systems formed mainly by stars, dark matter, gas and dust. Star are assembled in different stellar components, such as the disk (thin and thick), the nucleus, the stellar halo and the bar. The components interact with each other and each of them follows its own temporal evolution. For this reason, the description of the formation and evolution of galaxies is strongly linked to the formation and evolution of each of these individual components and their assemble in the final galaxy.

Dynamical decomposition it is a fundamental tool to separate each galaxy component for further study. Numerous methods exist in the literature to perform this task, but there is no tool that allows us to use several of them, providing the possibility of an easy comparison.

CHAPTER

TWO

DYNAMIC DECOMPOSITION MODELS IMPLEMENTED

- **JHistogram:** Implementation of the dynamic decomposition model of galaxies described by Abadi et al.(2003).
- **JThreshold:** Implementation of the dynamic decomposition model of galaxies used in Tissera et al.(2012), Vogelsberger et al.(2014), Marinacci et al.(2014), Park et al.(2019), etc.
- **KMeans:** Implementation of Scikit-Learn K-means as a model for dynamical decomposing of galaxies.
- **GaussianMixture:** Implementation of the dynamic decomposition model of galaxies described by Obreja et al.(2018).
- **AutoGaussianMixture:** Implementation of the dynamic decomposition model of galaxies described by Du et al.(2019)

And many more

CHAPTER

THREE

REQUIREMENTS

You need Python `3.7`, `3.8`, `3.9` to run GalaxyChop.

3.1 Standard Installation

You could find **GalaxyChop** at PyPI. The standar instalation via pip:

```
$ pip install galaxychop
```

3.2 Development Install

Clone this repo and then inside the local directory execute

```
$ git clone https://github.com/vcristiani/galaxy-chop.git
$ cd galaxy-chop
$ pip install -e .
```

CHAPTER

FOUR

AUTHORS

- Valeria Cristiani valeria.cristiani@unc.edu.ar (IATE-OAC-CONICET, FaMAF-UNC).
- Antonela Taverna (IATE-OAC-CONICET).
- Juan Cabral (IATE-OAC-CONICET, CONAE).
- Rafael Pignata (OAC-CONICET, FaMAF-UNC).
- Bruno Sanchez (Duke University).
- Martin Chalela (IATE-OAC-CONICET).
- Federico Benelli (IPQA-CONICET, FCEFyN-UNC).
- Ornela Marioni (IATE-OAC-CONICET, FaMAF-UNC).
- Nelson Villagra (UNPSJB).

CHAPTER

FIVE

CONTENTS

5.1 Installation

You need Python 3.7, 3.8 or 3.9 to run *GalaxyChop*.

5.1.1 Installing with pip

Make sure that the Python interpreter can load *GalaxyChop* code. The most convenient way to do this is to use virtualenv, virtualenvwrapper, and pip.

After setting up and activating the virtualenv, run the following command:

```
$ pip install galaxychop
```

That should be it all.

5.1.2 Installing the development version

If you'd like to be able to update your *GalaxyChop* code occasionally with the latest improvements and bug fixes, follow these instructions:

Make sure that you have Git installed and that you can run its commands from a shell. (Enter *git help* at a shell prompt to test this.)

Check out *GalaxyChop* main development branch like so:

```
$ git clone https://github.com/vcristiani/galaxy-chop.git
```

This will create a directory *galaxy-chop* in your current directory.

Then you can proceed to install with the commands

```
$ cd galaxy-chop
$ pip install -e .
```

5.2 Tutorials

The following page contains a collection of tutorials, practical examples and problems solved with Galaxychop.

From galaxy representation, to galaxy decompositions or even plotting of the components.

In addition, the user might come across tutorials related on how to use and customize the decompositions and plotting utilities, insight on the API capabilities and many other topics related to dynamic decomposition.

5.2.1 Quickstart

This chapter aims to explain how to use GalaxyChop in general. Here you can see a simple example of

- Reading a galaxy from a file HDF5.
- Explore the attributes of the galaxy.
- Some graphics.
- Aligning and centering the galaxies.
- A simple decomposition.

1. Conceptual overview

By a design decision, and after several prototypes, GalaxyChop was designed in two layers

1. A data representation level, where we defined a class `Galaxy` which encapsulates all the physical properties useful for the dynamic decomposition.
2. A `models` level, which understands the `Galaxy` instances and returns an instance of type `Components` which represents "to which part of the galaxy each stellar particle corresponds to".

This first tutorial aims only to give an overview of the project and not to go into too specific details.

Note: This tutorial is not intended to be a scientific explanation of why and how to perform a dynamic decomposition of galaxies, only to explain the functionalities of GalaxyChop.

---

2. Creating your first Galaxy

GalaxyChop represents galaxies with a class called `galaxychop.Galaxy`, which is a collection of *gas*, *dark matter* and *stars* particles each characterized by 3 positions, three velocities, masses and various energies.

To create a `Galaxy` type object, there are several alternatives, the most useful being to read all the data from a file type HDF5.

So first of all it is necessary to obtain a file with the appropriate content. It is recommended to download this file gal394242.h5 and place it where it is convenient for you.

Note: For convenience we have placed the file *gal394242.h5* in the same folder where this tutorial is located.

As a first step it is necessary to import the *GalaxChop* library, and by simple habit we have taken the practice of assigning to the library the alias `gchop`.

[1]:
```
import galaxychop as gchop
```

Next we load the file *gal394242.h5* using the `read_hdf5` function. This will generate an instance of the `gchop.Galaxy` class.

[2]:
```
gal = gchop.read_hdf5("gal394242.h5")
gal
```

[2]:
```
Galaxy(stars=32067, dark_matter=21156, gas=4061, potential=True)
```

3. Galaxy attributes

Galaxies internally consist of **three sets of particles** , one for `stars`, one for `dark_matter` and one for `gas`.

All these sets can be accessed, and expose a set of physical properties

For example, if we wanted to access the positions (`x`, `y`, and `z`) of the stellar particles:

[3]:
```
gal.stars.x, gal.stars.y, gal.stars.z
```

[3]:
```
(<Quantity [ 0.12017286, -0.01992682, -0.09766993, ..., 15.06957949,
            17.21394372,  9.28556454] kpc>,
 <Quantity [ 1.51520602e-03,  3.25905789e-02, -2.27875536e-02, ...,
             4.52400820e+00, -1.96094867e-01,  1.15665035e+01] kpc>,
 <Quantity [-3.73908514e-02,  5.12012087e-03,  4.66359649e-02, ...,
             1.15501605e+01, -1.40433638e+01,  7.57282498e+00] kpc>)
```

It can be observed that the positions have units which can be bypassed using the `arr_` accessor.

For example if we would like to access the `gas` masses (`m`) but without units:

[4]:
```
gal.gas.arr_.m
```

[4]:
```
array([2579202.70087197, 1368584.63520184, 1392260.60127839, ...,
        990849.80320185, 1456959.43501778, 1760535.66974588])
```

The galaxy instances, on the other hand, have useful methods to summarize the data from the three sets, such as summarize the data coming from the three sets, such as `angular_momentum_` or `gal.kinetic_energy_`.

Finally both the arrays and the galaxy in general have methods to create Pandas DataFrames with the particle data.

[5]:
```
gal.to_dataframe()
```

[5]:

	ptype	ptypev	m	x	y	z	vx	\
0	stars	0	5.224283e+05	0.120173	0.001515	-0.037391	10.773575	
1	stars	0	9.745897e+05	-0.019927	0.032591	0.005120	20.282349	
2	stars	0	6.935776e+05	-0.097670	-0.022788	0.046636	-14.897980	
3	stars	0	1.070959e+06	-0.007224	-0.138971	0.176169	-8.665253	
4	stars	0	6.013803e+05	0.095277	0.001167	0.100484	23.508469	
...	...	...	...	...	...	...	...	
57279	gas	2	1.400046e+06	2.415113	-8.443658	-0.427454	117.975723	
57280	gas	2	1.375867e+06	-4.756847	5.970680	-4.109694	-136.718105	
57281	gas	2	9.908498e+05	-9.376904	-1.965296	-0.563573	-188.311616	
57282	gas	2	1.456959e+06	-6.880090	0.314529	-6.284694	-261.390251	
57283	gas	2	1.760536e+06	-7.371887	2.670409	-6.167000	-282.899384	

(continues on next page)

(continued from previous page)

```
               vy          vz  softening      potential  kinetic_energy  \
0       -6.878906  -20.425400         0.0 -195699.620206      290.293111
1        8.661957   -7.947495         0.0 -196176.962277      274.782915
2        6.957092  -10.818886         0.0 -195152.120168      193.699612
3       -4.337433    5.506927         0.0 -194695.767625       62.113089
4       -7.842865   -3.754723         0.0 -195703.942688      314.128285
...           ...         ...         ...            ...             ...
57279 -243.209320   31.390686         0.0 -129252.826165    37027.209909
57280  232.501099  -56.024063         0.0 -129835.893336    37943.648417
57281   31.467773  179.615128         0.0 -132966.102621    34356.539847
57282   69.243408 -137.456593         0.0 -132205.106762    46006.913910
57283  138.414307 -134.297680         0.0 -127609.955549    58613.224160

        total_energy          Jx           Jy          Jz
0     -195409.327095   -0.288157     2.051746   -0.842982
1     -195902.179362   -0.303364    -0.054520   -0.833619
2     -194958.420557   -0.077915    -1.751461   -1.018987
3     -194633.654536   -0.001182    -1.486768   -1.172886
4     -195389.814402    0.783702     2.719973   -0.774688
...              ...         ...          ...         ...
57279  -92225.616256 -369.012963  -126.241204  408.768823
57280  -91892.244919  621.006707   295.371727 -289.672123
57281  -98609.562774 -335.262575  1790.361085 -665.158420
57282  -86198.192853  391.939479   697.043885 -394.186002
57283  -68996.731389  494.971200   754.613090 -264.917457

[57284 rows x 16 columns]
```

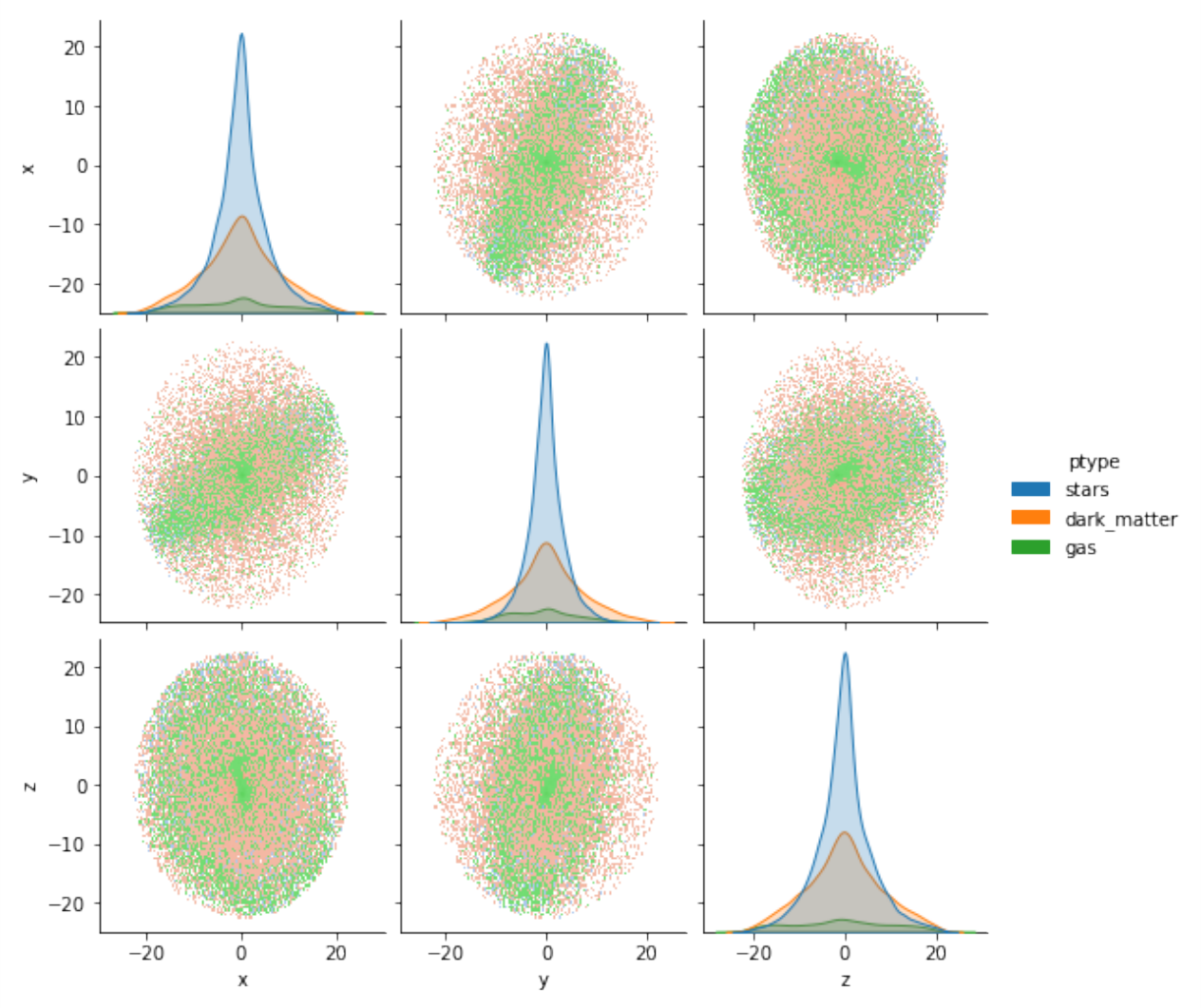
4. Visualizing the galaxy

Galaxies have a set of utilities to visualize on the one hand all the particles in the real space; and on the other hand only the stellar ones in the circularity space.

Thus a good summary plot of a galaxy can be made with the command:

[6]:
```
gal.plot()
```

[6]:
```
<seaborn.axisgrid.PairGrid at 0x7f288e98baf0>
```



At this point we observe in the graph that the galaxy is neither centered nor aligned, this can be validated with:

[7]:
```
print("Centered:", gchop.is_centered(gal))
print("Aligned:", gchop.is_star_aligned(gal))
```

```
Centered: False
Aligned: False
```

To correct this GalaxyChop has the `center()` and `star_align()` functions which can be composed as follows

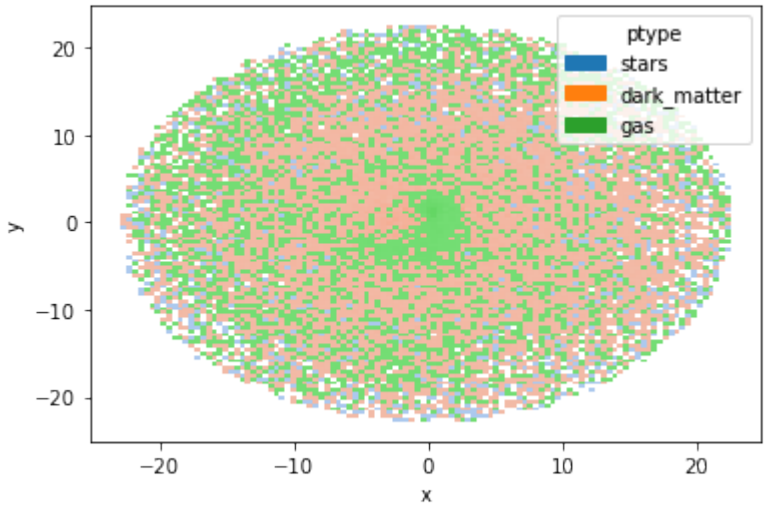
[8]:
```
gal = gchop.star_align(gchop.center(gal))
print("Centered:", gchop.is_centered(gal))
print("Aligned:", gchop.is_star_aligned(gal))
```

```
Centered: True
Aligned: True
```

We can also visualize the result, for example by comparing the positions at `x` and `y`, highlighting according to the particle type.

```
[9]: gal.plot.hist("x", "y", labels="ptype")
```

```
[9]: <AxesSubplot:xlabel='x', ylabel='y'>
```



5. Chop the galaxy

Finally, let's decompose a galaxy into its parts. The package offers several decomposition models as well as a framework to create new methods. methods.

For didactic reasons we will stick to the method presented in the work of Abadi et al.(2003), which in the context of GalaxyChop is called `JHistogram`.

The analysis can be summarized in the following steps

1. Instantiate the decomposer

```
[10]: decomp = gchop.models.JThreshold()
```

2. Decompose the galaxy into components

```
[11]: components = decomp.decompose(gal)
components
```

```
[11]: Components(57284, labels=[ 0.  1. nan], probabilities=False)
```

As you can see by the labels `[0 1 nan]` the decomposer found two components (`0` and `1`) and could not classify at least one particle, so the meaning of `nan` for Galaxychop can be understood as "I don't know what to do with this particle".

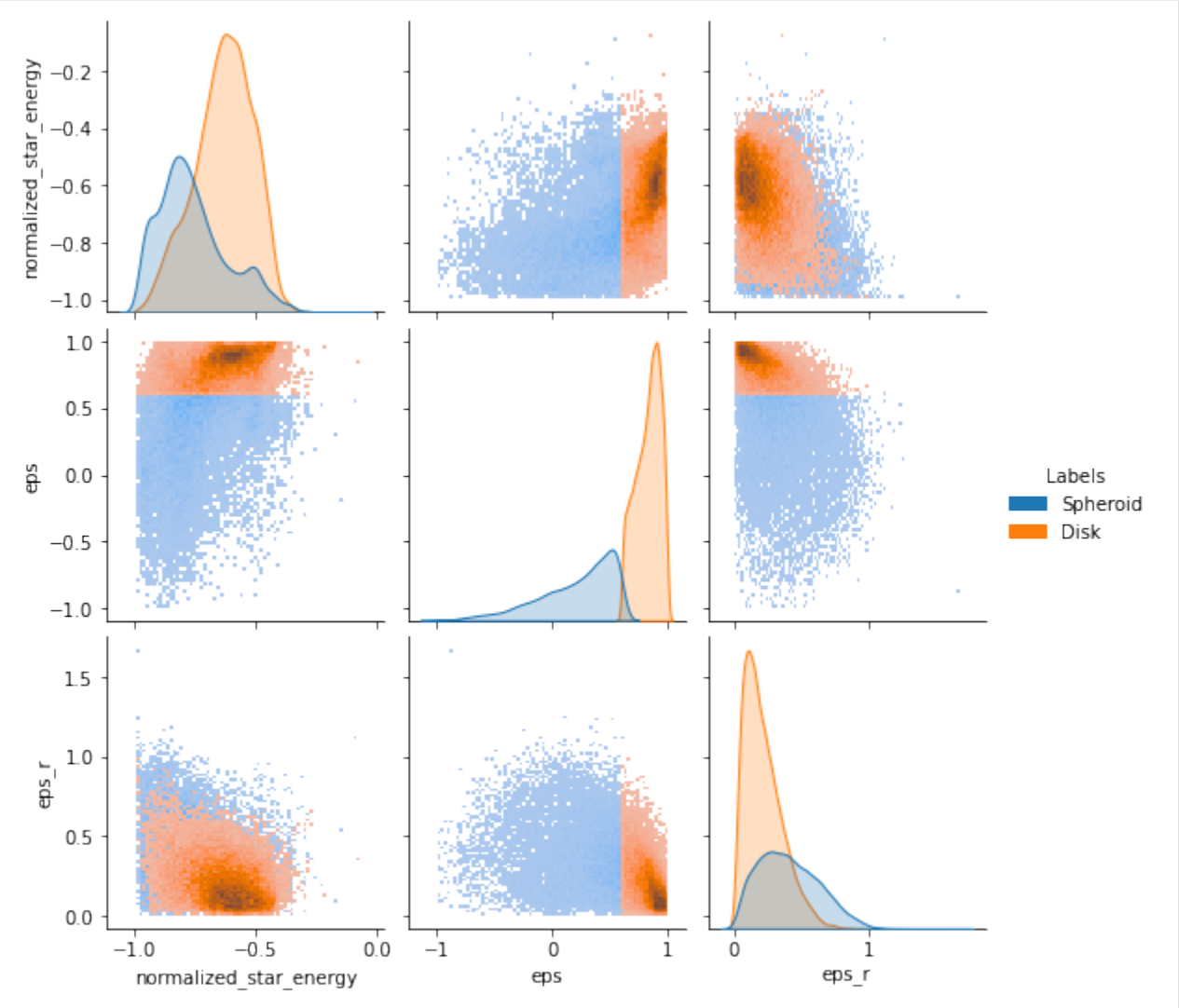
Another detail to take into account is that GalaxyChop decomposes galaxies, but since the dynamic decomposition for `gas` and `dark matter` is not defined, it assigns the label `nan` to all particles.

Finally, it is seen that this classifier does not provide probabilities of particle belonging to a label as it does particle to a label as does `GaussianMixture`.

3. Now we can see the division of the particles in the circularity space.

```
[12]: gal.plot.circ_pairplot(labels=components, lmap={0: "Spheroid", 1: "Disk"})
```

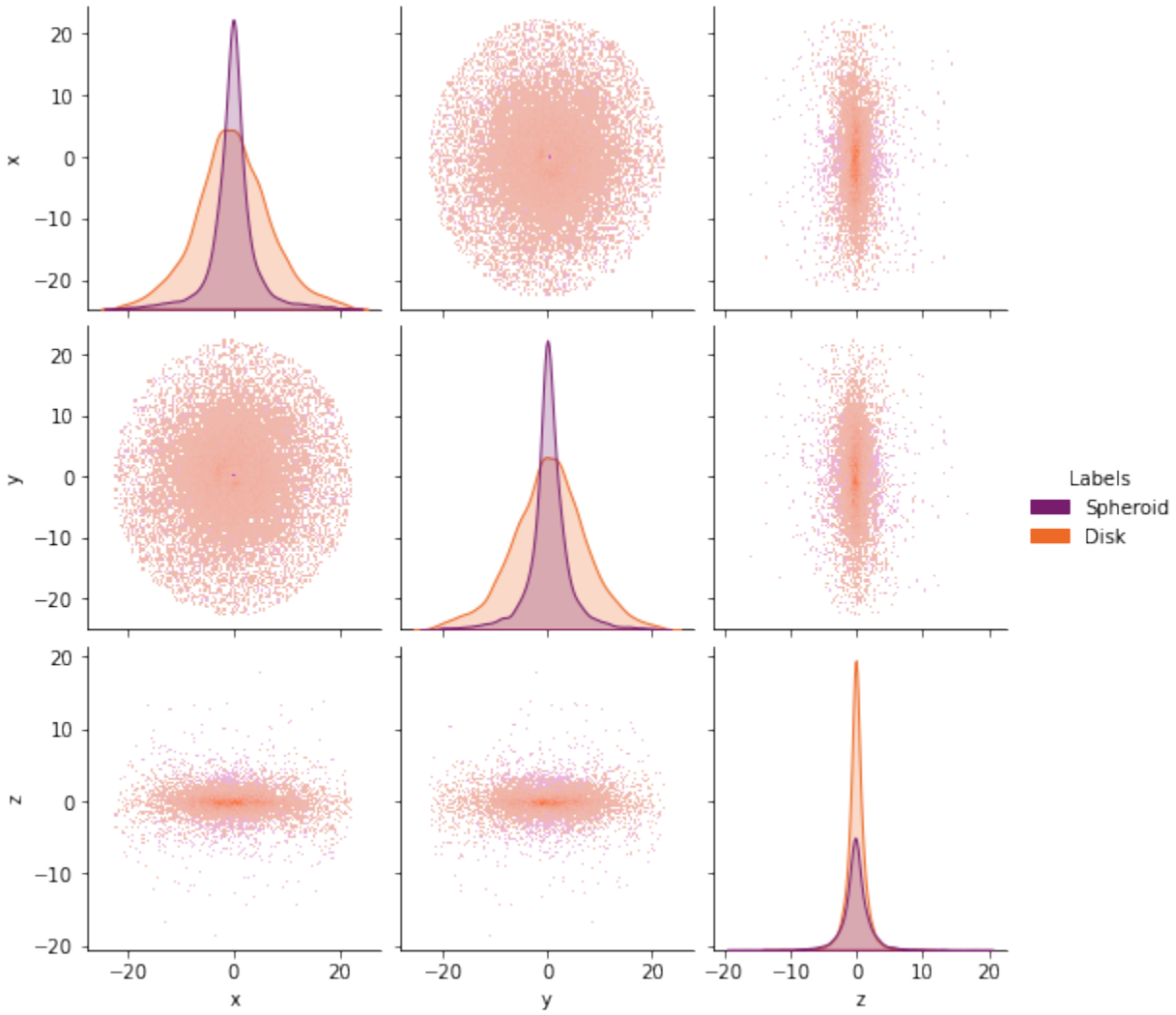
```
[12]: <seaborn.axisgrid.PairGrid at 0x7f288576a7c0>
```



or in the real space

[13]:
```
gal.plot.pairplot(
    labels=components,
    lmap={0: "Spheroid", 1: "Disk"},
    palette="inferno",
)
```

[13]:
```
<seaborn.axisgrid.PairGrid at 0x7f287e964fa0>
```



Finally you can convert the entire components into a `Dataframe`

[14]:
```
components.to_dataframe()
```

[14]:
```
       labels ptypes
0         0.0  stars
1         0.0  stars
2         0.0  stars
3         0.0  stars
4         0.0  stars
...       ...    ...
57279     NaN    gas
57280     NaN    gas
57281     NaN    gas
57282     NaN    gas
57283     NaN    gas

[57284 rows x 2 columns]
```

5.2.2 Understanding the GalaxyChop Galaxies

TODO

5.2.3 Galaxy plotting capabilities

TODO

5.2.4 Custom decomposers

TODO

5.3 galaxychop package

GalaxyChop.

Implementation of a few galaxy dynamic decomposition methods.

5.3.1 galaxychop.data module

Module galaxy-chop.

class galaxychop.data.**ParticleSetType**(*value*)

Bases: enum.IntEnum

Name of the particle type.

Name and number that are used to describe the particle type in the ParticleSet class.

classmethod mktype(*v*)

Create a ParticleSetType from name, or value.

humanize()

Particle type name in lower case.

class galaxychop.data.**ParticleSet**(*ptype*, *m*, *x*, *y*, *z*, *vx*, *vy*, *vz*, *potential*, *softening*)

Bases: object

ParticleSet class.

Creates a set particles of a particular type (stars, dark matter or gas) using masses, positions, velocities and potential energy.

Parameters

- **ptype** (ParticleSetType) – Indicates if this set corresponds to stars, dark matter or gas.
- **m** (*Quantity*) – Particle masses. Shape: (n,1). Default unit: M_sun
- **x** (*Quantity*) – Positions. Shapes: (n,1). Default unit: kpc.
- **y** (*Quantity*) – Positions. Shapes: (n,1). Default unit: kpc.
- **z** (*Quantity*) – Positions. Shapes: (n,1). Default unit: kpc.
- **vx** (*Quantity*) – Velocities. Shapes: (n,1). Default unit: km/s.
- **vy** (*Quantity*) – Velocities. Shapes: (n,1). Default unit: km/s.

- **vz** (*Quantity*) – Velocities. Shapes: (n,1). Default unit: km/s.
- **potential** (*Quantity, default value = 0*) – Specific potential energy of particles. Shape: (n,1). Default unit: (km/s)**2.
- **softening** (*Quantity, default value = 0*) – Softening radius of particles. Shape: (1,). Default unit: kpc.
- **kinetic_energy** (*Quantity*) – Specific kinetic energy of particles. Shape: (n,1). Default unit: (km/s)**2.
- **total_energy** (*Quantity*) – Specific total energy of particles. Shape: (n,1). Default unit: (km/s)**2.
- **Jx** (*Quantity*) – Components of angular momentum of particles. Shapes: (n,1). Default units: kpc*km/s.
- **Jy** (*Quantity*) – Components of angular momentum of particles. Shapes: (n,1). Default units: kpc*km/s.
- **Jz** (*Quantity*) – Components of angular momentum of particles. Shapes: (n,1). Default units: kpc*km/s.
- **has_potential** (*bool.*) – Indicates if the specific potential energy is computed.
- **arr** (Instances of `ArrayAccessor`) – Access to the attributes (defined with uttrs) of the provided instance, and if they are of atropy.units.Quantity type it converts them into numpy.ndarray.

property angular_momentum_
Components of specific angular momentum in units of kpc*km/s.

to_dataframe(*attributes=None*)
Convert to pandas data frame.

This method constructs a data frame with the particles and parameters of `ParticleSet class`.

Parameters **attributes** (*tuple, default value = None*) – Dictionary keys of ParticleSet parameters used to create the data frame. If it's None, the data frame is constructed from all the parameters of the `ParticleSet class`.

Returns **DataFrame** – Data frame of the particles with the selected parameters.

Return type pandas data frame

class galaxychop.data.**Galaxy**(*stars*, *dark_matter*, *gas*)
Bases: `object`

Galaxy class.

Builds a galaxy object from a `ParticleSet` for each type of particle (stars, dark matter and gas).

Parameters

- **stars** (`ParticleSet`) – Instance of `ParticleSet` with stars particles.
- **dark_matter** (`ParticleSet`) – Instance of `ParticleSet` with dark matter particles.
- **gas** (`ParticleSet`) – Instance of ParticleSet with gas particles.

has_potential_
Indicates if this Galaxy instance has the potential energy computed.

Type bool

to_dataframe(*, *ptypes=None*, *attributes=None*)
Convert to pandas data frame.

This method builds a data frame from the particles of the Galaxy.

Parameters

- **ptypes** (*tuple, default value = None*) – Strings indicating the ParticleSetType to include. If it's None, all particle types are included.
- **attributes** (*tuple, default value = None*) – Dictionary keys of ParticleSet parameters used to create the data frame. If it's None, the data frame is constructed from all the parameters of the `ParticleSet class`.

Returns **DataFrame** – Data frame of Galaxy with selected particles and parameters.

Return type pandas data frame

to_hdf5(*path_or_stream*, *metadata=None*, ***kwargs*)
Shortcut to `galaxychop.io.to_hdf5()`.

It is responsible for storing a galaxy in HDF5 format. The procedure only stores the attributes `m`, `x`, `y`, `z`, `vx`, `vy` and `vz`, since all the other attributes can be derived from these, and the `softenings` can be arbitrarily changed at the galaxy creation/reading process

Parameters

- **path_or_stream** (*str or file like.*) – Path or file like objet to the h5 to store the galaxy.
- **metadata** (*dict or None (default None)*) – Extra metadata to store in the h5 file.
- **kwargs** – Extra arguments to the function `astropy.io.misc.hdf5.write_table_hdf5()`

property plot
Plot accessor.

property kinetic_energy_
Specific kinetic energy of stars, dark matter and gas particles.

Returns **tuple** – (k_s, k_dm, k_g): Specific kinetic energy of stars, dark matter and gas respectively. Shape(n_s, n_dm, n_g). Unit: (km/s)**2

Return type Quantity

Examples

This returns the specific kinetic energy of stars, dark matter and gas particles respectively.

```
>>> import galaxychop as gchop
>>> galaxy = gchop.Galaxy(...)
>>> k_s, k_dm, k_g = galaxy.kinetic_energy_
```

property potential_energy_
Specific potential energy of stars, dark matter and gas particles.

This property doesn't compute the potential energy, only returns its value if it is already computed, i.e. `has_potential_` is True. To compute the potential use the `galaxychop.potential` function.

Returns **tuple** – (p_s, p_dm, p_g): Specific potential energy of stars, dark matter and gas respectively. Shape(n_s, n_dm, n_g). Unit: (km/s)**2

Return type Quantity

Examples

This returns the specific potential energy of stars, dark matter and gas particles respectively.

```
>>> import galaxychop as gchop
>>> galaxy = gchop.Galaxy(...)
>>> galaxy_with_potential = gchop.potential(galaxy)
>>> p_s, p_dm, p_g = galaxy_with_potential.potential_energy_
```

property total_energy_

Specific total energy calculation.

Calculates the specific total energy of dark matter, star and gas particles.

Returns **tuple** – (Etot_s, Etot_dm, Etot_g): Specific total energy of stars, dark matter and gas respectively. Shape(n_s, n_dm, n_g). Unit: (km/s)**2

Return type Quantity

Examples

This returns the specific total energy of stars, dark matter and gas particles respectively.

```
>>> import galaxychop as gchop
>>> galaxy = gchop.Galaxy(...)
>>> E_s, E_dm, E_g = galaxy.total_energy_
```

property angular_momentum_

Specific angular momentum calculation.

Compute the specific angular momentum of stars, dark matter and gas particles.

Returns **tuple** – (J_s, J_dm, J_g): Specific angular momentum of stars, dark matter and gas respectively. Shape(n_s, n_dm, n_g). Unit: (kpc * km / s)

Return type *Quantity*

Examples

This returns the specific angular momentum of stars, dark matter and gas particles respectively.

```
>>> import galaxychop as gchop
>>> galaxy = gchop.Galaxy(...)
>>> J_s, J_dm, J_g = galaxy.angular_momentum_
```

galaxychop.data.**galaxy_as_kwargs**(*galaxy*)

Galaxy init attributes as dictionary.

Parameters **galaxy** (Galaxy) – Instance of Galaxy.

Returns **kwargs** – Dictionary with `galaxy` attributes.

Return type dict

galaxychop.data.**mkgalaxy**(*m_s: numpy.ndarray*, *x_s: numpy.ndarray*, *y_s: numpy.ndarray*, *z_s: numpy.ndarray*, *vx_s: numpy.ndarray*, *vy_s: numpy.ndarray*, *vz_s: numpy.ndarray*, *m_dm: numpy.ndarray*, *x_dm: numpy.ndarray*, *y_dm: numpy.ndarray*, *z_dm: numpy.ndarray*, *vx_dm: numpy.ndarray*, *vy_dm: numpy.ndarray*, *vz_dm: numpy.ndarray*, *m_g: numpy.ndarray*, *x_g: numpy.ndarray*, *y_g: numpy.ndarray*, *z_g: numpy.ndarray*, *vx_g: numpy.ndarray*, *vy_g: numpy.ndarray*, *vz_g: numpy.ndarray*, *softening_s: float = 0.0*, *softening_dm: float = 0.0*, *softening_g: float = 0.0*, *potential_s: Optional[numpy.ndarray] = None*, *potential_dm: Optional[numpy.ndarray] = None*, *potential_g: Optional[numpy.ndarray] = None*)

Galaxy builder.

This function builds a galaxy object from a star, dark matter and gas ParticleSet.

Parameters

- **m_s** (*np.ndarray*) – Star masses. Shape: (n,1).
- **x_s** (*np.ndarray*) – Star positions. Shapes: (n,1).
- **y_s** (*np.ndarray*) – Star positions. Shapes: (n,1).
- **z_s** (*np.ndarray*) – Star positions. Shapes: (n,1).
- **vx_s** (*np.ndarray*) – Star velocities. Shape: (n,1).
- **vy_s** (*np.ndarray*) – Star velocities. Shape: (n,1).
- **vz_s** (*np.ndarray*) – Star velocities. Shape: (n,1).
- **m_dm** (*np.ndarray*) – Dark matter masses. Shape: (n,1).
- **x_dm** (*np.ndarray*) – Dark matter positions. Shapes: (n,1).
- **y_dm** (*np.ndarray*) – Dark matter positions. Shapes: (n,1).
- **z_dm** (*np.ndarray*) – Dark matter positions. Shapes: (n,1).
- **vx_dm** (*np.ndarray*) – Dark matter velocities. Shapes: (n,1).
- **vy_dm** (*np.ndarray*) – Dark matter velocities. Shapes: (n,1).
- **vz_dm** (*np.ndarray*) – Dark matter velocities. Shapes: (n,1).
- **m_g** (*np.ndarray*) – Gas masses. Shape: (n,1).
- **x_g** (*np.ndarray*) – Gas positions. Shapes: (n,1).
- **y_g** (*np.ndarray*) – Gas positions. Shapes: (n,1).
- **z_g** (*np.ndarray*) – Gas positions. Shapes: (n,1).
- **vx_g** (*np.ndarray*) – Gas velocities. Shapes: (n,1).
- **vy_g** (*np.ndarray*) – Gas velocities. Shapes: (n,1).
- **vz_g** (*np.ndarray*) – Gas velocities. Shapes: (n,1).
- **potential_s** (*np.ndarray, default value = None*) – Specific potential energy of star particles. Shape: (n,1).
- **potential_dm** (*np.ndarray, default value = None*) – Specific potential energy of dark matter particles. Shape: (n,1).
- **potential_g** (*np.ndarray, default value = None*) – Specific potential energy of gas particles. Shape: (n,1).

- **softening_s** (*float, default value = 0*) – Softening radius of stellar particles. Shape: (1,).
- **softening_dm** (*float, default value = 0*) – Softening radius of dark matter particles. Shape: (1,).
- **softening_g** (*float, default value = 0*) – Softening radius of gas particles. Shape: (1,).

Returns galaxy

Return type `Galaxy class` object.

5.3.2 `galaxychop.io` module

Module galaxy-chop.

`galaxychop.io.`**read_hdf5**(*path_or_stream*, *softening_s: float = 0.0*, *softening_dm: float = 0.0*, *softening_g: float = 0.0*)

h5py file reader.

Reads the file containing masses, positions, velocities of stellar, dark matter and gas particles, and constructs a galaxy object. The file may include particle potentials. The softening value can be included.

Parameters

- **path_or_stream** (*str or file-like*) – Path to the h5 file containing the properties of the galaxy particles.
- **softening_s** (*float, default value = 0*) – Softening radius of star particles.
- **softening_dm** (*float, default value = 0*) – Softening radius of dark matter particles.
- **softening_g** (*float, default value = 0*) – Softening radius of gas particles.

Returns galaxy

Return type `Galaxy class` object.

`galaxychop.io.`**to_hdf5**(*path_or_stream*, *galaxy*, *metadata=None*, ***kwargs*)

HDF5 file writer.

It is responsible for storing a galaxy in HDF5 format. The procedure only stores the attributes `m`, `x`, `y`, `z`, `vx`, `vy` and `vz`, since all the other attributes can be derived from these, and the `softenings` can be arbitrarily changed at the galaxy creation/reading process

Parameters

- **path_or_stream** (*str or file-like*) – Path or file like objet to the h5 to store the galaxy.
- **galaxy** (`galaxychop.data.Galaxy`) – The galaxy to store.
- **metadata** (*dict or None (default None)*) – Extra metadata to store in the h5 file.
- **kwargs** – Extra arguments to the function `astropy.io.misc.hdf5.write_table_hdf5()`

`galaxychop.io.`**read_npy**(*path_or_stream_star*, *path_or_stream_dark*, *path_or_stream_gas*, *columns*, *path_or_stream_pot_s=None*, *path_or_stream_pot_dm=None*, *path_or_stream_pot_g=None*, *softening_s: float = 0.0*, *softening_dm: float = 0.0*, *softening_g: float = 0.0*)

Npy file reader.

Reads npy files containing the masses, positions and velocities of stellar particles, dark matter and gas particles, and constructs a galaxy object. Files containing particle potentials can be included. The softening value can be included.

Parameters

- **path_or_stream_star** (*str or file like*) – Path to the npy file containing the properties of the star particles.
- **path_or_stream_dark** (*str or file like*) – Path to the npy file containing the properties of the dark matter particles.
- **path_or_stream_gas** (*str or file like*) – Path to the npy file containing the properties of the gas particles.
- **columns** (*list*) – Specify column names.
- **path_or_stream_pot_s** (*str or file like*) – Path to the npy file containing the potentials of the star particles.
- **path_or_stream_pot_dm** (*str or file like*) – Path to the npy file containing the potentials of the dark matter particles.
- **path_or_stream_pot_g** (*str or file like*) – Path to the npy file containing the potentials of the gas particles.
- **softening_s** (*float, default value = 0*) – Softening radius of star particles.
- **softening_dm** (*float, default value = 0*) – Softening radius of dark matter particles.
- **softening_g** (*float, default value = 0*) – Softening radius of gas particles.

Returns galaxy

Return type `Galaxy class` object.

5.3.3 `galaxychop.plot` module

Plot helper for the galaxy object.

class galaxychop.plot.**GalaxyPlotter**(*galaxy*)

Bases: `object`

Make plots of DecisionMatrix.

get_df_and_hue(*ptypes*, *attributes*, *labels*, *lmap*)

Dataframe and Hue constructor for plot implementations.

Parameters

- **ptypes** (keys of `ParticleSet class` parameters.) – Particle type.
- **attributes** (keys of `ParticleSet class` parameters.) – Names of `ParticleSet class` parameters.
- **labels** (keys of `ParticleSet class` parameters.) – Variable to map plot aspects to different colors.
- **lmap** (*dicts*) – Name assignment to the label.

Returns

- **df** (*pandas.DataFrame*) – DataFrame of galaxy properties with labels added.

- **hue** (keys of `ParticleSet class` parameters.) – Labels of all galaxy particles.

pairplot(*ptypes=None*, *attributes=None*, *labels='ptype'*, *lmap=None*, ***kwargs*)
Draw a pairplot of the galaxy properties.

By default, this function will create a grid of Axes such that each numeric variable in data will by shared across the y-axes across a single row and the x-axes across a single column. The diagonal plots drow a univariate distribution to show the marginal distribution of the data in each column. This function groups the values of all galaxy particles according to some `ParticleSet class` parameter.

Parameters

- **ptypes** (keys of `ParticleSet class` parameters.) – Particle type. Default value = None
- **attributes** (keys of `ParticleSet class` parameters.) – Names of `ParticleSet class` parameters. Default value = None
- **labels** (keys of `ParticleSet class` parameters.) – Variable to map plot aspects to different colors. Default value = None
- **lmap** (*dicts*) – Name assignment to the label. Default value = None
- ****kwargs** – Additional keyword arguments are passed and are documented in `seaborn.pairplot`.

Returns

Return type seaborn.axisgrid.PairGrid

scatter(*x*, *y*, *ptypes=None*, *labels=None*, *lmap=None*, ***kwargs*)
Draw a scatter plot of galaxy properties.

Shows the relationship between x and y. This function groups the values of all galaxy particles according to some `ParticleSet class` parameter.

Parameters

- **x** (keys of `ParticleSet class` parameters.) – Variables that specify positions on the x and y axes. Default value y = None.
- **y** (keys of `ParticleSet class` parameters.) – Variables that specify positions on the x and y axes. Default value y = None.
- **ptypes** (keys of `ParticleSet class` parameters.) – Particle type. Default value = None
- **labels** (keys of `ParticleSet class` parameters.) – Variable to map plot aspects to different colors. Default value = None
- **lmap** (*dicts*) – Name assignment to the label. Default value = None
- ****kwargs** – Additional keyword arguments are passed and are documented in `seaborn.scatterplot`.

Returns

Return type matplotlib.axes.Axes

hist(*x*, *y=None*, *ptypes=None*, *labels=None*, *lmap=None*, ***kwargs*)
Draw a histogram of galaxy properties.

Plot univariate or bivariate histograms to show distributions of datasets. This function groups the values of all galaxy particles according to some `ParticleSet class` parameter.

Parameters

- **x** (keys of `ParticleSet class` parameters.) – Variables that specify positions on the x and y axes. Default value y = None.
- **y** (keys of `ParticleSet class` parameters.) – Variables that specify positions on the x and y axes. Default value y = None.
- **ptypes** (keys of `ParticleSet class` parameters.) – Particle type. Default value = None
- **labels** (keys of `ParticleSet class` parameters.) – Variable to map plot aspects to different colors. Default value = None
- **lmap** (*dicts*) – Name assignment to the label. Default value = None
- ****kwargs** – Additional keyword arguments are passed and are documented in `seaborn.histplot`.

Returns

Return type matplotlib.axes.Axes

kde(*x*, *y=None*, *ptypes=None*, *labels=None*, *lmap=None*, ***kwargs*)

Draw a Kernel Density plot of galaxy properties.

Plot univariate or bivariate distributions using kernel density estimation (KDE). This plot represents the galaxy properties using a continuous probability density curve in one or more dimensions. This function groups the values of all galaxy particles according to some `ParticleSet class` parameter.

Parameters

- **x** (keys of `ParticleSet class` parameters.) – Variables that specify positions on the x and y axes. Default value y = None.
- **y** (keys of `ParticleSet class` parameters.) – Variables that specify positions on the x and y axes. Default value y = None.
- **ptypes** (keys of `ParticleSet class` parameters.) – Particle type. Default value = None
- **labels** (keys of `ParticleSet class` parameters.) – Variable to map plot aspects to different colors. Default value = None
- **lmap** (*dicts*) – Name assignment to the label. Default value = None
- ****kwargs** – Additional keyword arguments are passed and are documented in `seaborn.kdeplot`.

Returns

Return type matplotlib.axes.Axes

get_circ_df_and_hue(*cbins*, *attributes*, *labels*, *lmap*)

Dataframe and Hue constructor for plot implementations.

Parameters

- **cbins** (*tuple*) – It contains the two widths of bins necessary for the calculation of the circular angular momentum. Shape: (2,). Dafult value = (0.05, 0.005).
- **attributes** (keys of `JCirc` tuple.) – Keys of the normalized specific energy, the circularity parameter (J_z/J_circ) and/or the projected circularity parameter (J_p/J_circ) of the stellar particles.
- **labels** (keys of `JCirc` tuple.) – Variable to map plot aspects to different colors.
- **lmap** (*dicts*) – Name assignment to the label.

Returns

- **df** (*pandas.DataFrame*) – DataFrame of the normalized specific energy, the circularity parameter (J_z/J_circ) and/or the projected circularity parameter (J_p/J_circ) of the stellar particles with labels added.
- **hue** (keys of `JCirc` tuple.) – Labels of stellar particles.

`circ_pairplot`(*cbins=(0.05, 0.005)*, *attributes=None*, *labels=None*, *lmap=None*, ***kwargs*)

Draw a pairplot of circularity and normalized energy.

By default, this function will create a grid of Axes such that each numeric variable in data will by shared across the y-axes across a single row and the x-axes across a single column. The diagonal plots drow a univariate distribution to show the marginal distribution of the data in each column. This function groups the values of stellar particles according to some keys of `JCirc` tuple.

Parameters

- **cbins** (*tuple*) – It contains the two widths of bins necessary for the calculation of the circular angular momentum. Shape: (2,). Dafult value = (0.05, 0.005).
- **attributes** (keys of `ParticleSet class` parameters.) – Names of `ParticleSet class` parameters. Default value = None
- **labels** (keys of `JCirc` tuple.) – Variable to map plot aspects to different colors. Default value = None
- **lmap** (*dicts*) – Name assignment to the label. Default value = None
- ****kwargs** – Additional keyword arguments are passed and are documented in `seaborn.pairplot`.

Returns

Return type seaborn.axisgrid.PairGrid

`circ_scatter`(*x*, *y*, *cbins=(0.05, 0.005)*, *labels=None*, *lmap=None*, ***kwargs*)

Draw a scatter plot of circularity and normalized energy.

Shows the relationship between x and y. This function groups the values of stellar particles according to some keys of `JCirc` tuple.

Parameters

- **x** (keys of `JCirc` tuple.) – Variables that specify positions on the x and y axes. Default value y = None.
- **y** (keys of `JCirc` tuple.) – Variables that specify positions on the x and y axes. Default value y = None.
- **cbins** (*tuple*) – It contains the two widths of bins necessary for the calculation of the circular angular momentum. Shape: (2,). Dafult value = (0.05, 0.005).
- **labels** (keys of `JCirc` tuple.) – Variable to map plot aspects to different colors. Default value = None
- **lmap** (*dicts*) – Name assignment to the label. Default value = None
- ****kwargs** – Additional keyword arguments are passed and are documented in `seaborn.scatterplot`.

Returns

Return type matplotlib.axes.Axes

`circ_hist`(*x*, *y=None*, *cbins=(0.05, 0.005)*, *labels=None*, *lmap=None*, ***kwargs*)

Draw a histogram of circularity and normalized energy.

Plot univariate or bivariate histograms to show distributions of datasets. This function groups the values of stellar particles according to some keys of `JCirc` tuple.

Parameters

- **x** (keys of `JCirc` tuple.) – Variables that specify positions on the x and y axes. Default value y = None.
- **y** (keys of `JCirc` tuple.) – Variables that specify positions on the x and y axes. Default value y = None.
- **cbins** (*tuple*) – It contains the two widths of bins necessary for the calculation of the circular angular momentum. Shape: (2,). Dafult value = (0.05, 0.005).
- **labels** (keys of `JCirc` tuple.) – Variable to map plot aspects to different colors. Default value = None
- **lmap** (*dicts*) – Name assignment to the label. Default value = None
- ****kwargs** – Additional keyword arguments are passed and are documented in `seaborn.histplot`.

Returns

Return type matplotlib.axes.Axes

circ_kde(*x*, *y=None*, *cbins=(0.05, 0.005)*, *labels=None*, *lmap=None*, ***kwargs*)

Draw a Kernel Density plot of circularity and normalized energy.

Plot univariate or bivariate distributions using kernel density estimation (KDE). This plot represents normalized specific energy, the circularity parameter (J_z/J_circ) and/or the projected circularity parameter (J_p/J_circ) of the stellar particles using a continuous probability density curve in one or more dimensions. This function groups the values of stellar particles according to some keys of `JCirc` tuple.

Parameters

- **x** (keys of `JCirc` tuple.) – Variables that specify positions on the x and y axes. Default value y = None.
- **y** (keys of `JCirc` tuple.) – Variables that specify positions on the x and y axes. Default value y = None.
- **cbins** (*tuple*) – It contains the two widths of bins necessary for the calculation of the circular angular momentum. Shape: (2,). Dafult value = (0.05, 0.005).
- **labels** (keys of `JCirc` tuple.) – Variable to map plot aspects to different colors. Default value = None
- **lmap** (*dicts*) – Name assignment to the label. Default value = None
- ****kwargs** – Additional keyword arguments are passed and are documented in `seaborn.kdeplot`.

Returns

Return type matplotlib.axes.Axes

5.3.4 galaxychop.models package

Module for dynamical decomposition models.

class galaxychop.models.**Components**(*labels*, *ptypes*, *probabilities*)
Bases: object

Class of components resulting from dynamic decomposition.

This class creates the components of the galaxy from the result of the dynamic decomposition.

Parameters

- **labels** (*np.ndarray*) – 1D array with the index of the component to which each particle belongs. Shape: (n,1).
- **ptypes** (*np.ndarray*) – Indicates the type of particle: stars = 0, dark matter = 1, gas = 2. Shape: (n,1).
- **probabilities** (*np.ndarray or None*) – 1D array with probabilities of the particles to belong to each component, in case the dynamic decomposition model includes them. Shape: (n,1). Otherwise it adopts the value None.

to_dataframe(*attributes=None*)
Convert to pandas data frame.

This method builds a data frame of all parameters of Components.

Returns **DataFrame** – DataFrame of all Components data.

Return type pandas.DataFrame

class galaxychop.models.**GalaxyDecomposerABC**(*, *cbins=(0.05, 0.005)*)
Bases: object

Abstract class to facilitate the creation of decomposers.

This class requests the redefinition of three methods: get_attributes, get_rows_mask and split.

Parameters **cbins** (*tuple*) – It contains the two widths of bins necessary for the calculation of the circular angular momentum. Shape: (2,). Dafult value = (0.05, 0.005).

abstract get_attributes()
Attributes for the parameter space.

Returns **attributes** – Particle attributes used to operate the clustering.

Return type keys of ParticleSet class parameters

abstract get_rows_mask(*X*, *y*, *attributes*)
Mask for the valid rows to operate clustering.

This method gets the mask for the valid rows to operate clustering.

Parameters

- **X** (*np.ndarray(n_particles, attributes)*) – 2D array where each file it is a diferent particle and each column is a attribute of the particles. n_particles is the total number of particles.
- **y** (*np.ndarray(n_particles,)*) – 1D array where is identified the nature of each particle: 0 = stars, 1 = dark matter, 2 = gas. n_particles is the total number of particles.
- **attributes** (*tuple*) – Dictionary keys of ParticleSet class parameters with particle attributes used to operate the clustering.

Returns **mask** – Mask only with valid values to operate the clustering.

Return type nd.array(m_particles)

abstract split(*X*, *y*, *attributes*)
Compute clustering.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – Training instances to cluster.
- **y** (*Ignored*) – Not used, present here for API consistency by convention.

Returns

- **labels** (*np.ndarray(m_particles)*) – 1D array with the index of the clusters to which each particle belongs. m_particles is the total number of particles with valid values to operate the clustering.
- **probs** (*np.ndarray(m_particles) or None*) – Probabilities of the particles to belong to each component, in case the dynamic decomposition model includes them. Otherwise it adopts the value None.

attributes_matrix(*galaxy*, *attributes*)
Matrix of particle attributes.

This method obtains the matrix with the particles and attributes necessary to operate the clustering.

Parameters

- **galaxy** (`Galaxy class` object) – Instance of Galaxy class.
- **attributes** (keys of `ParticleSet class` parameters) – Particle attributes used to operate the clustering.

Returns

- **X** (*np.ndarray(n_particles, attributes)*) – 2D array where each file it is a diferent particle and each column is a attribute of the particles. n_particles is the total number of particles.
- **y** (*np.ndarray(n_particles)*) – 1D array where is identified the nature of each particle: 0 = STARS, 1=DM, 2=Gas. n_particles is the total number of particles.

complete_labels(*X*, *labels*, *rows_mask*)
Complete the labels of all particles.

This method assigns the labels obtained from clustering to the particles used for this purpose. The rest are assigned as label=Nan.

Parameters

- **X** (*np.ndarray(n_particles, attributes)*) – 2D array where each file it is a diferent particle and each column is a parameter of the particles. n_particles is the total number of particles.
- **labels** (*np.ndarray(m_particles)*) – 1D array with the index of the clusters to which each particle belongs. m_particles is the total number of particles with valid values to operate the clustering.
- **rows_mask** (*nd.array(m_particles)*) – Mask only with valid values to operate the clustering. m_particles is the total number of particles with valid values to operate the clustering.

Returns **new_labels** – 1D array with the index of the clusters to which each particle belongs. Particles that do not belong to any of them are assigned the label Nan. n_particles is the total number of particles.

Return type np.ndarray(n_particles)

complete_probs(*X*, *probs*, *rows_mask*)

Complete the probabilities of all particles.

This method assigns the probabilities obtained from clustering to the particles used for this purpose, the rest are assigned as label=Nan. This method returns None in case the clustering method returns None probabilities.

Parameters

- **X** (*np.ndarray(n_particles, attributes)*) – 2D array where each file it is a diferent particle and each column is a parameter of the particles. n_particles is the total number of particles.
- **probs** (*np.ndarray(n_cluster, m_particles)*) – 2D array with probabilities of belonging to each component. n_cluster is the number of components obtained. m_particles is the total number of particles with valid values to operate the clustering.
- **rows_mask** (*nd.array(m_particles)*) – Mask only with valid values to operate the clustering. m_particles is the total number of particles with valid values to operate the clustering.

Returns **new_probs** – 2D array with probabilities of belonging to each component. n_cluster is the number of components obtained. n_particles is the total number of particles. Particles that do not belong to any component are assigned the label Nan. This method returns None in case the clustering method returns None probabilities.

Return type np.ndarray(n_cluster, n_particles)

decompose(*galaxy*)

Decompose method.

Assign the component of the galaxy to which each particle belongs. Validation of the input galaxy instance.

Parameters **galaxy** (`Galaxy class` object) – Instance of Galaxy class.

Returns Instance of the `Component class`, with the result of the dynamic decomposition.

Return type *Components*

class galaxychop.models.**DynamicStarsDecomposerMixin**

Bases: `object`

Dynamic Stars Decomposer Mixin Class.

This class redefines the get_row_mask method so that dynamic decomposition is performed using only stellar particles.

get_rows_mask(*X*, *y*, *attributes*)

Note: Only stellar particles are used to carry out the dynamic decomposition. In addition, the parameters of the parameter space, where the dynamic decomposition is carried out, must have finite values.

Parameters

- **X** (*np.ndarray(n_particles, attributes)*) – 2D array where each file it is a diferent particle and each column is a attribute of the particles. n_particles is the total number of particles.
- **y** (*np.ndarray(n_particles,)*) – 1D array where is identified the nature of each particle: 0 = stars, 1 = dark matter, 2 = gas. n_particles is the total number of particles.
- **attributes** (*tuple*) – Dictionary keys of `ParticleSet class` parameters with particle attributes used to operate the clustering.

Returns **mask** – Mask only with valid values to operate the clustering.

Return type nd.array(m_particles)

class galaxychop.models.**JThreshold**(*, *cbins=(0.05, 0.005)*, *eps_cut=0.6*)

Bases: *galaxychop.models._base.DynamicStarsDecomposerMixin*, *galaxychop.models._base.GalaxyDecomposerABC*

JThreshold class.

Implementation of galaxy dynamical decomposition model using only the circularity parameter. Tissera et al.(2012)[2], Marinacci et al.(2014)[3], Vogelsberger et al.(2014)[4], Park et al.(2019)[5] .

Parameters **eps_cut** (*float, default=0.6*) – Cut-off value in the circularity parameter. Stellar particles with eps > eps_cut are assigned to the disk and stellar particles with eps <= eps_cut to the spheroid.

Notes

Index of the cluster each stellar particles belongs to: Index=0: correspond to galaxy spheroid. Index=1: correspond to galaxy disk.

Examples

Example of implementation.

```
>>> import galaxychop as gchop
>>> galaxy = gchop.read_hdf5(...)
>>> galaxy = gchop.star_align(gchop.center(galaxy))
>>> chopper = gchop.JThreshold()
>>> chopper.decompose(galaxy)
```

[2] Tissera, P. B., White, S. D. M., and Scannapieco, C., "Chemical signatures of formation processes in the stellar populations of simulated galaxies", Monthly Notices of the Royal Astronomical Society, vol. 420, no. 1, pp. 255-270, 2012. doi:10.1111/j.1365-2966.2011.20028.x. https://ui.adsabs.harvard.edu/abs/2012MNRAS.420..255T/abstract

[3] Marinacci, F., Pakmor, R., and Springel, V., "The formation of disc galaxies in high-resolution moving-mesh cosmological simulations", Monthly Notices of the Royal Astronomical Society, vol. 437, no. 2, pp. 1750-1775, 2014. doi:10.1093/mnras/stt2003. https://ui.adsabs.harvard.edu/abs/2014MNRAS.437.1750M/abstract

[4] Vogelsberger, M., "Introducing the Illustris Project: simulating the coevolution of dark and visible matter in the Universe", Monthly Notices of the Royal Astronomical Society, vol. 444, no. 2, pp. 1518-1547, 2014. doi:10.1093/mnras/stu1536. https://ui.adsabs.harvard.edu/abs/2014MNRAS.444.1518V/abstract

[5] Park, M.-J., "New Horizon: On the Origin of the Stellar Disk and Spheroid of Field Galaxies at z = 0.7", The Astrophysical Journal, vol. 883, no. 1, 2019. doi:10.3847/1538-4357/ab3afe. https://ui.adsabs.harvard.edu/abs/2019ApJ...883...25P/abstract

References

check_eps_cut(*attribute*, *value*)
Eps_cut value validator.

This method validates that the value of eps_cut is in the interval (-1,1).

get_attributes()
Attributes for the parameter space.

Returns **attributes** – Particle attributes used to operate the clustering.

Return type keys of `ParticleSet class` parameters

Notes

In this model the parameter space is given by eps: circularity parameter (J_z/J_circ).

split(*X*, *y*, *attributes*)
Compute clustering.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – Training instances to cluster.
- **y** (*Ignored*) – Not used, present here for API consistency by convention.

Returns

- **labels** (*np.ndarray(m_particles)*) – 1D array with the index of the clusters to which each particle belongs. m_particles is the total number of particles with valid values to operate the clustering.
- **probs** (*np.ndarray(m_particles) or None*) – Probabilities of the particles to belong to each component, in case the dynamic decomposition model includes them. Otherwise it adopts the value None.

Notes

The attributes used by the model are described in detail in the class documentation.

class galaxychop.models.**JHistogram**(***, *cbins=(0.05, 0.005)*, *n_bin=100*, *digits=2*, *random_state=None*)
Bases: *galaxychop.models._base.DynamicStarsDecomposerMixin*, *galaxychop.models._base.GalaxyDecomposerABC*

JHistogram class.

Implementation of galaxy dynamical decomposition model described in Abadi et al. (2003)[1].

Parameters

- **n_bin** (*int, default=100*) – Number of bins needed to build the circularity parameter histogram.
- **digits** (*int, default=2*) – Number of decimals to which an array is rounded.
- **seed** (*int, default=None*) – Seed to initialize the random generator.

[1] Abadi, M. G., Navarro, J. F., Steinmetz, M., and Eke, V. R., "Simulations of Galaxy Formation in a Cold Dark Matter Universe. II. The Fine Structure of Simulated Galactic Disks", The Astrophysical Journal, vol. 597, no. 1, pp. 21–34, 2003. doi:10.1086/378316. https://ui.adsabs.harvard.edu/abs/2003ApJ...597...21A/abstract

Notes

Index of the cluster each stellar particles belongs to: Index=0: correspond to galaxy spheroid. Index=1: correspond to galaxy disk.

Examples

Example of implementation of Abadi Model.

```
>>> import galaxychop as gchop
>>> galaxy = gchop.read_hdf5(...)
>>> galaxy = gchop.star_align(gchop.center(galaxy))
>>> chopper = gchop.JHistogram()
>>> chopper.decompose(galaxy)
```

References

`get_attributes`()
Attributes for the parameter space.

Returns **attributes** – Particle attributes used to operate the clustering.

Return type keys of `ParticleSet class` parameters

Notes

In this model the parameter space is given by eps: circularity parameter (J_z/J_circ).

`split`(*X*, *y*, *attributes*)
Compute clustering.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – Training instances to cluster.
- **y** (*Ignored*) – Not used, present here for API consistency by convention.

Returns

- **labels** (*np.ndarray(m_particles)*) – 1D array with the index of the clusters to which each particle belongs. m_particles is the total number of particles with valid values to operate the clustering.
- **probs** (*np.ndarray(m_particles) or None*) – Probabilities of the particles to belong to each component, in case the dynamic decomposition model includes them. Otherwise it adopts the value None.

Notes

The attributes used by the Abadi model are described in detail in the class documentation.

class galaxychop.models.**JEHistogram**(**, cbins=(0.05, 0.005), n_bin=100, digits=2, random_state=None, n_bin_E=20*)

Bases: *galaxychop.models._histogram.JHistogram*

JEHistogram class.

Implementation of a modification of Abadi galaxy dynamical decomposition model using the circularity parameter and specific energy distribution.

Parameters

- **n_bin_E** (*int, default=20*) – Number of bins needed to build the normalised specific energy histogram.
- ****kwargs** (*key, value mappings*) – Other optional keyword arguments are passed through to *JHistogram* classes.

Notes

Index of the cluster each stellar particles belongs to: Index=0: correspond to galaxy spheroid. Index=1: correspond to galaxy disk.

Examples

Example of the implementation of the modified Abadi model.

```
>>> import galaxychop as gchop
>>> galaxy = gchop.read_hdf5(...)
>>> galaxy = gchop.star_align(gchop.center(galaxy))
>>> chopper = gchop.JEHistogram()
>>> chopper.decompose(galaxy)
```

get_attributes()

Attributes for the parameter space.

Returns **attributes** – Particle attributes used to operate the clustering.

Return type keys of `ParticleSet class` parameters

Notes

In this model the parameter space is given by normalized_star_energy: normalized specific energy of the stars eps: circularity parameter (J_z/J_circ).

split(*X*, *y*, *attributes*)

Compute clustering.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – Training instances to cluster.
- **y** (*Ignored*) – Not used, present here for API consistency by convention.

Returns

- **labels** (*np.ndarray(m_particles)*) – 1D array with the index of the clusters to which each particle belongs. m_particles is the total number of particles with valid values to operate the clustering.
- **probs** (*np.ndarray(m_particles) or None*) – Probabilities of the particles to belong to each component, in case the dynamic decomposition model includes them. Otherwise it adopts the value None.

Notes

The attributes used by the modified Abadi model are described in detail in the class documentation.

class galaxychop.models.**KMeans**(*, *cbins=(0.05, 0.005)*, *n_components=2*, *init='k-means++'*, *n_init=10*, *max_iter=300*, *tol=0.0001*, *verbose=0*, *random_state=None*, *algorithm='auto'*)

Bases: *galaxychop.models._base.DynamicStarsDecomposerMixin*, *galaxychop.models._base.GalaxyDecomposerABC*

KMeans class.

Implementation of Scikit-learn[6] K-means as a method for dynamically decomposing galaxies.

Parameters

- **n_components** (*int, default=2*) – The number of clusters to form as well as the number of centroids to generate.
- **init** (*{'k-means++', 'random'}, callable or array-like of shape*) –
- **(n_clusters** – Parameter of :py:class:k-Means class into scikit-learn library.
- **n_features)** – Parameter of :py:class:k-Means class into scikit-learn library.
- **default="k-means++"** – Parameter of :py:class:k-Means class into scikit-learn library.
- **n_init** (*int, default=10*) – Parameter of :py:class:k-Means class into scikit-learn library.
- **max_iter** (*int, default=300*) – Parameter of :py:class:k-Means class into scikit-learn library.
- **tol** (*float, default=0.0001*) – Parameter of :py:class:k-Means class into scikit-learn library.
- **verbose** (*int, default=0*) – Parameter of :py:class:k-Means class into scikit-learn library.
- **random_state** (*int, default=None*) – Parameter of :py:class:k-Means class into scikit-learn library.
- **algorithm** (*{"auto", "full", "elkan"}, default="auto"*) – Parameter of :py:class:k-Means class into scikit-learn library.

[6] Pedregosa et al., Journal of Machine Learning Research 12, pp. 2825-2830, 2011. https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

Notes

More information for *KMeans* class: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Examples

Example of implementation of KMeans Model.

```
>>> import galaxychop as gchop
>>> galaxy = gchop.read_hdf5(...)
>>> galaxy = gchop.star_align(gchop.center(galaxy))
>>> chopper = gchop.KMeans()
>>> chopper.decompose(galaxy)
```

References

`get_attributes`()
Attributes for the parameter space.

Returns **attributes** – Particle attributes used to operate the clustering.

Return type keys of `ParticleSet class` parameters

Notes

In this model the parameter space is given by normalized_star_energy: normalized specific energy of the stars eps: circularity parameter (J_z/J_circ) eps_r: projected circularity parameter (J_p/J_circ).

`split`(*X*, *y*, *attributes*)
Compute clustering.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – Training instances to cluster.
- **y** (*Ignored*) – Not used, present here for API consistency by convention.

Returns

- **labels** (*np.ndarray(m_particles)*) – 1D array with the index of the clusters to which each particle belongs. m_particles is the total number of particles with valid values to operate the clustering.
- **probs** (*np.ndarray(m_particles) or None*) – Probabilities of the particles to belong to each component, in case the dynamic decomposition model includes them. Otherwise it adopts the value None.

Notes

The attributes used by the kmeans model are described in detail in the class documentation.

class galaxychop.models.**DynamicStarsGaussianDecomposerABC**(*, *cbins=(0.05, 0.005)*, *covariance_type='full'*, *tol=0.001*, *reg_covar=1e-06*, *max_iter=100*, *n_init=10*, *init_params='kmeans'*, *weights_init=None*, *means_init=None*, *precisions_init=None*, *random_state=None*, *warm_start=False*, *verbose=0*, *verbose_interval=10*)

Bases: *galaxychop.models._base.DynamicStarsDecomposerMixin*, *galaxychop.models._base.GalaxyDecomposerABC*

Dynamic Stars Gaussian Decomposer Class.

Parameters

- **covariance_type** (*{'full', 'tied', 'diag', 'spherical'}, default="full"*) – Parameter of :py:class:GaussianMixture class into *scikit-learn* library.
- **tol** (*float, default=0.001*) – Parameter of :py:class:GaussianMixture class into scikit-learn library.
- **reg_covar** (*float, default=1e-06*) – Parameter of :py:class:GaussianMixture class into scikit-learn library.
- **max_iter** (*float, default=100*) – Parameter of :py:class:GaussianMixture class into scikit-learn library.
- **n_init** (*int, default=10*) – Parameter of :py:class:GaussianMixture class into scikit-learn library.
- **init_params** (*{'kmeans', 'random'}, default="kmeans"*) – Parameter of :py:class:GaussianMixture class into scikit-learn library.
- **weights_init** (*array-like of shape (n_components,), default=None*) – Parameter of :py:class:GaussianMixture class into scikit-learn library.
- **means_init** (*array-like of shape (n_components, n_features), default=None*) – Parameter of :py:class:GaussianMixture class into scikit-learn library.
- **precisions_init** (*array-like, default=None*) – Parameter of :py:class:GaussianMixture class into scikit-learn library.
- **random_state** (*int, default=None*) – Parameter of :py:class:GaussianMixture class into scikit-learn library.
- **warm_start** (*bool, default=False*) – Parameter of :py:class:GaussianMixture class into scikit-learn library.
- **verbose** (*int, default=0*) – Parameter of :py:class:GaussianMixture class into scikit-learn library.
- **verbose_interval** (*int, default=10*) –

get_attributes()

Attributes for the parameter space.

Returns **attributes** – Particle attributes used to operate the clustering.

Return type keys of `ParticleSet class` parameters

Notes

In this model the parameter space is given by normalized_star_energy: normalized specific energy of the stars eps: circularity parameter (J_z/J_circ) eps_r: projected circularity parameter (J_p/J_circ).

class galaxychop.models.**GaussianMixture**(*, *cbins=(0.05, 0.005)*, *covariance_type='full'*, *tol=0.001*, *reg_covar=1e-06*, *max_iter=100*, *n_init=10*, *init_params='kmeans'*, *weights_init=None*, *means_init=None*, *precisions_init=None*, *random_state=None*, *warm_start=False*, *verbose=0*, *verbose_interval=10*, *n_components=2*)

Bases: *galaxychop.models._gaussian_mixture.DynamicStarsGaussianDecomposerABC*

GaussianMixture class.

Implementation of the method for dynamically decomposing galaxies described by Obreja et al.(2018)[7] .

Parameters

- **n_components** (*int, default=2*) – The number of mixture components. Parameter of :py:class:`GaussianMixture` class into `scikit-learn` library.
- ****kwargs** (*key, value mappings*) – Other optional keyword arguments are passed through to :py:class:`GaussianMixture` class into `scikit-learn` library.

Notes

More information for `GaussianMixture` class: https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html

Examples

Example of implementation of Gaussian Mixture Model.

```
>>> import galaxychop as gchop
>>> galaxy = gchop.read_hdf5(...)
>>> galaxy = gchop.star_align(gchop.center(galaxy))
>>> chopper = gchop.GaussianMixture()
>>> chopper.decompose(galaxy)
```

[7] Obreja, A., "Introducing galactic structure finder: the multiple stellar kinematic structures of a simulated Milky Way mass galaxy", Monthly Notices of the Royal Astronomical Society, vol. 477, no. 4, pp. 4915-4930, 2018. doi:10.1093/mnras/sty1022. https://ui.adsabs.harvard.edu/abs/2018MNRAS.477.4915O/abstract

References

split(*X*, *y*, *attributes*)
Compute clustering.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – Training instances to cluster.
- **y** (*Ignored*) – Not used, present here for API consistency by convention.

Returns

- **labels** (*np.ndarray(m_particles)*) – 1D array with the index of the clusters to which each particle belongs. m_particles is the total number of particles with valid values to operate the clustering.
- **probs** (*np.ndarray(m_particles) or None*) – Probabilities of the particles to belong to each component, in case the dynamic decomposition model includes them. Otherwise it adopts the value None.

Notes

The attributes used by the model are described in detail in the class documentation.

class galaxychop.models.**AutoGaussianMixture**(**, cbins=(0.05, 0.005), covariance_type='full', tol=0.001, reg_covar=1e-06, max_iter=100, n_init=10, init_params='kmeans', weights_init=None, means_init=None, precisions_init=None, random_state=None, warm_start=False, verbose=0, verbose_interval=10, c_bic=0.1, n_jobs=None*)

Bases: *galaxychop.models._gaussian_mixture.DynamicStarsGaussianDecomposerABC*

AutoGaussianMixture class.

Implementation of the auto-gmm method for dynamically decomposing galaxies described by Du et al.(2019)[8] .

Parameters

- **c_bic** (*float, default=0.1*) – Cut value of the criteria for the automatic choice of the number of gaussians.
- **n_jobs** (*int, default=None*) –
- ****kwargs** (*key, value mappings*) – Other optional keyword arguments are passed through to :py:class:GaussianMixture class into scikit-learn library.

[8] Du, M., "Identifying Kinematic Structures in Simulated Galaxies Using Unsupervised Machine Learning", The Astrophysical Journal, vol. 884, no. 2, 2019. doi:10.3847/1538-4357/ab43cc. https://ui.adsabs.harvard.edu/abs/2019ApJ...884..129D/abstract

Notes

Index of the cluster each stellar particles belongs to: Index of the cluster each stellar particles belongs to. Index=0: correspond to galaxy stellar halo. Index=1: correspond to galaxy bulge. Index=2: correspond to galaxy cold disk. Index=3: correspond to galaxy warm disk.

Examples

Example of implementation of auto-gmm model.

```
>>> import galaxychop as gchop
>>> galaxy = gchop.read_hdf5(...)
>>> galaxy = gchop.star_align(gchop.center(galaxy))
>>> chopper = gchop.AutoGaussianMixture()
>>> chopper.decompose(galaxy)
```

References

split(*X*, *y*, *attributes*)
Compute clustering.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – Training instances to cluster.
- **y** (*Ignored*) – Not used, present here for API consistency by convention.

Returns

- **labels** (*np.ndarray(m_particles)*) – 1D array with the index of the clusters to which each particle belongs. m_particles is the total number of particles with valid values to operate the clustering.
- **probs** (*np.ndarray(m_particles) or None*) – Probabilities of the particles to belong to each component, in case the dynamic decomposition model includes them. Otherwise it adopts the value None.

galaxychop.models.**hparam**(*default*, ***kwargs*)
Create a hyper parameter for decomposers.

By design decision, hyper-parameter is required to have a sensitive default value.

Parameters

- **default** – Sensitive default value of the hyper-parameter.
- ****kwargs** – Additional keyword arguments are passed and are documented in `attr.ib()`.

Returns

Return type Hyper parameter with a default value.

Notes

This function is a thin-wrapper over the attrs function `attr.ib()`.

5.3.5 `galaxychop.utils` package

Utilities module.

`galaxychop.utils.`**`potential`**(*galaxy*, *backend='numpy'*)
Potential energy calculation.

Given the positions and masses of particles, calculate their specific gravitational potential energy.

Parameters **`galaxy`** (`Galaxy class` object) –

Returns **galaxy** – A new galaxy object with the specific potential energy of particles calculated.

Return type new `Galaxy class` object

`galaxychop.utils.`**`center`**(*galaxy*)
Particle centring.

Centers the position of all galaxy particles respect to the position of the lowest potential particle.

Parameters **`galaxy`** (`Galaxy class` object) –

Returns **galaxy** – A new galaxy object with centered positions respect to the position of the lowest potential particle.

Return type new `Galaxy class` object

`galaxychop.utils.`**`is_centered`**(*galaxy*, *rtol=1e-05*, *atol=1e-08*)
Validate if the galaxy is centered.

Parameters

- **`galaxy`** (`Galaxy class` object) –
- **`rtol`** (*`float`*) – Relative tolerance. Default value = 1e-05.
- **`atol`** (*`float`*) – Absolute tolerance. Default value = 1e-08.

Returns True if galaxy is centered respect to the position of the lowest potential particle, False otherwise.

Return type bool

`galaxychop.utils.`**`star_align`**(*galaxy*, ***, *r_cut=None*)
Align the galaxy.

Rotates the positions, velocities and angular momentum of the particles so that the total angular moment of the stars particles coincides with the z-axis. Optionally, only stars particles within a cutting radius *(r_cut)* can be used to calculate the rotation matrix.

Parameters

- **`galaxy`** (`Galaxy class` object) –
- **`r_cut`** (*`float, optional`*) – Default value = None. If it's provided, it must be positive and the rotation matrix *A* is calculated from the particles with radii smaller than r_cut.

Returns **galaxy** – A new galaxy object with their total angular momentum aligned with the z-axis.

Return type new `Galaxy class` object

galaxychop.utils.**is_star_aligned**(*galaxy*, ***, *r_cut=None*, *rtol=1e-05*, *atol=1e-08*)
Validate if the galaxy is aligned.

Parameters

- **galaxy** (Galaxy class object) –
- **r_cut** (*float, optional*) – Default value = None. If it's provided, it must be positive and the rotation matrix *A* is calculated from the particles with radii smaller than r_cut.
- **rtol** (*float*) – Relative tolerance. Default value = 1e-05.
- **atol** (*float*) – Absolute tolerance. Default value = 1e-08.

Returns True if the total angular momentum of the galaxy is aligned with the z-axis, False otherwise.

Return type bool

class galaxychop.utils.**JCirc**(*normalized_star_energy*, *normalized_star_Jz*, *eps*, *eps_r*, *x*, *y*)
Bases: object

Circularity information about the stars particles of a galaxy.

Parameters

- **normalized_star_energy** (*np.array*) – Normalized specific energy of stars.
- **normalized_star_Jz** (*np.array*) – z-component normalized specific angular momentum of the stars.
- **eps** (*np.array*) – Circularity parameter (eps : J_z/J_circ).
- **eps_r** (*np.array*) – Projected circularity parameter (eps_r: J_p/J_circ).
- **x** (*np.array*) – Normalized specific energy for the particle with the maximum z-component of the normalized specific angular momentum per bin.
- **y** (*np.array*) – Maximum value of the z-component of the normalized specific angular momentum per bin.

classmethod circularity_attributes()
Retrieve all the circularity attributes stored in the JCirc class.

This method returns a tuple of str ignoring those that are marked as "asdict=False".

as_dict()
Convert the instance to a dict.

Attributes are ignored if they are marked as "asdict=False".

isfinite()
Return a mask of which elements are finite in all attributes.

Attributes are ignored if they are marked as "asdict=False".

galaxychop.utils.**jcirc**(*galaxy*, *bin0=0.05*, *bin1=0.005*, *runtime_warnings='ignore'*)
Calculate galaxy stars particles circularity information.

Calculation of Normalized specific energy of the stars, z-component normalized specific angular momentum of the stars, circularity parameter, projected circularity parameter, and the points to build the function of the circular angular momentum.

Parameters

- **galaxy** (Galaxy class object) –

- **bin0** (*float. Default=0.05*) – Size of the specific energy bin of the inner part of the galaxy, in the range of (-1, -0.1) of the normalized energy.
- **bin1** (*float. Default=0.005*) – Size of the specific energy bin of the outer part of the galaxy, in the range of (-0.1, 0) of the normalized energy.
- **runtime_warnings** (*Any warning filter action (default "ignore")*) – jcirc usually launches RuntimeWarning during the eps calculation because there may be some particle with jcirc=0. By default the function decides to ignore these warnings. *runtime_warnings* can be set to any valid "action" in the python warnings module.

Returns Circularity attributes of the star components of the galaxy

Return type *JCirc*

Notes

The *x* and *y* are calculated from the binning in the normalized specific energy. In each bin, the particle with the maximum value of z-component of standardized specific angular momentum is selected. This value is assigned to the *y* parameter and its corresponding normalized specific energy pair value to *x*.

Examples

This returns the normalized specific energy of stars (E_star_norm), the z-component normalized specific angular momentum of the stars (Jz_star_norm), the circularity parameters (eps : J_z/J_circ and eps_r: J_p/J_circ), and the normalized specific energy for the particle with the maximum z-component of the normalized specific angular momentum per bin (*x*) and the maximum value of the z-component of the normalized specific angular momentum per bin (*y*).

```
>>> import galaxychop as gchop
>>> galaxy = gchop.Galaxy(...)
>>> E_star_norm, Jz_star_norm, eps, eps_r, x, y =
>>>         galaxy.jcir(bin0=0.05, bin1=0.005)
```

galaxychop.utils.**doc_inherit**(*parent*)

Inherit the 'parent' docstring.

Returns a function/method decorator that, given parent, updates the docstring of the decorated function/method based on the *numpy* style and the corresponding attribute of parent.

Parameters **parent** (*Union[str, Any]*) – The docstring, or object of which the docstring is utilized as the parent docstring during the docstring merge.

Notes

This decorator is a thin layer over py:function:*custom_inherit.doc_inherit decorator*.

Check: <github *https://github.com/rsokl/custom_inherit*>__

5.4 Licence

CHAPTER

SIX

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

g

INDEX

H

I

J

K

M

P

R

S

T