SAGE-analysis Documentation

Release 0.0.1

Jacob Seiler, Manodeep Sinha, Darren Croton

Sep 21, 2023

User Documentation

This is the documentation for the Semi-Analytic Galaxy Evolution (**SAGE**) analysis package. This package ingests, analyses, and plots the data products of the **SAGE** model, located here. Please refer to the to the **SAGE** repo for full documentation regarding how to run the base model.

CHAPTER 1

Installation

The recommended installing method is through pip:

```
$ pip install sage-analysis
```

CHAPTER 2

Maintainers

- Jacob Seiler (@jacobseiler)
- *User Documentation*
- *API Reference*

2.1 Introduction

The **SAGE**-analysis package was developed to handle the data products produced by the **SAGE** model, available here.

2.1.1 Why Did We Create A New Repo?

SAGE is an extremely modular and flexible semi-analytic model. The base model (presented in Croton et al., 2016) has been adjusted and altered to answer a number of science questions on a variety of topics including galactic HI and angular momentum properties in DARK SAGE, the Epoch of Reionization in RSAGE, and the star formation history and galactic dust in *DUSTY SAGE* _.

Due to the wide array of science that **SAGE** can cover and the number of models that spawned from its development, there has been a need to develop a framework to ingest in data from (ideally) any **SAGE** variant. This repo represents such an effort. It represents a series of modules intended to provide the easy ingestion and analysis of the base **SAGE** data, whilst serving as a template for analysing any other **SAGE** flavours.

2.1.2 Advantages of the Package

- Easy analysis and plotting of multiple different **SAGE** models. For example, comparing **SAGE** models with/without supernova feedback.
- Memory efficient analysis of **SAGE** output. All calculations are performed using only a single output file at a time, ensuring no extra memory overhead associated with opening many files.
- Support for the user to implement their own functions to analysis + plotting.

- Template for creating custom data classes to ingest any arbitrary **SAGE** data output. Useful if you're looking to develop using **SAGE**.

2.2 Setting up SAGE

This package ingests, analyses, and plots the data products produced by **SAGE**. Hence, the first step is to run the **SAGE** model and simulate some galaxies! We defer to the **SAGE** documentation for instructions on how run the **SAGE** model.

2.3 Analyzing SAGE Output

The output from **SAGE** is analyzed using the respective parameter files used to run **SAGE** itself. Here, we will assume the parameter file is located in `/home/Desktop/sage-model/input/millennium.ini`.

On this page, we outline some of the basic features of **sage-analysis** that can be used to analyze and plot **SAGE** output.

2.3.1 Basic Analysis

Out of the box, **sage-analysis** will analyse the latest snapshot (i.e., the lowest redshift) and save the plots in the `./plots` directory.

```
from sage_analysis.galaxy_analysis import GalaxyAnalysis

par_fnames = ["/home/Desktop/sage-model/input/millennium.ini"]

galaxy_analysis = GalaxyAnalysis(par_fnames)
galaxy_analysis.analyze_galaxies()
galaxy_analysis.generate_plots()
```

The output path can be changed by adjusting the `plot_output_path` variable passed to `generate_plots()`.

If you ran **SAGE** using `sage-binary` output, you will need to specify the *`first_file_to_analyze`*, *`last_file_to_analyze`*, and *`num_sage_output_files`* for each model. These will need to be specified for all the following examples. For brevity, we will omit them in the following and assume that **SAGE** has been run using `sage-hdf5` output (recommended).

```
from sage_analysis.galaxy_analysis import GalaxyAnalysis

par_fnames = ["/home/Desktop/sage-model/input/millennium.ini"]
first_files_to_analyze = [0]  # The first files that you wish to analyze + plot.
last_files_to_analyze = [0]  # The last files that you wish to analyze + plot.
num_sage_output_files = [1]  # The number of files that SAGE produced; usually the␣
↪number of processors it ran on.

galaxy_analysis = GalaxyAnalysis(
    par_fnames,
    first_files_to_analyze=first_files_to_analyze,
    last_files_to_analyze=last_files_to_analyze,
    num_sage_output_files=num_sage_output_files
)
galaxy_analysis.analyze_galaxies()
galaxy_analysis.generate_plots()
```

2.3.2 Accessing Galaxy Properties

Galaxy properties can be accessed via the `properties` attribute of an individual *Model* class. It can be useful to generate only the properties and not performing plotting (e.g., if you wish to use the properties for your own purpose).

```
from sage_analysis.galaxy_analysis import GalaxyAnalysis

par_fnames = ["/home/Desktop/sage-model/input/millennium.ini"]

galaxy_analysis = GalaxyAnalysis(par_fnames)
galaxy_analysis.analyze_galaxies()

print(galaxy_analysis.models)
print(galaxy_analysis.models[0].bins["stellar_mass_bins"])  # The stellar mass bins␣
↪(log10 Msun).
print(galaxy_analysis.models[0].properties["snapshot_63"]["SMF"])  # The number of␣
↪galaxies in each bin.

>>> [========================
... Model Mini-Millennium
... SAGE File:/home/Desktop/sage-model/input/millennium.ini
... SAGE Output Format: sage_hdf5
... First file to read: 0
... Last file to read: 0
... ========================]

>>> [ 8.   8.1  8.2  8.3  8.4  8.5  8.6  8.7  8.8  8.9  9.   9.1  9.2  9.3
...  9.4  9.5  9.6  9.7  9.8  9.9 10.  10.1 10.2 10.3 10.4 10.5 10.6 10.7
...  10.8 10.9 11.  11.1 11.2 11.3 11.4 11.5 11.6 11.7 11.8 11.9 12. ]

>>> [1148. 1328. 1456. 1698. 1836. 1824. 1778. 1576. 1313. 1091.  955.  830.
...  791.  734.  656.  662.  659.  593.  550.  552.  496.  483.  475.  425.
...  401.  291.  293.  248.  229.  190.  124.   71.   47.   18.    3.    0.
...  0.    0.    0.    0.]
```

2.3.3 Analyze Only a Subset of Files

For extremely large simulations, it may be prudent to analyze only a subset of files. For example, if **SAGE** run in parallel across 32 processors, we may only wish to analyze a quarter of these. This can be achieved by specifying the *first_file_to_analyze* and *last_file_to_analyze* for each model.

```
from sage_analysis.galaxy_analysis import GalaxyAnalysis

par_fnames = ["/home/Desktop/sage-model/input/millennium.ini"]
first_files_to_analyze = [0]
last_files_to_analyze = [7]

galaxy_analysis = GalaxyAnalysis(
    par_fnames,
    first_files_to_analyze=first_files_to_analyze,
    last_files_to_analyze=last_files_to_analyze,
)
galaxy_analysis.analyze_galaxies()
galaxy_analysis.generate_plots()
```

2.3.4 Turning On and Off Properties

Properties are analyzed and plotted according to the values in `plot_toggles`. The default values of this dictionary are set to analyze all basic properties, with the exception of properties tracked over time.

```
from sage_analaysis.default_analysis_arguments import default_plot_toggles
print(default_plot_toggles)

>>> {
        'SMF': True,
        'BMF': True,
        'GMF': True,
        'BTF': True,
        'sSFR': True,
        'gas_fraction': True,
        'metallicity': True,
        'bh_bulge': True,
        'quiescent': True,
        'bulge_fraction': True,
        'baryon_fraction': True,
        'reservoirs': True,
        'spatial': True,
        'SMF_history': False,
        'SFRD_history': False,
        'SMD_history': False
    }
```

By adjusting these properties, or specifying a custom set, you can control which properties you want to analyze.

```
from sage_analaysis.default_analysis_arguments import default_plot_toggles
from sage_analysis.galaxy_analysis import GalaxyAnalysis

par_fnames = ["/home/Desktop/sage-model/input/millennium.ini"]

# Plot only the stellar mass function and black hole-bulge relationship.
galaxy_analysis = GalaxyAnalysis(par_fnames, plot_toggles={"SMF": True, "bh_bulge":␣
↪True})
galaxy_analysis.analyze_galaxies()
galaxy_analysis.generate_plots()

# Plot all properties EXCEPT the mass-metallicity relationship.
plot_toggles = default_plot_toggles.copy()  # Copy to ensure ``default_plot_toggles``␣
↪aren't overwritten.
plot_toggles["metallicity"] = False

galaxy_analysis = GalaxyAnalysis(par_fnames, plot_toggles=plot_toggles)
galaxy_analysis.analyze_galaxies()
galaxy_analysis.generate_plots()
```

2.3.5 Analyzing Basic Properties Over Redshift

It can also be very useful to investigate how properties evolve over many snapshots. By default, **sage-analysis** supports analyzing the stellar mass function, stellar mass density, and star formation rate density over redshift.

Note: Ensure that **SAGE** has outputs for multiple snapshots. Try setting `NumOutputs` to `-1` and re-running **SAGE**.

These extra properties can be set by turning their respective entries in `plot_toggles`.

```
from sage_analysis.galaxy_analysis import GalaxyAnalysis

par_fnames = ["/home/Desktop/sage-model/input/millennium.ini"]

galaxy_analysis = GalaxyAnalysis(
    par_fnames, plot_toggles={"SMF_history": True, "SMD_history": True, "SFRD_history
↪": True},
)
galaxy_analysis.analyze_galaxies()
galaxy_analysis.generate_plots()
```

By default, these extra properties are analyzed and plotted for all available redshifts. You can also specify which redshifts you want to analyze, with **sage-analysis** selecting the snapshots that are closest to the desired redshifts specified. This is especially useful for the stellar mass function where we often want to investigate the evolution at specific redshifts.

```
from sage_analysis.galaxy_analysis import GalaxyAnalysis

par_fnames = ["/home/Desktop/sage-model/input/millennium.ini"]

galaxy_analysis = GalaxyAnalysis(
    par_fnames,
    plot_toggles={"SMF_history": True},
    history_redshifts={"SMF_history": [0.0, 0.5, 1.0, 2.0, 3.0]},
   )
galaxy_analysis.analyze_galaxies()
galaxy_analysis.generate_plots()
```

To analyse and plot these properties in addition to the other properties (e.g., the baryon fraction, quiescent fraction, etc), use and update the `default_plot_toggles` value.

```
from sage_analysis.default_analysis_arguments import default_plot_toggles

plot_toggles = default_plot_toggles.copy()  # Copy to ensure ``default_plot_toggles``␣
↪aren't overwritten.

plot_toggles["SMF_history"] = True
plot_toggles["SMD_history"] = True
plot_toggles["SFRD_history"] = True

galaxy_analysis = GalaxyAnalysis(par_fnames, plot_toggles=plot_toggles)
galaxy_analysis.analyze_galaxies()
galaxy_analysis.generate_plots()
```

2.3.6 Changing the Snapshot

By default, **sage-analysis** will analyze the lowest redshift snapshot for each model. This behaviour can be adjusted to analyze any arbitrary snapshot.

```
from sage_analysis.galaxy_analysis import GalaxyAnalysis

par_fnames = ["/home/Desktop/sage-model/input/millennium.ini"]
snapshots = [[50]]
```

(continues on next page)

(continued from previous page)

```
galaxy_analysis = GalaxyAnalysis(par_fnames)
galaxy_analysis.analyze_galaxies(snapshots=snapshots)
galaxy_analysis.generate_plots(snapshots=snapshots)
```

2.3.7 Changing the Redshift

Alternatively, rather than specifying the snapshot for each model, one can specify the redshift. **sage-analysis** will analyze the snapshot closest to these redshifts.

```
from sage_analysis.galaxy_analysis import GalaxyAnalysis

par_fnames = ["/home/Desktop/sage-model/input/millennium.ini"]
redshifts = [[1.0]]

galaxy_analysis = GalaxyAnalysis(par_fnames)
galaxy_analysis.analyze_galaxies(redshifts=redshifts)
galaxy_analysis.generate_plots(redshifts=redshifts)
```

Note: The `snapshots` and `redshifts` parameters **cannot both be** specified, only one may be used.

2.3.8 Multiple Models

sage-analysis supports analyzing and plotting of multiple **SAGE** model outputs. For example, let's say we want to compare the stellar mass function for **SAGE** run with and without supernovae feedback. This model has been run using a parameter file `/home/Desktop/sage-model/input/millennium_no_SN.ini`

```
from sage_analysis.galaxy_analysis import GalaxyAnalysis

par_fnames = ["/home/Desktop/sage-model/input/millennium.ini", "/home/Desktop/sage-
↪model/input/millennium_no_SN.ini"]
labels = ["Supernovae feedback on", "Supernovae feedback off"]

galaxy_analysis = GalaxyAnalysis(par_fnames, labels=labels)
galaxy_analysis.analyze_galaxies()
galaxy_analysis.generate_plots()
```

2.3.9 Multiple Simulations

In the above example, we ran **SAGE** on the same underlying N-body simulation. However, we may wish to analyze how **SAGE** performs on different simulations, at the same redshift; e.g., we may wish to compare the stellar mass function at z = 1 for *Millennium* and *Bolshoi*.

```
from sage_analysis.galaxy_analysis import GalaxyAnalysis

par_fnames = ["/home/Desktop/sage-model/input/millennium.ini", "/home/Desktop/sage-
↪model/input/bolshoi.ini"]
labels = ["Millennium", "Bolshoi"]

galaxy_analysis = GalaxyAnalysis(par_fnames, labels=labels)
```

(continues on next page)

(continued from previous page)

```
redshifts = [[1.0], [1.0]]  # Specify the redshift for each model; necessary because␣
↪the snapshots are not aligned.
galaxy_analysis.analyze_galaxies(redshifts=redshifts)
galaxy_analysis.generate_plots(redshifts=redshifts)
```

Or perhaps we wish to see how the stellar mass density evolves for the different simulations…

```
from sage_analysis.galaxy_analysis import GalaxyAnalysis

par_fnames = ["/home/Desktop/sage-model/input/millennium.ini", "/home/Desktop/sage-
↪model/input/bolshoi.ini"]
labels = ["Millennium", "Bolshoi"]
plot_toggles = {"SFRD_history": True}

galaxy_analysis = GalaxyAnalysis(par_fnames, plot_toggles=plot_toggles)

galaxy_analysis.analyze_galaxies()
galaxy_analysis.generate_plots()
```

2.3.10 Adding Extra Keywords for Analysis and Plotting

Some properties can be broken down into sub-populations and analyzed separately. For example, the stellar mass function can be split into red and blue galaxies or the baryon fraction can be split into its constituent reservoirs. To access these extra functionalities, the *calculation_functions* and `plot_functions` dictionaries passed to the `GalaxyAnalysis` constructor need to be adjusted.

```
from sage_analysis.utils import generate_func_dict
from sage_analysis.galaxy_analysis import GalaxyAnalysis

par_fnames = ["/home/Desktop/sage-model/input/millennium.ini"]
plot_toggles = {"SMF": True, "baryon_fraction": True}

# For each toggle, specify the extra keyword arguments and their values.

# The calculation and plotting step can each have different keywords.
extra_keywords_calculations = {"SMF": {"calc_sub_populations": True}}
extra_keywords_plotting = {
    "SMF": {"plot_sub_populations": True},
    "baryon_fraction": {"plot_sub_populations": True}
}

# Now build a dictionary with these extra arguments.
calculation_functions = generate_func_dict(
    plot_toggles, "sage_analysis.example_calcs", "calc_", extra_keywords_calculations
)
plot_functions = generate_func_dict(
    plot_toggles, "sage_analysis.example_plots", "plot_", extra_keywords_plotting
)

# Then construct with these new dictionaries.
galaxy_analysis = GalaxyAnalysis(
    par_fnames,
    plot_toggles=plot_toggles,
    calculation_functions=calculation_functions,
```

(continues on next page)

(continued from previous page)

```
    plot_functions=plot_functions
)

galaxy_analysis.analyze_galaxies()
galaxy_analysis.generate_plots()
```

2.4 Defining Custom Properties

2.4.1 Default Properties

Out of the box, **sage-analysis** supports the analysis of a number of different properties.

Property	Plot Toggle Name	Description	Property Type
Stellar mass function	SMF	Number of galaxies with a given stellar mass.	Binned (on stellar mass).
Baryonic mass function	BMF	Number of galaxies with a given stellar plus cold gas mass.	Binned (on stellar mass).
Gas mass function	GMF	Number of galaxies with a given cold gas mass.	Binned (on stellar mass).
Baryonic Tully-Fisher	BTF	Maximum velocity of a galaxy as a function of baryonic (stellar plus cold gas) mass.	Scatter.
Specific star formation rate	sSFR	Specific star formation rate as a function of stellar mass.	Scatter.
Gas fraction	gas_frac	Fraction of baryons (stellar plus cold gas) in the form of cold gas as a function of stellar mass.	Scatter.
Mass metallicity	metallicity	Metallicity as a function of stellar mass.	Scatter.
Black hole bulge	bh_bulge	Mass of galaxy black hole as a function of galaxy bulge mass.	Scatter.
Quiescent galaxy population	quiescent	Fraction of galaxies that are quiescent as a function of stellar mass.	Binned (on stellar mass).
Bulge fraction	bulge_fraction	Fraction of stellar mass in the form of bulge/disk as a function of stellar mass.	Scatter.
Baryon fraction	baryon_fraction	Baryon fraction in each reservoir (cold, hot, stellar, ejected, intracluster, and black hole) as a function of FoF halo virial mass.	Binned (on FoF halo virial mass).
Reservoir mass	reservoirs	Amount of mass in each reservoir (cold, hot, stellar, ejected, intracluster, and black hole) as a function of FoF halo virial mass.	Scatter.
Spatial distribution	spatial	Spatial distribution of galaxies across the simulation box.	Scatter.

There are also a handful of toggles available to analyse properties over a number of redshifts.

Property	Plot Toggle Name	Description	Binning Type
Stellar mass function	SMF_history	Number of galaxies with a given stellar mass over multiple redshifts for each model.	Binned (on stellar mass).
Star formation rate density	SFRD_history	Total star formation rate density across entire simulation box as a function of redshift.	Single.
Stellar mass density	SMD_history	Total stellar mass density across entire simulation box as a function of redshift.	Single.

2.4.2 Property Types

sage-analysis supports three different property types.

binned

These properties are binned against another variable. For example, the stellar mass function counts the number of galaxy in stellar mass bins, the baryon fraction measures the fraction of baryons in each reservoir in friends-of-friend halo virial mass.

A property of this types requires the following fields:

```
<string denoting the name of the bins>: {
    "type": "binned",
    "bin_low": <float denoting the lower bound of the bins>,
    "bin_high": <float denoting the upper bound of the bins>,
    "bin_width": <float denoting the width of each bin>,
    "property_names": [<list of strings denoting the name of properties to be␣
↪initialized>],
}
```

For example, the stellar mass bins needed for default operation are initialized using:

```
"stellar_mass_bins": {
    "type": "binned",
    "bin_low": 8.0,
    "bin_high": 12.0,
    "bin_width": 0.1,
    "property_names": [
        "SMF", "red_SMF", "blue_SMF", "BMF", "GMF",
        "centrals_MF", "satellites_MF", "quiescent_galaxy_counts",
        "quiescent_centrals_counts", "quiescent_satellites_counts",
        "fraction_bulge_sum", "fraction_bulge_var",
        "fraction_disk_sum", "fraction_disk_var", "SMF_history",
    ],
}
```

The bins are accessed using `model.properties["<bin_name>"]` (e.g., `model.properties["stellar_mass_bins"]`) and the properties themselves as `model.properties["<propety_name>"]` (e.g., `model.properties["SMF"]` or `model.properties["quiescent_galaxy_counts"]`. Each property is initialized as a list of 0s.

scatter

To get a better picutre of some properties, it is useful to display them as a scatter plot. For example, the `metallicity` property shows the stellar mass vs metallicity for a number of randomly selected galaxies.

A property of this types requires the following fields:

```
<string denoting a unique name>: {
    "type": "scatter",
    "property_names": [<list of strings denoting the name of properties to be␣
↪initialized>],
}
```

For example, the default scatter properties are initialized using:

```
"scatter_properties": {
    "type": "scatter",
    "property_names": [
        "BTF_mass", "BTF_vel", "sSFR_mass", "sSFR_sSFR",
        "gas_frac_mass", "gas_frac", "metallicity_mass",
        "metallicity", "bh_mass", "bulge_mass", "reservoir_mvir",
        "reservoir_stars", "reservoir_cold", "reservoir_hot",
        "reservoir_ejected", "reservoir_ICS", "x_pos",
        "y_pos", "z_pos"
    ],
}
```

The properties are accessed as `model.properties["<propety_name>"]` (e.g., `model.properties["BTF_mass"]` or `model.properties["BTF_vel"]`. Each property is initialized as an empty list.

single

Finally, we may wish to summarize a property using a single number over an entire snapshot. For example, the stellar mass density is the sum of stellar mass divided by the volume for a single snapshot. This is useful for tracking properties over a number of snapshots as they can then be depicted as a line on a stellar mass density vs redshift plot.

A property of this types requires the following fields:

```
<string denoting a unique name>: {
    "type": "single",
    "property_names": [<list of strings denoting the name of properties to be␣
↪initialized>],
}
```

For example, the default single properties are initialized using:

```
"scatter_properties": {
    "type": "single",
    "property_names": ["SMD_history", "SFRD_history"],
}
```

The properties are accessed as `model.properties["<propety_name>"]` (e.g., `model.properties["SMD_history"]` or `model.properties["SFRD_history"]`. Each property is initialized with a value of `0.0`.

2.5 Analyzing Custom Properties

We show here a worked example of defining a custom property, writing a custom function to compute its value as the galaxies are processed, and then plotting the output. We refer to *Defining Custom Properties* for further detail on the available property types that can be defined.

2.5.1 Things To Be Aware Of When Analyzing Custom Properties

SAGE operates by allowing each processor to write to its own file as galaxies are evolved through cosmic time, with **sage-analysis** processing the galaxy properties for each of these files individually and separately. Consequently, each property should **MUST** have an entry in `model.properties["snapshot_<snapshot_number>]` that is carried **across** the different files.

For `binned` properties (see *Defining Custom Properties*), the entry in `model.properties["snapshot_<snapshot_number>"]` is a list that is continuously updated for each file. For example, the stellar mass function for snapshot 63 is stored in `model.properties["snapshot_63"]["SMF"]`. When the galaxies are processed for file 0, the stellar mass function at snapshot 63 is computed and added to `model.properties["snapshot_63"]["SMF"]`. These galaxies are discarded and new ones read in for file 1, with the stellar mass function at snapshot 63 for these new galaxies computed and added to `model.properties["snapshot_63"]["SMF"]`, and so on.

For `scatter` properties (see *Defining Custom Properties*), the entry in `model.properties["snapshot_<snapshot_number>"]` is an expanding list. For example, 10 galaxies at snapshot 63 from file 0 are appended to `model.properties["snapshot_63"]["BTF_mass"]`, 10 galaxies from file 1, 10 galaxies from file 2, etc.

For `single` properties (see *Defining Custom Properties*), the entry in `model.properties["snapshot_<snapshot_number>"]` is a single number that is adjusted for each file. For example, the sum of stellar mass divided by the volume at snapshot 63 in file 0 is added to `model.properties["snapshot_63"]["SMD_history"]`. The stellar mass density at snapshot 63 in file 1 is then added, and so on.

2.5.2 Worked Examples

We show here how to compute the number of particles in the background FoF halo (as a binned property), the mass of hot gas as a function of cold gas (as a scatter property), and the time of last major merger (as a single property) tracked over redshift.

Number of Particles

Firstly, we need to tell **sage_analysis** the properties that we are analyzing and plotting.

```
plot_toggles = {"halo_parts": True}
```

Now, lets define the properties that will be used to store all of our results. As outlined in *Defining Custom Properties*, each property type is defined in a slightly different manner.

```
galaxy_properties_to_analyze = {
    "number_particles_bins": {
        "type": "binned",
        "bin_low": 0,
        "bin_high": 5,
```

(continues on next page)

(continued from previous page)

```
        "bin_width": 0.1,
        "property_names": ["particle_mass_function"],
    },
```

Next, we need the function that will compute the values relevant for this property.

```
# Saved in ``my_calculations_functions.py``.
from typing import Any

import numpy as np

from sage_analysis.model import Model

def calc_halo_parts(model: Model, gals: Any, snapshot: int) -> None:

    non_zero_parts = np.where(gals["Len"][:] > 0)[0]
    halo_len = np.log10(gals["Len"][:][non_zero_parts])  # Ensure that the data is␣
↪the same units as bins.
    gals_per_bin, _ = np.histogram(halo_len, bins=model.bins["number_particles_bins"])

    # Update properties to keep persistent across files.
    model.properties[f"snapshot_{snapshot}"]["particle_mass_function"] += gals_per_bin
```

Then, the function that will plot the results.

```
# Save as ``my_plot_functions.py``.
from typing import List

from sage_analysis.model import Model

import matplotlib
import matplotlib.pyplot as plt
import numpy as np

colors = ["r", "g", "b", "c"]
linestyles = ["--", "-.", "."]
markers = ["x", "o"]


def plot_halo_parts(
    models: List[Model], snapshots: List[List[int]], plot_output_path: str, plot_
↪output_format: str = "png",
) -> matplotlib.figure.Figure:

    fig = plt.figure()
    ax = fig.add_subplot(111)

    # Go through each of the models and plot.
    for model_num, (model, model_snapshots) in enumerate(zip(models, snapshots)):

        # Set the x-axis values to be the centre of the bins.
        bin_widths = model.bins["number_particles_bins"][1::] - model.bins["number_
↪particles_bins"][0:-1]
        bin_middles = model.bins["number_particles_bins"][:-1] + bin_widths

        # Colour will be used for the snapshot, linestyle for the model.
        ls = linestyles[model_num]
```

(continues on next page)

(continued from previous page)

```
        label = model.label

        for snapshot_num, snapshot in enumerate(model_snapshots):
            color = colors[snapshot_num]
            ax.plot(
                bin_middles,
                model.properties[f"snapshot_{snapshot}"]["particle_mass_function"],
                color=color,
                ls=ls,
                label=f"{label} - z = {model._redshifts[snapshot]:.2f}",
            )

    ax.set_xlabel(r"$\log_{10} Number Particles in Halo$")
    ax.set_ylabel(r"$N$")

    ax.set_yscale("log", nonpositive="clip")
    ax.legend()

    fig.tight_layout()

    output_file = f"{plot_output_path}particles_in_halos.{plot_output_format}"
    fig.savefig(output_file)
    print(f"Saved file to {output_file}")
    plt.close()

    return fig
```

With everything defined and our functions written, we are now ready to execute **sage-analysis** itself.

```
from sage_analysis.galaxy_analysis import GalaxyAnalysis
from sage_analysis.utils import generate_func_dict

par_fnames = ["/home/Desktop/sage-model/input/millennium.ini"]

# Generate the dictionaries with our custom functions.
calculation_functions = generate_func_dict(plot_toggles, __name__, "calc_")
plot_functions = generate_func_dict(plot_toggles, __name__, "plot_")

# We're good to go now!
galaxy_analysis = GalaxyAnalysis(
    par_fnames,
    plot_toggles=plot_toggles,
    galaxy_properties_to_analyze=galaxy_properties_to_analyze,
    history_redshifts=history_redshifts,
    calculation_functions=calculation_functions,
    plot_functions=plot_functions
)

galaxy_analysis.analyze_galaxies()
galaxy_analysis.generate_plots()
```

Mass of Hot Gas as Function of Cold Gas

```
plot_toggles = {"hot_cold": True}
galaxy_properties_to_analyze = {
```

(continues on next page)

(continued from previous page)

```
    "hot_cold_scatter": {
        "type": "scatter",
        "property_names": ["hot_gas", "cold_gas"],
    },
}


def calc_hot_cold(model: Model, gals: Any, snapshot: int) -> None:

    non_zero_stellar = np.where(gals["StellarMass"][:] > 0.0)[0]

    # Remember that mass is kept in units of 1.0e10 Msun/h. Convert to log10(Msun).
    hot_gas_mass = np.log10(gals["HotGas"][:][non_zero_stellar] * 1.0e10 / model.
↪hubble_h)
    cold_gas_mass = np.log10(gals["ColdGas"][:][non_zero_stellar] * 1.0e10 / model.
↪hubble_h)

    # Append to properties to keep persistent across files.
    model.properties[f"snapshot_{snapshot}"]["hot_gas"] = np.append(
        model.properties[f"snapshot_{snapshot}"]["hot_gas"], hot_gas_mass
    )

    model.properties[f"snapshot_{snapshot}"]["cold_gas"] = np.append(
        model.properties[f"snapshot_{snapshot}"]["cold_gas"], cold_gas_mass
    )


def plot_hot_cold(
    models: List[Model], snapshots: List[List[int]], plot_output_path: str, plot_
↪output_format: str = "png",
) -> matplotlib.figure.Figure:

    fig = plt.figure()
    ax = fig.add_subplot(111)

    # Go through each of the models and plot.
    for model_num, (model, model_snapshots) in enumerate(zip(models, snapshots)):

        # Colour will be used for the snapshot, marker style for the model.
        marker = markers[model_num]
        label = model.label

        for snapshot_num, snapshot in enumerate(model_snapshots):
            color = colors[snapshot_num]

            ax.scatter(
                model.properties[f"snapshot_{snapshot}"]["cold_gas"],
                model.properties[f"snapshot_{snapshot}"]["hot_gas"],
                marker=marker,
                s=1,
                color=color,
                alpha=0.5,
                label=f"{label} - z = {model._redshifts[snapshot]:.2f}",
            )

    ax.set_xlabel(r"$\log_{10} Cold Gas Mass [M_\odot]$")
    ax.set_ylabel(r"$\log_{10} Hot Gas Mass [M_\odot]$")
```

(continues on next page)

(continued from previous page)

```
    ax.legend()

    fig.tight_layout()

    output_file = f"{plot_output_path}hot_cold.{plot_output_format}"
    fig.savefig(output_file)
    print(f"Saved file to {output_file}")
    plt.close()

    return fig
```

Defining the Properties

Now, lets define the properties that will be used to store all of our results. As outlined in *Defining Custom Properties*, each property type is defined in a slightly different manner.

```
galaxy_properties_to_analyze = {
    "number_particles_bins": {
        "type": "binned",
        "bin_low": 0,
        "bin_high": 5,
        "bin_width": 0.1,
        "property_names": ["particle_mass_function"],
    },
    "hot_cold_scatter": {
        "type": "scatter",
        "property_names": ["hot_gas", "cold_gas"],
    },
    "time_major_merger": {
        "type": "single",
        "property_names": ["sum_time_since_major_merger", "var_time_since_major_merger
↪", "num_galaxies"]
    }
}
```

For the first property, we have used log-spaced bins. For the last property, we will track the sum and variance of the time since major merger alongside the total number of galaxies. Then, when we plot, we can compute the mean and show the mean plus variance trend.

Tracking a Property Over Redshift

We want to track the time since major merger over redshift explicitly. To do so, we need to specify the redshifts we wish to track it over, `history_redshifts`.

```
history_redshifts = {"major_merger_history": "All"}
```

Note: The key names in this dictionary must exactly match the key name in `plot_toggles`.

Defining the Functions

The `GalaxyAnalysis` constructor accepts two key parameters: *:calculation_functions* and `plot_functions`. From these two dictionaries, the exact functions that need to be run for each galaxy file and the functions that produce the final plots are defined. Under the hood, **sage-analysis** operates by looping over `calculation_functions` and calling the constituent functions with the galaxies loaded for each file. To plot, each function in `plot_functions` is called using the model data that has been previously analyzed.

Hence, to define your own custom properties, we must first update the `calculation_functions` and `plot_functions` and pass it to the `GalaxyAnalysis` constructor.

Let's write the functions that will define the calculation functions that will be saved to module `my_calculation_functions.py`. These will use our properties defined above to keep the values across different files.

```python
# Saved in ``my_calculations_functions.py``.
from typing import Any

import numpy as np

from sage_analysis.model import Model

def calc_halo_parts(model: Model, gals: Any, snapshot: int) -> None:

    non_zero_parts = np.where(gals["Len"][:] > 0)[0]
    halo_len = np.log10(gals["Len"][:][non_zero_parts])  # Ensure that the data is␣
↪the same units as bins.
    gals_per_bin, _ = np.histogram(halo_len, bins=model.bins["number_particles_bins"])

    # Update properties to keep persistent across files.
    model.properties[f"snapshot_{snapshot}"]["particle_mass_function"] += gals_per_bin


def calc_hot_cold(model: Model, gals: Any, snapshot: int) -> None:

    non_zero_stellar = np.where(gals["StellarMass"][:] > 0.0)[0]

    # Remember that mass is kept in units of 1.0e10 Msun/h. Convert to log10(Msun).
    hot_gas_mass = np.log10(gals["HotGas"][:][non_zero_stellar] * 1.0e10 / model.
↪hubble_h)
    cold_gas_mass = np.log10(gals["ColdGas"][:][non_zero_stellar] * 1.0e10 / model.
↪hubble_h)

    # Append to properties to keep persistent across files.
    model.properties[f"snapshot_{snapshot}"]["hot_gas"] = np.append(
        model.properties[f"snapshot_{snapshot}"]["hot_gas"], hot_gas_mass
    )

    model.properties[f"snapshot_{snapshot}"]["cold_gas"] = np.append(
        model.properties[f"snapshot_{snapshot}"]["cold_gas"], cold_gas_mass
    )


def calc_major_merger_history(model: Model, gals: Any, snapshot: int) -> None:

    non_zero_stellar = np.where(gals["StellarMass"][:] > 0.0)[0]

    time_since_major_merger = gals["TimeOfLastMajorMerger"][:][non_zero_stellar]
```

(continues on next page)

(continued from previous page)

```
    # A galaxy that has not experienced a major merger will have a value of -1. Lets␣
↪filter these out.
    time_since_major_merger = time_since_major_merger[time_since_major_merger > 0.0]

    # We will handle dividing out the number of galaxies and the number of samples (i.
↪e., number of files) when it
    # comes time to plot.
    model.properties[f"snapshot_{snapshot}"]["sum_time_since_major_merger"] += np.
↪sum(time_since_major_merger)
    model.properties[f"snapshot_{snapshot}"]["var_time_since_major_merger"] += np.
↪var(time_since_major_merger)
    model.properties[f"snapshot_{snapshot}"]["num_galaxies"] += len(time_since_major_
↪merger)
```

With our calculation functions defined, we now need to define the plot functions. These functions will be used by **sage-analysis** to generate the plots themselves. We will save these functions to the module `my_plot_functions.py`.

```
# Save as ``my_plot_functions.py``.
from typing import List

from sage_analysis.model import Model

import matplotlib
import matplotlib.pyplot as plt
import numpy as np

colors = ["r", "g", "b", "c"]
linestyles = ["--", "-.", "."]
markers = ["x", "o"]


def plot_halo_parts(
    models: List[Model], snapshots: List[List[int]], plot_output_path: str, plot_
↪output_format: str = "png",
) -> matplotlib.figure.Figure:

    fig = plt.figure()
    ax = fig.add_subplot(111)

    # Go through each of the models and plot.
    for model_num, (model, model_snapshots) in enumerate(zip(models, snapshots)):

        # Set the x-axis values to be the centre of the bins.
        bin_widths = model.bins["number_particles_bins"][1::] - model.bins["number_
↪particles_bins"][0:-1]
        bin_middles = model.bins["number_particles_bins"][:-1] + bin_widths

        # Colour will be used for the snapshot, linestyle for the model.
        ls = linestyles[model_num]
        label = model.label

        for snapshot_num, snapshot in enumerate(model_snapshots):
            color = colors[snapshot_num]
            ax.plot(
```

(continues on next page)

(continued from previous page)

```
                bin_middles,
                model.properties[f"snapshot_{snapshot}"]["particle_mass_function"],
                color=color,
                ls=ls,
                label=f"{label} - z = {model._redshifts[snapshot]:.2f}",
            )

    ax.set_xlabel(r"$\log_{10} Number Particles in Halo$")
    ax.set_ylabel(r"$N$")

    ax.set_yscale("log", nonpositive="clip")
    ax.legend()

    fig.tight_layout()

    output_file = f"{plot_output_path}particles_in_halos.{plot_output_format}"
    fig.savefig(output_file)
    print(f"Saved file to {output_file}")
    plt.close()

    return fig


def plot_hot_cold(
    models: List[Model], snapshots: List[List[int]], plot_output_path: str, plot_
↪output_format: str = "png",
) -> matplotlib.figure.Figure:

    fig = plt.figure()
    ax = fig.add_subplot(111)

    # Go through each of the models and plot.
    for model_num, (model, model_snapshots) in enumerate(zip(models, snapshots)):

        # Colour will be used for the snapshot, marker style for the model.
        marker = markers[model_num]
        label = model.label

        for snapshot_num, snapshot in enumerate(model_snapshots):
            color = colors[snapshot_num]

            ax.scatter(
                model.properties[f"snapshot_{snapshot}"]["cold_gas"],
                model.properties[f"snapshot_{snapshot}"]["hot_gas"],
                marker=marker,
                s=1,
                color=color,
                alpha=0.5,
                label=f"{label} - z = {model._redshifts[snapshot]:.2f}",
            )

    ax.set_xlabel(r"$\log_{10} Cold Gas Mass [M_\odot]$")
    ax.set_ylabel(r"$\log_{10} Hot Gas Mass [M_\odot]$")

    ax.legend()

    fig.tight_layout()
```

(continues on next page)

(continued from previous page)

```
    output_file = f"{plot_output_path}hot_cold.{plot_output_format}"
    fig.savefig(output_file)
    print(f"Saved file to {output_file}")
    plt.close()

    return fig


def plot_major_merger_history(
    models: List[Model], snapshots: List[List[int]], plot_output_path: str, plot_
↪output_format: str = "png",
) -> matplotlib.figure.Figure:

    fig = plt.figure()
    ax = fig.add_subplot(111)

    for (model_num, model) in enumerate(models):

        label = model.label
        color = colors[model_num]
        linestyle = linestyles[model_num]
        marker = markers[model_num]

        sum_time_since_major_merger = np.array(
            [model.properties[f"snapshot_{snap}"]["sum_time_since_major_merger"] for␣
↪snap in range(len(model.redshifts))]
        )
        var_time_since_major_merger = np.array(
            [model.properties[f"snapshot_{snap}"]["var_time_since_major_merger"] for␣
↪snap in range(len(model.redshifts))]
        )
        num_galaxies = np.array(
            [model.properties[f"snapshot_{snap}"]["num_galaxies"] for snap in␣
↪range(len(model.redshifts))]
        )
        redshifts = model.redshifts

        # mean =  sum / number of samples.
        mean_time_since_major_merger = sum_time_since_major_merger / num_galaxies

        # Need to divide out the number of samples for the variance. This is the␣
↪number of files that we analyzed.
        var_time_since_major_merger /= (model.last_file_to_analyze - model.first_file_
↪to_analyze + 1)

        # All snapshots are initialized with zero values, we only want to plot those␣
↪non-zero values.
        non_zero_inds = np.where(mean_time_since_major_merger > 0.0)[0]

        # Only use a line if we have enough snapshots to plot.
        if len(non_zero_inds) > 20:
            ax.plot(
                redshifts[non_zero_inds],
                mean_time_since_major_merger[non_zero_inds],
                label=label,
                color=color,
```

(continues on next page)

(continued from previous page)

```
                ls=linestyle
            )
        else:
            ax.scatter(
                redshifts[non_zero_inds],
                mean_time_since_major_merger[non_zero_inds],
                label=label,
                color=color,
                marker=marker,
            )

    ax.set_xlabel(r"$\mathrm{redshift}$")
    ax.set_ylabel(r"$Time Since Last Major Merger [Myr]$")

    ax.set_xlim([0.0, 8.0])
    #ax.set_ylim([-3.0, -0.4])

    ax.xaxis.set_minor_locator(plt.MultipleLocator(1))
    #ax.yaxis.set_minor_locator(plt.MultipleLocator(0.5))

    ax.legend()

    fig.tight_layout()

    output_file = f"{plot_output_path}time_since_last_major_merger.{plot_output_
↪format}"
    fig.savefig(output_file)
    print("Saved file to {0}".format(output_file))
    plt.close()

    return fig
```

Putting it Together

With everything defined and our functions written, we are now ready to execute **sage-analysis** itself.

```
import my_calculation_functions, my_plot_functions

from sage_analysis.galaxy_analysis import GalaxyAnalysis
from sage_analysis.utils import generate_func_dict

par_fnames = ["/home/Desktop/sage-model/input/millennium.ini"]

# Generate the dictionaries with our custom functions.
calculation_functions = generate_func_dict(plot_toggles, "my_calculation_functions",
↪"calc_")
plot_functions = generate_func_dict(plot_toggles, "my_plot_functions", "plot_")

# We're good to go now!
galaxy_analysis = GalaxyAnalysis(
    par_fnames,
    plot_toggles=plot_toggles,
    galaxy_properties_to_analyze=galaxy_properties_to_analyze,
    history_redshifts=history_redshifts,
    calculation_functions=calculation_functions,
```

(continues on next page)

(continued from previous page)

```
    plot_functions=plot_functions
)

galaxy_analysis.analyze_galaxies()
galaxy_analysis.generate_plots()
```

And these are our plots that are generated…

2.6 Analyzing Custom Data

2.7 sage_analysis.GalaxyAnalysis

class sage_analysis.**GalaxyAnalysis**(*sage_parameter_fnames: List[str], plot_toggles: Optional[Dict[str, bool]] = None, sage_output_formats: Optional[List[str]] = None, labels: Optional[List[str]] = None, first_files_to_analyze: Optional[List[int]] = None, last_files_to_analyze: Optional[List[int]] = None, num_sage_output_files: Optional[List[int]] = None, output_format_data_classes_dict: Optional[Dict[str, Any]] = None, random_seeds: Optional[List[int]] = None, history_redshifts: Optional[Dict[str, Union[List[float], str]]] = None, calculation_functions: Optional[Dict[str, Tuple[Callable, Dict[str, Any]]]] = None, plot_functions: Optional[Dict[str, Tuple[Callable, Dict[str, Any]]]] = None, galaxy_properties_to_analyze: Optional[Dict[str, Dict[str, Union[str, List[str]]]]] = None, plots_that_need_smf: Optional[List[str]] = None, IMFs: Optional[List[str]] = None*)

Handles the ingestion, analysis, and plotting of **SAGE** galaxy outputs.

__init__(*sage_parameter_fnames: List[str], plot_toggles: Optional[Dict[str, bool]] = None, sage_output_formats: Optional[List[str]] = None, labels: Optional[List[str]] = None, first_files_to_analyze: Optional[List[int]] = None, last_files_to_analyze: Optional[List[int]] = None, num_sage_output_files: Optional[List[int]] = None, output_format_data_classes_dict: Optional[Dict[str, Any]] = None, random_seeds: Optional[List[int]] = None, history_redshifts: Optional[Dict[str, Union[List[float], str]]] = None, calculation_functions: Optional[Dict[str, Tuple[Callable, Dict[str, Any]]]] = None, plot_functions: Optional[Dict[str, Tuple[Callable, Dict[str, Any]]]] = None, galaxy_properties_to_analyze: Optional[Dict[str, Dict[str, Union[str, List[str]]]]] = None, plots_that_need_smf: Optional[List[str]] = None, IMFs: Optional[List[str]] = None*)

Parameters

- **sage_parameter_fnames** (*list of strings*) – The name of the **SAGE** parameter files that are to be analyzed. These are the `.ini` files used to generate the galaxy files. The length of this variable is equal to the number of models to be analyzed.
- **plot_toggles** (*dict [str, bool], optional*) – Specifies which properties should be analyzed and plotted.

  If not specified, uses

```
default_plot_toggles = {
    "SMF" : True,
    "BMF" : True,
    "GMF" : True,
    "BTF" : True,
    "sSFR" : True,
    "gas_fraction" : True,
    "metallicity" : True,
    "bh_bulge" : True,
    "quiescent" : True,
    "bulge_fraction" : True,
    "baryon_fraction" : True,
    "reservoirs" : True,
    "spatial" : True,
    "SMF_history": False,
    "SFRD_history": False,
    "SMD_history": False,
}
```

- **sage_output_formats** (*list of strings, optional*) – The output formats of each **SAGE** model being analyzed. Each value here **MUST** have a corresponding entry in `output_format_data_classes_dict`. The length of this variable is equal to the number of models to be analyzed.

  If not specified, will use the `OutputFormat` entry from the respective **SAGE** parameter file.

- **labels** (*list of strings, optional*) – The labels to be used in the legend for each model. The length of this variable is equal to the number of models to be analyzed.

  If not specified, will use the `FileNameGalaxies` entry from the respective **SAGE** parameter file.

- **first_files_to_analyze, last_files_to_analyze** (*list of ints, optional-ish*) – The output **SAGE** files to be analyzed. This is an inclusive range, with the output files analyzed ranging from `[first_file_to_analyze, last_file_to_analyze]` for each model. The length of this variable is equal to the number of models to be analyzed.

  If the corresponding entry in `sage_output_format` is `sage_binary` (whether passed explicitly or read from `sage_file`), these two variables **MUST** be specified. Otherwise, if not specified, will analyze **ALL** output HDF5 files.

- **num_sage_output_files** (*list of ints, optional-ish*) – Specifies the number of output files that were generated by running **SAGE**. This will generally be equal to the number of processors used to run **SAGE** and can be different to the range specified by `[first_file_to_analyze, last_file_to_analyze]`.

  If the corresponding entry in `sage_output_format` is `sage_binary` (whether passed explicitly or read from `sage_file`), this **MUST** be specified. Otherwise, this variable is **NOT** used.

- **output_format_data_classes_dict** (*dict [string, class], optional*) – A dictionary that maps the output format name to the corresponding data class. Each value in `sage_output_formats` **MUST** have an entry in this dictionary.

  If not specified, will use a default value `output_format_data_classes_dict = {"sage_binary": SageBinaryData, "sage_hdf5": SageHdf5Data}`.

- **random_seeds** (*list of ints, optional*) – The values to seed the random number generator for each model. If the value is `None`, then the generator is seeded using the `np.random.`

seed() method. The length of this variable is equal to the number of models to be analyzed.

If not specified, uses None for each model (i.e., no predetermined seed).

- **history_redshifts** (*dict [string, string or list of floats], optional*) – Specifies which redshifts should be analyzed for properties and plots that are tracked over time. The keys here **MUST** have the same name as in plot_toggles.

  If the value of the entry is "All", then all snapshots will be analyzed. Otherwise, will search for the closest snapshots to the requested redshifts.

  If not specified, uses

```
history_redshifts = {
    "SMF_history": "All",
    "SMD_history": "All",
    "SFRD_history": "All",
}
```

- **calculation_functions** (*dict [string, tuple(function, dict[string, variable])], optional*) – A dictionary of functions that are used to compute the properties of galaxies being analyzed. Here, the string is the name of the plot toggle (e.g., "SMF"), the value is a tuple containing the function itself (e.g., calc_SMF()), and another dictionary which specifies any optional keyword arguments to that function with keys as the name of variable (e.g., "calc_sub_populations") and values as the variable value (e.g., True).

  The functions in this dictionary are called for all files analyzed and **MUST** have a signature func(model, gals, snapshot, optional_keyword_arguments). This dict can be generated using *generate_func_dict()*.

  If not specified, will use the functions found in *example_calcs*, filtered to ensure that only those functions necessary to plot the plots specified by plot_toggles are run.

- **plot_functions** (*dict [string, tuple(function, dict[string, variable])], optional*) – A dictionary of functions that are used to plot the properties of galaxies being analyzed. Here, the string is the name of the function (e.g., "plot_SMF"), the value is a tuple containing the function itself (e.g., plot_SMF()), and another dictionary which specifies any optional keyword arguments to that function with keys as the name of variable (e.g., "plot_sub_populations") and values as the variable value (e.g., True).

  The functions in this dictionary are called for all files analyzed and **MUST** have a signature func(models, snapshots, plot_helper, optional_keyword_arguments). This dict can be generated using *generate_func_dict()*.

  If not specified, will use the functions found in *example_plots*, filtered to ensure that only those functions necessary to plot the plots specified by plot_toggles are run.

- **galaxy_properties_to_analyze** (*dict [string, dict[str, float or str or list of strings]], optional*) – The galaxy properties that are used when running calculation_functions. The properties initialized here will be accessible through model.properties["property_name"].

  This variable is a nested dictionary with the outer dictionary specifying the name of the bins (if the properties will be binned), or a unique name otherwise.

  The inner dictionary has a number of fields that depend upon the type of property. We support properties being either binned against a property (e.g., the stellar or halo mass functions are binned on stellar/halo mass), plotted as x-vs-y scatter plots (e.g., specific star formation rate vs stellar mass for 1000 galaxies), or as a single value (e.g., the stellar mass density).

For binned against a property, the key/value pairs are: `"type": "binned"`, `bin_low: The lower bound of the bin (float)`, `bin_high: The upper bound of the bin (float)`, `bin_width: The width of the bin (float)`, `property_names: A list of strings denoting the properties to be initialised`. The bin values are all initialized as 0.0.

For properties to be plotted as x-vs-y scatter plots, the key/value pairs are: `"type": "scatter"`, `property_names: A list of strings denoting the properties to be initialised`. All properties are initialized as empty lists.

For properties that are single values, the key/value pairs are: `"type": "single"`, `property_names: A list of strings denoting the properties to be initialised`. All properties are initialized with a value of 0.0.

If not specified, uses

```
default_galaxy_properties_to_analyze = {
    "stellar_mass_bins": {
        "type": "binned",
        "bin_low": 8.0,
        "bin_high": 12.0,
        "bin_width": 0.1,
        "property_names": [
            "SMF", "red_SMF", "blue_SMF", "BMF", "GMF",
            "centrals_MF", "satellites_MF", "quiescent_galaxy_
↪counts",
            "quiescent_centrals_counts", "quiescent_satellites_
↪counts",
            "fraction_bulge_sum", "fraction_bulge_var",
            "fraction_disk_sum", "fraction_disk_var", "SMF_history
↪",
        ],
    },
    "halo_mass_bins": {
        "type": "binned",
        "bin_low": 10.0,
        "bin_high": 14.0,
        "bin_width": 0.1,
        "property_names": ["fof_HMF"] + [f"halo_{component}_
↪fraction_sum"
            for component in ["baryon", "stars", "cold", "hot",
↪"ejected", "ICS", "bh"]
        ],
    },
    "scatter_properties": {
        "type": "scatter",
        "property_names": [
            "BTF_mass", "BTF_vel", "sSFR_mass", "sSFR_sSFR",
            "gas_frac_mass", "gas_frac", "metallicity_mass",
            "metallicity", "bh_mass", "bulge_mass", "reservoir_
↪mvir",
            "reservoir_stars", "reservoir_cold", "reservoir_hot",
            "reservoir_ejected", "reservoir_ICS", "x_pos",
            "y_pos", "z_pos"
        ],
    },
```

(continues on next page)

(continued from previous page)

```
    "single_properties": {
        "type": "single",
        "property_names": ["SMD_history", "SFRD_history"],
    },
}
```

- **plots_that_need_smf** (*list of strings, optional*) – The plot toggles that require the stellar mass function to be properly computed and analyzed. For example, plotting the quiescent fraction of galaxies requires knowledge of the total number of galaxies. The strings here must **EXACTLY** match the keys in `plot_toggles`.

  If not specified, uses a default value of `["SMF", "quiescent", "bulge_fraction", "SMF_history"]`.

- **IMFs** (list of strings, optional, `{"Chabrier", "Salpeter"}`) – The initial mass functions used during the analysis of the galaxies. This is used to shift the observational data points. The length of this variable is equal to the number of models to be analyzed.

  If not specified, uses a `"Chabrier"` IMF for each model.

analyze_galaxies(*snapshots: Optional[List[List[Union[str, int]]]] = None*, *redshifts: Optional[List[List[Union[float, str]]]] = None*, *analyze_history_snapshots: bool = True*) → None

Analyses the galaxies of the initialized *models*. These attributes will be updated directly, with the properties accessible via `GalaxyAnalysis.models[<model_num>].properties[<snapshot>][<property_name>]`.

Also, all snapshots required to track the properties over time (as specified by `_history_snaps_to_loop`) will be analyzed, unless `analyze_history_snapshots` is `False`.

Parameters

- **snapshots** (*nested list of ints or string, optional*) – The snapshots to analyze for each model. If both this variable and `redshifts` are not specified, uses the highest snapshot (i.e., lowest redshift) as dictated by the *redshifts* attribute from the parameter file read for each model.

  If an entry if `"All"`, then all snapshots for that model will be analyzed.

  The length of the outer list **MUST** be equal to *num_models*.

  ### Notes

  If `analyze_history_snapshots` is `True`, then the snapshots iterated over will be the unique combination of the snapshots required for history snapshots and those specified by this variable.

  Warning: Only **ONE** of `snapshots` and `redshifts` can be specified.

- **redshifts** (*nested list of ints, optional*) – The redshift to analyze for each model. If both this variable and `snapshots` are not specified, uses the highest snapshot (i.e., lowest redshift) as dictated by the *redshifts* attribute from the parameter file read for each model.

The snapshots selected for analysis will be those that result in the redshifts closest to those requested. If an entry if `"All"`, then all snapshots for that model will be analyzed.

The length of the outer list **MUST** be equal to *`num_models`*.

Notes

If `analyze_history_snapshots` is `True`, then the snapshots iterated over will be the unique combination of the snapshots required for history snapshots and those specified by this variable.

> **Warning:** Only **ONE** of `snapshots` and `redshifts` can be specified.

- **analyze_history_snapshots** (*bool, optional*) – Specifies whether the snapshots required to analyze the properties tracked over time (e.g., stellar mass or star formation rate density) should be iterated over. If not specified, then only `snapshot` will be analyzed.

Notes

If you wish to analyze different properties to when you initialized an instance of *`GalaxyAnalysis`*, you **MUST** re-initialize another instance. Otherwise, the properties will be non-zeroed and not initialized correctly.

ValueError Thrown if **BOTH** `snapshots` and `redshifts` are specified.

`generate_plots`(*snapshots: Optional[List[List[Union[str, int]]]] = None, redshifts: Optional[List[List[Union[float, str]]]] = None, plot_helper: Optional[sage_analysis.plot_helper.PlotHelper] = None*) → Optional[List[matplotlib.figure.Figure]]

Generates the plots for the *`models`* being analyzed. The plots to be created are defined by the values of *`plot_toggles`* specified when an instance of *`GalaxyAnalysis`* was initialized. If you wish to analyze different properties or create different plots, you **MUST** initialize another instance of *`GalaxyAnalysis`* with the new values for *`plot_toggles`* (ensuring that values of `calcuations_functions` and `plot_functions` are updated if using non-default values for `plot_toggles`).

This method should be run after analysing the galaxies using **:py:method:‘~analyze_galaxies‘**.

Parameters

- **snapshots** (*nested list of ints or string, optional*) – The snapshots to plot for each model. If both this variable and `redshifts` are not specified, uses the highest snapshot (i.e., lowest redshift) as dictated by the *`redshifts`* attribute from the parameter file read for each model.

  If an entry if `"All"`, then all snapshots for that model will be analyzed.

  The length of the outer list **MUST** be equal to *`num_models`*.

  For properties that aren't analyzed over redshift, the snapshots for each model will be plotted on each figure. For example, if we are plotting a single model, setting this variable to `[[63, 50]]` will give results for snapshot 63 and 50 on each figure. For some plots (e.g., those properties that are scatter plotted), this is undesirable and one should instead iterate over single snapshot values instead.

Notes

If `analyze_history_snapshots` is `True`, then the snapshots iterated over will be the unique combination of the snapshots required for history snapshots and those specified by this variable.

> **Warning:** Only **ONE** of `snapshots` and `redshifts` can be specified.

- **redshifts** (*nested list of ints, optional*) – The redshift to plot for each model. If both this variable and `snapshots` are not specified, uses the highest snapshot (i.e., lowest redshift) as dictated by the *`redshifts`* attribute from the parameter file read for each model.

  The snapshots selected for analysis will be those that result in the redshifts closest to those requested. If an entry if `"All"`, then all snapshots for that model will be analyzed.

  The length of the outer list **MUST** be equal to *`num_models`*.

  > **Warning:** Only **ONE** of `snapshots` and `redshifts` can be specified.

- **plot_helper** (`PlotHelper`, optional) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.

  If not specified, then will initialize a default instance of `PlotHelper`. Refer to the `PlotHelper` documentation for a list of default attributes.

Returns

- *None* – Returned if *`plot_toggles`* is an empty dictionary.
- *figs* – The figures generated by the *`plot_functions`* functions.

`history_redshifts`

Specifies which redshifts should be analyzed for properties and plots that are tracked over time. The keys here **MUST** correspond to the keys in *`plot_toggles`*. If the value of the entry is `"All"`, then all snapshots will be analyzed. Otherwise, will search for the closest snapshots to the requested redshifts.

Type dict [string, string or list of floats]

`models`

The *`Model`* s being analyzed.

Type list of *`Model`* class instances

`num_models`

The number of models being analyzed.

Type int

`output_format_data_classes_dict`

A dictionary that maps the output format name to the corresponding data class.

Type dict [str, class]

`plot_functions`

A dictionary of functions that are used to plot the properties of galaxies being analyzed. Here, the outer key is the name of the corresponding plot toggle (e.g., `"SMF"`), the value is a tuple containing the function itself (e.g., `plot_SMF()`), and another dictionary which specifies any optional keyword arguments to that function with keys as the name of variable (e.g., `"plot_sub_populations"`) and values as the variable value (e.g., `True`).

The functions in this dictionary are called for all files analyzed and **MUST** have a signature `func(Models, snapshot, plot_helper, plot_output_format, optional_keyword_arguments)`. This dict can be generated using *`generate_func_dict()`*.

Type dict [str, tuple(function, dict [str, any])]

`plot_toggles`
Specifies which properties should be analyzed and plotted.

Type dict [str, bool]

2.8 sage_analysis.Model

This module contains the `Model` class. The `Model` class contains all the data paths, cosmology etc for calculating galaxy properties.

To read **SAGE** data, we make use of specialized Data Classes (e.g., *`SageBinaryData`* and:py:class:~*sage_analysis.sage_hdf5.SageHdf5Data*). We refer to ../user/data_class for more information about adding your own Data Class to ingest data.

To calculate (and plot) extra properties from the **SAGE** output, we refer to ../user/calc.rst and ../user/plotting.rst.

class `sage_analysis.model.`**`Model`**(*sage_file: str, sage_output_format: Optional[str], label: Optional[str], first_file_to_analyze: int, last_file_to_analyze: int, num_sage_output_files: Optional[int], random_seed: Optional[int], IMF: str, plot_toggles: Dict[str, bool], plots_that_need_smf: List[str], sample_size: int = 1000, sSFRcut: float = -11.0*)

Handles all the galaxy data (including calculated properties) for a `SAGE` model.

The ingestion of data is handled by inidivudal Data Classes (e.g., *`SageBinaryData`* and *`SageHdf5Data`*). We refer to ../user/data_class for more information about adding your own Data Class to ingest data.

`__init__`(*sage_file: str, sage_output_format: Optional[str], label: Optional[str], first_file_to_analyze: int, last_file_to_analyze: int, num_sage_output_files: Optional[int], random_seed: Optional[int], IMF: str, plot_toggles: Dict[str, bool], plots_that_need_smf: List[str], sample_size: int = 1000, sSFRcut: float = -11.0*)

Sets the galaxy path and number of files to be read for a model. Also initialises the plot toggles that dictates which properties will be calculated.

Parameters

- **label** (*str, optional*) – The label that will be placed on the plots for this model. If not specified, will use `FileNameGalaxies` read from `sage_file`.
- **sage_output_format** (*str, optional*) – If not specified will use the `OutputFormat` read from `sage_file`.
- **num_sage_output_files** (*int, optional*) – Specifies the number of output files that were generated by running **SAGE**. This can be different to the range specified by [first_file_to_analyze, last_file_to_analyze].

Notes

This variable only needs to be specified if `sage_output_format` is `sage_binary`.

- **sample_size** (*int, optional*) – Specifies the length of the *properties* attributes stored as 1-dimensional `ndarray`. These *properties* are initialized using *init_scatter_properties()*.
- **sSFRcut** (*float, optional*) – The specific star formation rate above which a galaxy is flagged as "star forming". Units are log10.

calc_properties (*calculation_functions*, *gals*, *snapshot: int*)

Calculates galaxy properties for a single file of galaxies.

Parameters

- **calculation_functions** (*dict [string, function]*) – Specifies the functions used to calculate the properties. All functions in this dictionary are called on the galaxies. The function signature is required to be `func(Model, gals)`
- **gals** (exact format given by the *Model* Data Class.) – The galaxies for this file.
- **snapshot** (*int*) – The snapshot that we're calculating properties for.

Notes

If *sage_output_format* is `sage_binary`, `gals` is a `numpy` structured array. If *sage_output_format*: is `sage_hdf5`, `gals` is an open HDF5 group. We refer to ../user/data_class for more information about adding your own Data Class to ingest data.

calc_properties_all_files (*calculation_functions*, *snapshot: int*, *close_file: bool = True*, *use_pbar: bool = True*, *debug: bool = False*)

Calculates galaxy properties for all files of a single *Model*.

Parameters

- **calculation_functions** (*dict [string, list(function, dict[string, variable])]*) – Specifies the functions used to calculate the properties of this *Model*. The key of this dictionary is the name of the plot toggle. The value is a list with the 0th element being the function and the 1st element being a dictionary of additional keyword arguments to be passed to the function. The inner dictionary is keyed by the keyword argument names with the value specifying the keyword argument value.

  All functions in this dictionary for called after the galaxies for each sub-file have been loaded. The function signature is required to be `func(Model, gals, <Extra Keyword Arguments>)`.
- **snapshot** (*int*) – The snapshot that we're calculating properties for.
- **close_file** (*boolean, optional*) – Some data formats have a single file data is read from rather than opening and closing the sub-files in `read_gals()`. Hence once the properties are calculated, the file must be closed. This variable flags whether the data class specific `close_file()` method should be called upon completion of this method.
- **use_pbar** (*Boolean, optional*) – If set, uses the `tqdm` package to create a progress bar.
- **debug** (*Boolean, optional*) – If set, prints out extra useful debug information.

init_binned_properties (*bin_low: float, bin_high: float, bin_width: float, bin_name: str, property_names: List[str], snapshot: int*)

Initializes the *properties* (and respective *bins*) that will binned on some variable. For example, the stellar mass function (SMF) will describe the number of galaxies within a stellar mass bin.

bins can be accessed via `Model.bins["bin_name"]` and are initialized as `ndarray`. *properties* can be accessed via `Model.properties["property_name"]` and are initialized using `numpy.zeros`.

Parameters

- **bin_low, bin_high, bin_width** (*floats*) – Values that define the minimum, maximum and width of the bins respectively. This defines the binning axis that the `property_names` properties will be binned on.
- **bin_name** (*string*) – Name of the binning axis, accessed by `Model.bins["bin_name"]`.
- **property_names** (*list of strings*) – Name of the properties that will be binned along the defined binning axis. Properties can be accessed using `Model.properties["property_name"]`; e.g., `Model.properties["SMF"]` would return the stellar mass function that is binned using the `bin_name` bins.
- **snapshot** (*int*) – The snapshot we're initialising the properties for.

`init_scatter_properties`(*property_names: List[str], snapshot: int*)

Initializes the *`properties`* that will be extended as `ndarray`. These are used to plot (e.g.,) a the star formation rate versus stellar mass for a subset of *`sample_size`* galaxies. Initializes as empty `ndarray`.

Parameters

- **property_names** (*list of strings*) – Name of the properties that will be extended as `ndarray`.
- **snapshot** (*int*) – The snapshot we're initialising the properties for.

`init_single_properties`(*property_names: List[str], snapshot: int*) → None

Initializes the *`properties`* that are described using a single number. This is used to plot (e.g.,) a the sum of stellar mass across all galaxies. Initializes as `0.0`.

Parameters

- **property_names** (*list of strings*) – Name of the properties that will be described using a single number.
- **snapshot** (*int*) – The snapshot we're initialising the properties for.

`select_random_galaxy_indices`(*inds: numpy.ndarray, num_inds_selected_already: int*) → numpy.ndarray

Selects random indices (representing galaxies) from `inds`. This method assumes that the total number of galaxies selected across all **SAGE** files analyzed is *`sample_size`* and that (preferably) these galaxies should be selected **equally** amongst all files analyzed.

For example, if we are analyzing 8 **SAGE** output files and wish to select 10,000 galaxies, this function would hence select 1,250 indices from `inds`.

If the length of `inds` is less than the number of requested values (e.g., `inds` only contains 1,000 values), then the next file analyzed will attempt to select 1,500 random galaxies (1,250 base plus an addition 250 as the previous file could not find enough galaxies).

At the end of the analysis, if there have not been enough galaxies selected, then a message is sent to the user.

`IMF`

The initial mass function.

Type `{"Chabrier", "Salpeter"}`

`base_sage_data_path`

Base path to the output data. This is the path without specifying any extra information about redshift or the file extension itself.

Type string

bins

The bins used to bin some *properties*. Bins are initialized through *init_binned_properties()*. Key is the name of the bin, (bin_name in *init_binned_properties()*).

Type dict [string, ndarray]

box_size

Size of the simulation box. Units are Mpc/h.

Type float

calculation_functions

A dictionary of functions that are used to compute the properties of galaxies. Here, the string is the name of the toggle (e.g., "SMF"), the value is a tuple containing the function itself (e.g., calc_SMF()), and another dictionary which specifies any optional keyword arguments to that function with keys as the name of variable (e.g., "calc_sub_populations") and values as the variable value (e.g., True).

Type dict[str, tuple[func, dict[str, any]]]

first_file_to_analyze

The first *SAGE* sub-file to be read. If *sage_output_format* is sage_binary, files read must be labelled *sage_data_path*.XXX. If *sage_output_format* is sage_hdf5, the file read will be *sage_data_path* and the groups accessed will be Core_XXX. In both cases, XXX represents the numbers in the range [*first_file_to_analyze*, *last_file_to_analyze*] inclusive.

Type int

hubble_h

Value of the fractional Hubble parameter. That is, H = 100*hubble_h.

Type float

label

Label that will go on axis legends for this *Model*.

Type string

last_file_to_analyze

The last **SAGE** sub-file to be read. If *sage_output_format* is sage_binary, files read must be labelled *sage_data_path*.XXX. If *sage_output_format* is sage_hdf5, the file read will be *sage_data_path* and the groups accessed will be Core_XXX. In both cases, XXX represents the numbers in the range [*first_file_to_analyze*, *last_file_to_analyze*] inclusive.

Type int

num_gals_all_files

Number of galaxies across all files. For HDF5 data formats, this represents the number of galaxies across all *Core_XXX* sub-groups.

Type int

num_sage_output_files

The number of files that **SAGE** wrote. This will be equal to the number of processors the **SAGE** ran with.

Notes

If *sage_output_format* is sage_hdf5, this attribute is not required.

Type int

output_path

Path to where some plots will be saved. Used for `plot_spatial_3d()`.

Type string

parameter_dirpath

The directory path to where the **SAGE** paramter file is located. This is only the base directory path and does not include the name of the file itself.

Type str

plot_toggles

Specifies which plots should be created for this model. This will control which properties should be calculated; e.g., if no stellar mass function is to be plotted, the stellar mass function will not be computed.

Type dict[str, bool]

plots_that_need_smf

Specifies the plot toggles that require the stellar mass function to be properly computed and analyzed. For example, plotting the quiescent fraction of galaxies requires knowledge of the total number of galaxies. The strings here must **EXACTLY** match the keys in *`plot_toggles`*.

Type list of ints

properties

The galaxy properties stored across the input files and snapshots. These properties are updated within the respective `calc_<plot_toggle>` functions.

The outside key is `"snapshot_XX"` where `XX` is the snapshot number for the property. The inner key is the name of the proeprty (e.g., `"SMF"`).

Type dict [string, dict [string, `ndarray`]] or dict[string, dict[string, float]

random_seed

Specifies the seed used for the random number generator, used to select galaxies for plotting purposes. If `None`, then uses default call to `seed()`.

Type Optional[int]

redshifts

Redshifts for this simulation.

Type `ndarray`

sSFRcut

The specific star formation rate above which a galaxy is flagged as “star forming”. Units are log10.

Type float

sage_data_path

Path to the output data. If *`sage_output_format`* is `sage_binary`, files read must be labelled *`sage_data_path`*.XXX. If *`sage_output_format`* is `sage_hdf5`, the file read will be *`sage_data_path`* and the groups accessed will be Core_XXX at snapshot *`snapshot`*. In both cases, `XXX` represents the numbers in the range [*`first_file_to_analyze`*, *`last_file_to_analyze`*] inclusive.

Type string

sage_file

The path to where the **SAGE** `.ini` file is located.

Type str

sage_output_format
The output format **SAGE** wrote in. A specific Data Class (e.g., *SageBinaryData* and *SageHdf5Data*) must be written and used for each *sage_output_format* option. We refer to ../user/data_class for more information about adding your own Data Class to ingest data.

Type {"sage_binary", "sage_binary"}

sample_size
Specifies the length of the *properties* attributes stored as 1-dimensional ndarray. These *properties* are initialized using *init_scatter_properties()*.

Type int

snapshot
Specifies the snapshot to be read. If *sage_output_format* is sage_hdf5, this specifies the HDF5 group to be read. Otherwise, if *sage_output_format* is sage_binary, this attribute will be used to index *redshifts* and generate the suffix for *sage_data_path*.

Type int

volume
Volume spanned by the trees analyzed by this model. This depends upon the number of files processed, [:py:attr:`~first_file_to_analyze`, :py:attr:`~last_file_to_analyze`], relative to the total number of files the simulation spans over, num_sim_tree_files.

Notes

This is **not** necessarily *box_size* cubed. It is possible that this model is only analysing a subset of files and hence the volume will be less.

Type volume

2.9 sage_analysis.sage_hdf5

This module defines the SageHdf5Data class. This class interfaces with the *Model* class to read in binary data written by **SAGE**. The value of *sage_output_format* is generally sage_hdf5 if it is to be read with this class.

If you wish to ingest data from your own flavour of SAGE, please open a Github issue, I plan to add this documentation in future :)

Author: Jacob Seiler.

class sage_analysis.sage_hdf5.**SageHdf5Data**(*model: sage_analysis.model.Model*, *sage_file_to_read: str*)
Class intended to inteface with the *Model* class to ingest the data written by **SAGE**. It includes methods for reading the output galaxies, setting cosmology etc. It is specifically written for when *sage_output_format* is sage_hdf5.

__init__(*model: sage_analysis.model.Model*, *sage_file_to_read: str*) → None
Instantiates the Data Class for reading in **SAGE** HDF5 data. In particular, opens up the file and ensures the data version matches the expected value.

Parameters **model** (*Model* instance) – The model that this data class is associated with; this class will read the data for this model.

close_file(*model*)
Closes the open HDF5 file.

determine_num_gals(*model: sage_analysis.model.Model*, *snapshot: int*, **args*)
Determines the number of galaxies in all cores for this model at the specified snapshot.

Parameters

- **model** (`Model` class) – The `Model` we're reading data for.
- **snapshot** (*int*) – The snapshot we're analysing.
- ***args** (*Any*) – Extra arguments to allow other data class to pass extra arguments to their version of `determine_num_gals`.

determine_volume_analyzed(*model: sage_analysis.model.Model*) → float
Determines the volume analyzed. This can be smaller than the total simulation box.

Parameters **model** (`Model` instance) – The model that this data class is associated with.

Returns **volume** – The numeric volume being processed during this run of the code in (Mpc/h)^3.

Return type float

read_gals(*model: sage_analysis.model.Model*, *core_num: int*, *snapshot: int*, *pbar: Optional[tqdm.std.tqdm] = None*, *plot_galaxies: bool = False*, *debug: bool = False*) → Any
Reads the galaxies of a single core at the specified `snapshot`.

Parameters

- **model** (`Model` class) – The `Model` we're reading data for.
- **core_num** (*Integer*) – The core group we're reading.
- **pbar** (`tqdm` class instance, optional) – Bar showing the progress of galaxy reading. If `None`, progress bar will not show.
- **plot_galaxies** (*Boolean, optional*) – If set, plots and saves the 3D distribution of galaxies for this file.
- **debug** (*Boolean, optional*) – If set, prints out extra useful debug information.

Returns **gals** – The galaxies for this file.

Return type `h5py` group

Notes

`tqdm` does not play nicely with printing to stdout. Hence we disable the `tqdm` progress bar if `debug=True`.

read_sage_params(*sage_file_path: str*) → Dict[str, Any]
Read the **SAGE** parameter file.

Parameters **sage_file_path** (*string*) – Path to the **SAGE** parameter file.

Returns **model_dict** – Dictionary containing the parameter names and their values.

Return type dict [str, var]

update_snapshot_and_data_path(*model: sage_analysis.model.Model*, *snapshot: int*)
Updates the `snapshot` attribute to `snapshot`. As the HDF5 file contains all snapshot information, we do **not** need to update the path to the output data. However, ensure that the file itself is still open.

2.10 sage_analysis.sage_binary

This module defines the `SageBinaryData` class. This class interfaces with the *`Model`* class to read in binary data written by **SAGE**. The value of *`sage_output_format`* is generally `sage_binary` if it is to be read with this class.

If you wish to ingest data from your own flavour of SAGE, please open a Github issue, I plan to add this documentation in future :)

Author: Jacob Seiler.

class `sage_analysis.sage_binary.`**`SageBinaryData`**(*model: sage_analysis.model.Model*, *sage_file_to_read: str*)

Class intended to inteface with the *`Model`* class to ingest the data written by **SAGE**. It includes methods for reading the output galaxies, setting cosmology etc. It is specifically written for when *`sage_output_format`* is `sage_binary`.

`__init__`(*model: sage_analysis.model.Model*, *sage_file_to_read: str*) → None

Instantiates the Data Class for reading in **SAGE** binary data. In particular, generates the `numpy` structured array to read the output galaxies.

model: *`Model`* instance The model that this data class is associated with; this class will read the data for this model.

`close_file`(*model: sage_analysis.model.Model*)

An empty method to ensure consistency with the HDF5 data class. This is empty because snapshots are saved over different files by default in the binary format.

`determine_num_gals`(*model: sage_analysis.model.Model*, **args*)

Determines the number of galaxies in all files for this *`Model`*.

Parameters

- **model** (*`Model`* class) – The *`Model`* we're reading data for.
- ***args** (*Any*) – Extra arguments to allow other data class to pass extra arguments to their version of `determine_num_gals`.

`determine_volume_analyzed`(*model: sage_analysis.model.Model*) → float

Determines the volume analyzed. This can be smaller than the total simulation box.

Parameters **model** (*`Model`* instance) – The model that this data class is associated with.

Returns **volume** – The numeric volume being processed during this run of the code in (Mpc/h)^3.

Return type float

`read_gals`(*model: sage_analysis.model.Model*, *file_num: int*, *snapshot: int*, *pbar: Optional[tqdm.std.tqdm] = None*, *plot_galaxies: bool = False*, *debug: bool = False*)

Reads the galaxies of a model file at snapshot specified by *`snapshot`*.

Parameters

- **model** (*`Model`* class) – The *`Model`* we're reading data for.
- **file_num** (*int*) – Suffix number of the file we're reading.
- **pbar** (`tqdm` class instance, optional) – Bar showing the progress of galaxy reading. If `None`, progress bar will not show.
- **plot_galaxies** (*bool, optional*) – If set, plots and saves the 3D distribution of galaxies for this file.

- **debug** (*bool, optional*) – If set, prints out extra useful debug information.

Returns **gals** – The galaxies for this file.

Return type `numpy` structured array with format given by **:py:method:'~_get_galaxy_struct'**

Notes

`tqdm` does not play nicely with printing to stdout. Hence we disable the `tqdm` progress bar if `debug=True`.

`read_sage_params` (*sage_file_path: str*) → Dict[str, Any]

Read the **SAGE** parameter file.

Parameters **sage_file_path** (*string*) – Path to the **SAGE** parameter file.

Returns **model_dict** – Dictionary containing the parameter names and their values.

Return type dict [str, var]

`update_snapshot_and_data_path` (*model: sage_analysis.model.Model, snapshot: int, use_absolute_path: bool = False*)

Updates the `_sage_data_path` to point to a new redshift file. Uses the redshift array *`redshifts`*.

Parameters

- **snapshot** (*int*) – Snapshot we're updating `_sage_data_path` to point to.
- **use_absolute_path** (*bool*) – If specified, will use the absolute path to the **SAGE** output data. Otherwise, will use the path that is relative to the **SAGE** parameter file. This is hand because the **SAGE** parameter file can contain either relative or absolute paths.

2.11 sage_analysis.example_calcs

Here we show a myriad of functions that can be used to calculate properties from the **SAGE** output. By setting the correct plot toggles and calling *`generate_func_dict()`*, a dictionary containing these functions can be generated and passed to *`calc_properties_all_files()`* to calculate the properties.

The properties are stored (and updated) in the *`properties`* attribute.

We refer to ../user/analysing_sage for more information on how the calculations are handled.

Author: Jacob Seiler

`sage_analysis.example_calcs.`**`calc_BMF`** (*model, gals, snapshot: int*)

Calculates the baryon mass function of the given galaxies. That is, the number of galaxies at a given baryon (stellar + cold gas) mass.

The `Model.properties["snapshot_<snapshot>"]["BMF"]` array will be updated.

`sage_analysis.example_calcs.`**`calc_BTF`** (*model, gals, snapshot: int*)

Calculates the baryonic Tully-Fisher relation for spiral galaxies in the given set of galaxies.

The number of galaxies added to `Model.properties["snapshot_<snapshot>"]["BTF_mass"]` and `Model.properties["snapshot_<snapshot>"]["BTF_vel"]` arrays is given by *`sample_size`* weighted by `number_spirals_passed / _num_gals_all_files`. If this value is greater than `number_spirals_passed`, then all spiral galaxies will be used.

`sage_analysis.example_calcs.`**`calc_GMF`**(*model*, *gals*, *snapshot: int*)

Calculates the gas mass function of the given galaxies. That is, the number of galaxies at a given cold gas mass.

The `Model.properties["snapshot_<snapshot>"]["GMF"]` array will be updated.

`sage_analysis.example_calcs.`**`calc_SFRD_history`**(*model*, *gals*, *snapshot: int*)

Calculates the sum of the star formation across all galaxies. This will be normalized by the simulation volume to determine the density. See `plot_SFRD()` for full implementation.

The `Model.properties["snapshot_<snapshot>"]["SFRD"]` value is updated.

`sage_analysis.example_calcs.`**`calc_SMD_history`**(*model*, *gals*, *snapshot: int*)

Calculates the sum of the stellar mass across all galaxies. This will be normalized by the simulation volume to determine the density. See `plot_SMD()` for full implementation.

The `Model.properties["snapshot_<snapshot>"]["SMD"]` value is updated.

`sage_analysis.example_calcs.`**`calc_SMF`**(*model: sage_analysis.model.Model*, *gals*, *snapshot: int*, *calc_sub_populations: bool = False*, *smf_property_name: str = 'SMF'*)

Calculates the stellar mass function of the given galaxies. That is, the number of galaxies at a given stellar mass.

The `Model.properties["snapshot_<snapshot>"]"SMF"]` array will be updated. We also split the galaxy population into "red" and "blue" based on the value of *`sSFRcut`* and update the `Model.properties["snapshot_<snapshot>"]["red_SMF"]` and `Model.properties["snapshot_<snapshot>"]["blue_SMF"]` arrays.

Parameters

- **snapshot** (*int*) – The snapshot the SMF is being calculated at.
- **plot_sub_populations** (*boolean, optional*) – If `True`, calculates the stellar mass function for red and blue sub-populations.
- **smf_property_name** (*string, optional*) – The name of the property used to store the stellar mass function. Useful if different calculations are computing the stellar mass function but saving it as a different property.

`sage_analysis.example_calcs.`**`calc_SMF_history`**(*model*, *gals*, *snapshot: int*)

Calculates the stellar mass function of the given galaxies. That is, the number of galaxies at a given stellar mass.

The `Model.properties["SMF"_history]` array will be updated.

`sage_analysis.example_calcs.`**`calc_baryon_fraction`**(*model*, *gals*, *snapshot: int*)

Calculates the `mass_baryons / halo_virial_mass` as a function of halo virial mass for each baryon reseroivr (stellar, cold, hot, ejected, intra-cluster stars and black hole). Also calculates the ratio for the total baryonic mass.

The `Model.properties["snapshot_<snapshot>"]["halo_<reservoir_name>_fraction_sum"]` arrays are updated for each reservoir. In addition, `Model.properties["snapshot_<snapshot>"]["halo_baryon_f` is updated.

Notes

The halo virial mass we use is the **background FoF halo**, not the immediate host halo of each galaxy.

We only **sum** the baryon mass in each stellar mass bin. When converting this to the mass fraction, one must divide by the number of halos in each halo mass bin, `Model.properties["snapshot_<snapshot>"]["fof_HMF"]`. See *`plot_baryon_fraction()`* for full implementation.

If the `Model.properties["snapshot_<snapshot>"]["fof_HMF"]` property, with associated bins `Model.bins["halo_mass"bin"]` have not been initialized, a `ValueError` is thrown.

`sage_analysis.example_calcs.`**`calc_bh_bulge`**(*model*, *gals*, *snapshot: int*)

Calculates the black hole mass as a function of bulge mass.

The number of galaxies added to `Model.properties["snapshot_<snapshot>"]["BlackHoleMass"]` and `Model.propertiesp["snapshot_<snapshot>"]["BulgeMass"]` arrays is given by *`sample_size`* weighted by `number_galaxies_passed / _num_gals_all_files`. If this value is greater than `number_galaxies_passed`, then all galaxies will be used.

Notes

We only consider galaxies with bulge mass greater than 10^8 Msun/h and a black hole mass greater than 10^5 Msun/h.

`sage_analysis.example_calcs.`**`calc_bulge_fraction`**(*model*, *gals*, *snapshot: int*)

Calculates the `bulge_mass / stellar_mass` and `disk_mass / stellar_mass` ratios as a function of stellar mass.

The `Model.properties["snapshot_<snapshot>"]["fraction_bulge_sum"]`, `Model.properties["snapshot_<snapshot>"]["fraction_disk_sum"]`, `Model.properties["snapshot_<snapshot>"]["fraction_bulge_var"]`, `Model.properties["snapshot_<snapshot>"]["fraction_disk_var"]` arrays will be updated.

Notes

We only **sum** the bulge/disk mass in each stellar mass bin. When converting this to the mass fraction, one must divide by the number of galaxies in each stellar mass bin, the stellar mass function `Model.properties["snapshot_<snapshot>"]["SMF"]`. See *`plot_bulge_fraction()`* for full implementation.

`sage_analysis.example_calcs.`**`calc_gas_fraction`**(*model*, *gals*, *snapshot: int*)

Calculates the fraction of baryons that are in the cold gas reservoir as a function of stellar mass.

The number of galaxies added to `Model.properties["snapshot_<snapshot>"]["gas_frac_mass"]` and `Model.properties["snapshot_<snapshot>"]["gas_frac"]` arrays is given by *`sample_size`* weighted by `number_spirals_passed / _num_gals_all_files`. If this value is greater than `number_spirals_passed`, then all spiral galaxies will be used.

`sage_analysis.example_calcs.`**`calc_metallicity`**(*model*, *gals*, *snapshot: int*)

Calculates the metallicity as a function of stellar mass.

The number of galaxies added to `Model.properties["snapshot_<snapshot>"]["metallicity_mass"]` and `Model.properties["snapshot_<snapshot>"]["metallicity"]` arrays is given by *`sample_size`* weighted by `number_centrals_passed / _num_gals_all_files`. If this value is greater than `number_centrals_passed`, then all central galaxies will be used.

`sage_analysis.example_calcs.`**`calc_quiescent`**(*model*, *gals*, *snapshot: int*)

Calculates the quiescent galaxy fraction as a function of stellar mass. The galaxy population is also split into central and satellites and the quiescent fraction of these are calculated.

The `Model.properties["snapshot_<snapshot>"]["centrals_MF"]`, `Model.properties["snapshot_<snapshot>"]["satellites_MF"]`, `Model.properties["snapshot_<snapshot>"]["quiescent_galaxy_counts"]`, `Model.properties["snapshot_<snapshot>"]["quiescent_centrals_counts"]`, and `Model.properties["snapshot_<snapshot>"]["quiescent_satellites_counts"]` arrays will be updated.

Notes

We only **count** the number of quiescent galaxies in each stellar mass bin. When converting this to the quiescent fraction, one must divide by the number of galaxies in each stellar mass bin, the stellar mass function `Model.properties["snapshot_<snapshot>"]["SMF"]`. See *plot_quiescent()* for an example implementation.

`sage_analysis.example_calcs.`**`calc_reservoirs`**(*model*, *gals*, *snapshot: int*)

Calculates the mass in each reservoir as a function of halo virial mass.

The number of galaxies added to `Model.properties["snapshot_<snapshot>"]["reservoir_mvir"]` and `Model.properties["snapshot_<snapshot>"]["reservoir_<reservoir_name>"]` arrays is given by *sample_size* weighted by `number_centrals_passed / _num_gals_all_files`. If this value is greater than `number_centrals_passed`, then all central galaxies will be used.

`sage_analysis.example_calcs.`**`calc_sSFR`**(*model*, *gals*, *snapshot: int*)

Calculates the specific star formation rate (star formation divided by the stellar mass of the galaxy) as a function of stellar mass.

The number of galaxies added to `Model.properties["snapshot_<snapshot>"]["sSFR_mass"]` and `Model.properties["snapshot_<snapshot>"]["sSFR_sSFR"]` arrays is given by *sample_size* weighted by `number_gals_passed / _num_gals_all_files`. If this value is greater than `number_gals_passed`, then all galaxies with non-zero stellar mass will be used.

`sage_analysis.example_calcs.`**`calc_spatial`**(*model*, *gals*, *snapshot: int*)

Calculates the spatial position of the galaxies.

The number of galaxies added to `Model.properties["snapshot_<snapshot>"]["<x/y/z>_pos"]` arrays is given by *sample_size* weighted by `number_galaxies_passed / _num_gals_all_files`. If this value is greater than `number_galaxies_passed`, then all galaxies will be used.

2.12 sage_analysis.example_plots

Here we show a myriad of functions that can be used to plot properties calculated from the **SAGE** output.

We refer to ../user/plot for more information on how plotting is handled.

Authors: (Jacob Seiler, Manodeep Sinha)

`sage_analysis.example_plots.`**`plot_BMF`**(*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper*) → matplotlib.figure.Figure

Plots the baryonic mass function for the specified models. This is the mass function for the stellar mass + cold gas.

Parameters

- **models** (List of *Model* class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.
- **snapshots** (*nested list of ints*) – The snapshots to be plotted for each *Model* in `models`.

  The length of the outer list **MUST** be equal to the length of `models`. For each model, the baryonic mass function of all snapshots are plotted on the figure.

- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.
- **Generates**
- **———**
- **The plot will be saved as "<output_path>2.BaryonicMassFunction.<output_format>"**

`sage_analysis.example_plots.`**`plot_BTF`**`(`*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper*`)` → matplotlib.figure.Figure

Plots the baryonic Tully-Fisher relationship for the specified models.

Parameters

- **models** (List of *`Model`* class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.
- **snapshots** (*nested list of ints*) – The snapshots to be plotted for each *`Model`* in `models`.

  The length of the outer list **MUST** be equal to the length of `models`. For each model, the baryonic Tully-Fisher relationship of all snapshots are plotted on the figure.
- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.
- **Generates**
- **———**
- **The plot will be saved as "<output_path>4.BaryonicTullyFisher.<output_format>"**

`sage_analysis.example_plots.`**`plot_GMF`**`(`*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper*`)` → matplotlib.figure.Figure

Plots the gas mass function for the specified models. This is the mass function for the cold gas.

Parameters

- **models** (List of *`Model`* class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.
- **snapshots** (*nested list of ints*) – The snapshots to be plotted for each *`Model`* in `models`.

  The length of the outer list **MUST** be equal to the length of `models`. For each model, the gas mass function of all snapshots are plotted on the figure.
- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.
- **Generates**
- **———**
- **The plot will be saved as "<output_path>3.GasMassFunction.<output_format>"**

`sage_analysis.example_plots.`**`plot_SFRD_history`**(*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper*) → matplotlib.figure.Figure

Plots the evolution of star formation rate density for the specified models.

Parameters

- **models** (List of *`Model`* class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.
- **snapshots** (*nested list of ints*) – This is a dummy variable that is present to ensure the signature is identical to the other plot functions.
- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.
- **Generates**
- **———**
- **The plot will be saved as "<output_path>B.SFRDensity.<output_format>"**

`sage_analysis.example_plots.`**`plot_SMD_history`**(*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper*) → matplotlib.figure.Figure

Plots the evolution of stellar mass density for the specified models.

Parameters

- **models** (List of *`Model`* class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.
- **snapshots** (*nested list of ints*) – This is a dummy variable that is present to ensure the signature is identical to the other plot functions.
- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.
- **Generates**
- **———**
- **The plot will be saved as "<output_path>C.StellarMassDensity.<output_format>"**

`sage_analysis.example_plots.`**`plot_SMF`**(*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper, plot_sub_populations: bool = False*) → matplotlib.figure.Figure

Plots the stellar mass function for the specified models.

Parameters

- **models** (List of *`Model`* class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.

- **snapshots** (*nested list of ints*) – The snapshots to be plotted for each `Model` in `models`.

  The length of the outer list **MUST** be equal to the length of `models`. For each model, the stellar mass function of all snapshots are plotted on the figure.
- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.
- **plot_sub_populations** (*Boolean, default False*) – If `True`, plots the stellar mass function for red and blue sub-populations.
- **Generates**
- **———**
- **The plot will be saved as "<output_path>1.StellarMassFunction.<output_format>"**

`sage_analysis.example_plots.`**`plot_SMF_history`**(*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper*) → matplotlib.figure.Figure

Plots the evolution of the stellar mass function for the specified models. This function loops over the value of `model.SMF_snaps` and plots and the SMFs at each snapshots.

Parameters

- **models** (List of `Model` class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.
- **snapshots** (*nested list of ints*) – This is a dummy variable that is present to ensure the signature is identical to the other plot functions.
- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.
- **Generates**
- **———**
- **The plot will be saved as "<output_path>A.StellarMassFunction.<output_format>"**

`sage_analysis.example_plots.`**`plot_baryon_fraction`**(*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper, plot_sub_populations: bool = False*) → matplotlib.figure.Figure

Plots the total baryon fraction as afunction of halo mass for the specified models.

Parameters

- **models** (List of `Model` class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.
- **snapshots** (*nested list of ints*) – The snapshots to be plotted for each `Model` in `models`.

  The length of the outer list **MUST** be equal to the length of `models`. For each model, the baryon fraction of all snapshots are plotted on the figure.

- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.
- **plot_sub_populations** (*Boolean, default False*) – If `True`, plots the baryon fraction for each reservoir. Otherwise, only plots the total baryon fraction.
- **Generates**
- **———**
- **The plot will be saved as "<output_path>11.BaryonFraction.<output_format>"**

`sage_analysis.example_plots.`**`plot_bh_bulge`**(*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper*) → matplotlib.figure.Figure

Plots the black-hole bulge relationship for the specified models.

Parameters

- **models** (List of *`Model`* class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.
- **snapshots** (*nested list of ints*) – The snapshots to be plotted for each *`Model`* in `models`.

  The length of the outer list **MUST** be equal to the length of `models`. For each model, the black hole bulge relationship of all snapshots are plotted on the figure.
- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.
- **Generates**
- **———**
- **The plot will be saved as "<output_path>8.BlackHoleBulgeRelationship.<output_format>"**

`sage_analysis.example_plots.`**`plot_bulge_fraction`**(*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper, plot_disk_fraction: bool = False, plot_var: bool = False*) → matplotlib.figure.Figure

Plots the fraction of the stellar mass that is located in the bulge/disk as a function of stellar mass for the specified models.

Parameters

- **models** (List of `Model` class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.
- **snapshots** (*nested list of ints*) – The snapshots to be plotted for each *`Model`* in `models`.

  The length of the outer list **MUST** be equal to the length of `models`. For each model, the bulge fraction of all snapshots are plotted on the figure.
- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.

- **plot_disk_fraction** (*bool, optional*) – If specified, will also plot the disk fraction.
- **plot_var** (*Boolean, default False*) – If `True`, plots the variance as shaded regions.
- **Generates**
- **———**
- **The plot will be saved as :py:attr:'~sage_analysis.plot_helper.PlotHelper.output_path'10.BulgeMassFraction.**
- **:py:attr:'~sage_analysis.plot_helper.PlotHelper.output_format'.**

`sage_analysis.example_plots.`**`plot_gas_fraction`**(*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper*) → matplotlib.figure.Figure

Plots the fraction of baryons that are in the cold gas reservoir as a function of stellar mass for the specified models.

Parameters

- **models** (List of *`Model`* class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.
- **snapshots** (*nested list of ints*) – The snapshots to be plotted for each *`Model`* in `models`.

  The length of the outer list **MUST** be equal to the length of `models`. For each model, the gas fraction of all snapshots are plotted on the figure.
- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.
- **Generates**
- **———**
- **The plot will be saved as "<output_path>6.GasFraction.<output_format>"**

`sage_analysis.example_plots.`**`plot_metallicity`**(*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper*) → matplotlib.figure.Figure

Plots the metallicity as a function of stellar mass for the speicifed models.

Parameters

- **models** (List of *`Model`* class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.
- **snapshots** (*nested list of ints*) – The snapshots to be plotted for each *`Model`* in `models`.

  The length of the outer list **MUST** be equal to the length of `models`. For each model, the metallicity of all snapshots are plotted on the figure.
- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.
- **Generates**
- **———**
- **The plot will be saved as "<output_path>7.Metallicity.<output_format>"**

`sage_analysis.example_plots.`**`plot_quiescent`**(*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper, plot_sub_populations: bool = False*) → matplotlib.figure.Figure

Plots the fraction of galaxies that are quiescent as a function of stellar mass for the specified models. The quiescent cut is defined by *`sSFRcut`*.

Parameters

- **models** (List of *`Model`* class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.
- **snapshots** (*nested list of ints*) – The snapshots to be plotted for each *`Model`* in `models`.

  The length of the outer list **MUST** be equal to the length of `models`. For each model, the quiescent fraction of all snapshots are plotted on the figure.
- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.
- **plot_sub_populations** (*Boolean, default False*) – If `True`, plots the centrals and satellite sub-populations.
- **Generates**
- **———**
- **The plot will be saved as "<output_path>9.QuiescentFraction.<output_format>"**

`sage_analysis.example_plots.`**`plot_reservoirs`**(*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper*) → List[matplotlib.figure.Figure]

Plots the mass in each reservoir as a function of halo mass for the specified models.

Parameters

- **models** (List of *`Model`* class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.
- **snapshots** (*nested list of ints*) – The snapshots to be plotted for each *`Model`* in `models`.

  The length of the outer list **MUST** be equal to the length of `models`. For each model, each snapshot will be plotted and saved as a separate figure.
- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.
- **Generates**
- **———**
- **A plot will be saved as '"<output_path>12.MassReservoirs<model.label>.<output_format>"' for each mode.**

`sage_analysis.example_plots.`**`plot_sSFR`**(*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper*) → matplotlib.figure.Figure

Plots the specific star formation rate as a function of stellar mass for the specified models.

Parameters

- **models** (List of `Model` class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.
- **snapshots** (*nested list of ints*) – The snapshots to be plotted for each `Model` in `models`.

  The length of the outer list **MUST** be equal to the length of `models`. For each model, the specific star formation rate of all snapshots are plotted on the figure.
- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.
- **Generates**
- **———**
- **The plot will be saved as "<output_path>5.SpecificStarFormationRate.<output_format>"**

`sage_analysis.example_plots.`**`plot_spatial`**(*models: List[sage_analysis.model.Model], snapshots: List[List[int]], plot_helper: sage_analysis.plot_helper.PlotHelper*) → matplotlib.figure.Figure

Plots the spatial distribution of the galaxies for specified models.

Parameters

- **models** (List of `Model` class instance) – Models that will be plotted. These instances contain the properties necessary to create the plot, accessed via `Model.properties["snapshot_<snapshot>"]["property_name"]`.
- **snapshots** (*nested list of ints*) – The snapshots to be plotted for each `Model` in `models`.

  The length of the outer list **MUST** be equal to the length of `models`. For each model, the spatial position of all snapshots are plotted on the figure.
- **plot_helper** (`PlotHelper`) – A helper class that contains attributes and methods to assist with plotting. In particular, the path where the plots will be saved and the output format. Refer to ../user/plot_helper for more information on how to initialize this class and its use.
- **Generates**
- **———**
- **A plot will be saved as '"<output_path>13.SpatialDistribution<model.label>.<output_format>"' for each**
- **model.**

`sage_analysis.example_plots.`**`plot_spatial_3d`**(*pos, output_file, box_size*) → matplotlib.figure.Figure

Plots the 3D spatial distribution of galaxies.

Parameters

- **pos** (`numpy` 3D array with length equal to the number of galaxies) – The position (in Mpc/h) of the galaxies.
- **output_file** (*String*) – Name of the file the plot will be saved as.

Returns

Return type None. A plot will be saved as `output_file`.

2.13 sage_analysis.utils

`sage_analysis.utils.`**`find_closest_indices`**(*values: List[float], target_values: List[float]*) → List[int]

Finds the indices in `values` that result in values closest to `target_values`.

`sage_analysis.utils.`**`generate_func_dict`**(*plot_toggles*, *module_name*, *function_prefix*, *keyword_args={}*) → Dict[str, Tuple[Callable, Dict[str, Any]]]

Generates a dictionary where the keys are the function name and the value is a list containing the function itself (0th element) and keyword arguments as a dictionary (1st element). All functions in the returned dictionary are expected to have the same call signature for non-keyword arguments. Functions are only added when the `plot_toggles` value is non-zero.

Functions are required to be named `<module_name><function_prefix><plot_toggle_key>` For example, the default calculation function are kept in the `model.py` module and are named `calc_<toggle>`. E.g., `sage_analysis.model.calc_SMF()`, `sage_analysis.model.calc_BTF()`, `sage_analysis.model.calc_sSFR()` etc.

Parameters

- **plot_toggles** (*dict, [string, int]*) – Dictionary specifying the name of each property/plot and whether the values will be generated + plotted. A value of 1 denotes plotting, whilst a value of 0 denotes not plotting. Entries with a value of 1 will be added to the function dictionary.
- **module_name** (*string*) – Name of the module where the functions are located. If the functions are located in this module, pass an empty string "".
- **function_prefix** (*string*) – Prefix that is added to the start of each function.
- **keyword_args** (*dict [string, dict[string, variable]], optional*) – Allows the adding of keyword aguments to the functions associated with the specified plot toggle. The name of each keyword argument and associated value is specified in the inner dictionary.

Returns **func_dict** – The key of this dictionary is the name of the function. The value is a list with the 0th element being the function and the 1st element being a dictionary of additional keyword arguments to be passed to the function. The inner dictionary is keyed by the keyword argument names with the value specifying the keyword argument value.

Return type dict [string, tuple(function, dict[string, variable])]

`sage_analysis.utils.`**`read_generic_sage_params`**(*sage_file_path: str*) → Dict[str, Any]

Reads the **SAGE** parameter file values. This function is used for the default `sage_binary` and `sage_hdf5` formats. If you have a custom format, you will need to write a `read_sage_params` function in your own data class.

Parameters **sage_file_path** (*string*) – Path to the **SAGE** parameter file.

Returns

- **model_dict** (*dict [str, var]*) – Dictionary containing the parameter names and their values.
- *Errors*
- ——
- *FileNotFoundError* – Raised if the specified **SAGE** parameter file is not found.

`sage_analysis.utils.`**`select_random_indices`**(*inds: numpy.ndarray*, *global_num_inds_available: int*, *global_num_inds_requested: int*, *seed: Optional[int] = None*) → numpy.ndarray

Select a random subset of indices if the total number of indices (across all files) is known. This function is used if selecting (e.g.,) 100 galaxies from a sample of 10,000.

However, if the total number of indices is **NOT** known, then this function is not valid. For example, if one wanted to select 100 spiral galaxies, we may not know how many spiral galaxies are present across all files. In such scenarios, `select_random_indices_assumed_equal_distribution()` should be used.

Parameters

- **vals** (`ndarray` of values) – Values that the random subset is selected from.
- **global_num_inds_available** (*int*) – The total number of indices available across all files.
- **global_num_inds_requested** (*int*) – The total number of indices requested across all files.
- **seed** (*int, optional*) – If specified, seeds the random number generator with the specified seed.

Returns **random_inds** – Values chosen.

Return type `ndarray` of values

Python Module Index

Index