SAGE Documentation

Release 1.0

Jacob Seiler, Manodeep Sinha, Darren Croton

Mar 08, 2021

User Documentation

This is the documentation for the Semi-Analytic Galaxy Evolution model. **SAGE** was original developed by *Darren Croton* and is a significant update to the model described in Croton et al., 2006. The updated **SAGE** model is described Croton et al., 2016. The code is publicly available and can be found on Github.

CHAPTER 1

Citation

If you use **SAGE** in a publication, please cite the following:

```
@ARTICLE{2016ApJS..222...22C,
    author = {{Croton}, D.~J. and {Stevens}, A.~R.~H. and {Tonini}, C. and
            {Garel}, T. and {Bernyk}, M. and {Bibiano}, A. and {Hodkinson}, L. and
            {Mutch}, S.~J. and {Poole}, G.~B. and {Shattow}, G.~M.},
    title = "{Semi-Analytic Galaxy Evolution (SAGE): Model Calibration and Basic
Results}",
    journal = {\apjs},
    archivePrefix = "arXiv",
    eprint = {1601.04709},
    keywords = {galaxies: active, galaxies: evolution, galaxies: halos, methods:
numerical},
    year = 2016,
    month = feb,
    volume = 222,
    eid = {22},
    pages = {22},
    doi = {10.3847/0067-0049/222/2/22},
    adsurl = {http://adsabs.harvard.edu/abs/2016ApJS..222...22C},
    adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```

CHAPTER 2

Maintainers

- Jacob Seiler (@jacobseiler)
- Manodeep Sinha (@manodeep)
- Darren Croton (@darrencroton)
- user-docs
- *API Reference*

2.1 Introduction

SAGE is a publicly available code-base for modelling galaxy formation in a cosmological context. A description of the model and its default calibration results can be found in Croton et al. (2016). SAGE is a significant update to that previously used in Croton et al. (2006).

SAGE is written in C and was built to be modular and customisable. It will run on any N-body simulation whose trees are organised in a supported format and contain a minimum set of basic halo properties. For testing purposes, treefiles for the mini-Millennium Simulation are available here.

Galaxy formation models built using SAGE on the Millennium, Bolshoi and simulations can be downloaded at the Theoretical Astrophysical Observatory (TAO). You can also find SAGE on ascl.net.

2.1.1 Why Use SAGE?

2.2 Getting Started

2.2.1 Pre-Requisites

SAGE should compile on most systems out of the box and the only required tool is a C99 compiler. GSL is recommended but not necessary.

2.2.2 Downloading

SAGE can be installed by cloning the GitHub repository:

```
$ git clone https://github.com/sage-home/sage-model
$ cd sage-model/
```

2.2.3 Building

To create the SAGE executable, simply run the following command:

```
$ make
```

SAGE is MPI compatible which can be enabled setting `USE-MPI = yes` in the `Makefile`. To run in parallel, ensure that you have a installed an MPI distribution (OpenMPI, MPICH, Intel MPI etc). When compiling with MPI support, the `Makefile` expects that the MPI compiler is called `mpicc` and is configured appropriately.

2.2.4 HDF5 Support

SAGE supports reading and writing in **'HDF5'_**. We recommend writing your output data in HDF5 for ease-of-use. To easily handle your HDF5 installation, we also recommend using Conda.

```
$ conda install -q --yes -c conda-forge hdf5
```

HDF5 support can be enabled by setting `USE-HDF5 = yes` in the `Makefile`. Adjust the `tree_type` and `OutputFormat` in your **SAGE** parameter file to allow reading/writing of HDF5 files.

2.2.5 Example Usage

SAGE runs on dark matter halo merger trees constructed in a *vertical* format. The trees for the Mini-Millennium dark matter simulation (a smaller box size version of the *Millennium simulation* with identical mass resolution) can be retrieved by executing the `first_run.sh` script from within the `sage-model` directory. This will create the necessary file structure and parameter file required for running **SAGE**.

```
$ ./first_run.sh
```

After this, the model can be run using:

```
$ ./sage input/millennium.par
```

or in parallel as:

```
$ mpirun -np <NUMBER_PROCESSORS> ./sage input/millennium.par
```

2.3 Plotting SAGE Results

2.3.1 Installation

We have created a small analysis package (**sage_analysis**) to provide greater flexibility in the plotting options for **SAGE**. Indeed, this package is highly customizable and we recommend using it for a variety of projects! It can be locally installed via:

```
$ cd sage-model/
$ pip install . (--user)
```

2.3.2 Example Usage

In the plotting

In the analysis directory are a number of Python scripts to read and parse the SAGE output. The most important file is example.py which creates plots for the default Mini-Millennium galaxies.

```
$ cd analysis/
$ python example.py
```

and will create a number of useful diagnostic plots in the analysis/plots directory.

We also include the ability to compare the properties of a number of different models. See the documenation in the __main__ function call of example.py to use this functionality.

2.4 Defining Custom Plot Toggles

2.5 Comprehensive API reference

2.5.1 sage_analysis package

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*)

Bases: object

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

_determine_history_snapshots(*model: sage_analysis.model.Model*) → Optional[List[int]]

Determines which snapshots need to be iterated over to track properties over time. For each Model, the _history_<property>_redshifts and _history_<property>_snapshots attributes are updated.

Parameters model (Model) – The Model instance to be updated.

Returns snapshots_to_loop – The snapshots that need to be analyzed for this model to ensure that the requested redshifts are analyzed for the history properties.

Return type list of ints

_determine_snapshots_to_use(*snapshots: Optional[List[List[int]]], redshifts: Optional[List[List[int]]]*) → List[List[int]]

Determine which snapshots should be analyzed/plotted based on the input from the user.

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`*.

  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`*.

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

Returns

- **snapshots_for_models** (*nested list of ints*) – The snapshots to be analyzed for each model.
- *Errors*
- ——
- *ValueError* – Thrown if **BOTH** `snapshots` and `redshifts` are specified.

_does_smf_need_computing(*model: sage_analysis.model.Model*) → bool

Determines whether the stellar mass function needs to be calculated based on the values of *`plot_toggles`* *`plots_that_need_smf`*.

Parameters **model** (*`Model`*) – The *`Model`* instance we're checking.

Returns A boolean indicating whether the stellar mass function needs to be computed or not.

Return type bool

_initialise_properties(*name: str, model: sage_analysis.model.Model, galaxy_properties: Dict[str, Union[str, List[str]]], snapshot: int*) → None

Initialises galaxy properties that will be analyzed.

Parameters

- **name** (*string*) – The name of the bins if the properties will be binned or a unique identifying name otherwise.
- **model** (*`Model`*) – The *`Model`* instance to be updated.

- **galaxy_properties** (*dict[str, float or str or list of strings]]*) – The galaxy properties that will be initialized. We defer to `galaxy_properties_to_analyze` in the **:py:method:'~__init__'** method for a full description of this variable.
- **snapshot** (*int*) – The snapshot the properties are being updated for.

_read_sage_file(*model: sage_analysis.model.Model*) → None

Reads a **SAGE** parameter file to determine all parameters such as cosmology, redshift list, etc. In particular, also initializes the `data_class` for each model. This attribute is unique depending upon the value of *sage_output_format* and the corresponding entry in *output_format_data_classes_dict*.

Parameters **model** (*Model*) – The *Model* instance to be updated.

analyze_galaxies(*snapshots: Optional[List[List[Union[int, str]]]] = 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[int, str]]]] = 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]

Submodules

sage_analysis.model module

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*)

Bases: `object`

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.

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

sage_analysis.sage_binary module

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*)

Bases: `sage_analysis.data_class.DataClass`

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`.

_check_for_file(*model: sage_analysis.model.Model*, *file_num: int*) → Optional[str]

Checks to see if a file for the given file number exists. Importantly, we check assuming that the path given in the **SAGE** parameter file is **relative** and **absolute**.

Parameters **file_num** (*int*) – The file number that we're checking for files.

Returns If a file exists, the name of that file. Otherwise, if the file does not exist (using either relative or absolute paths), then `None`.

Return type fname or `None`

_get_galaxy_struct()

Sets the `numpy` structured array for holding the galaxy data.

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.

sage_analysis.sage_hdf5 module

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*)
Bases: `sage_analysis.data_class.DataClass`

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`.

_check_model_compatibility (*model: sage_analysis.model.Model, sage_dict: Optional[Dict[str, Any]]*) → None
Ensures that the attributes in the *`Model`* instance are compatible with the variables read from the **SAGE** parameter file (if read at all).

Parameters

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

- **sage_dict** (*optional, dict[str, Any]*) – A dictionary containing all of the fields read from the **SAGE** parameter file.

> **Warning:**
>
> **UserWarning** Raised if the user initialized `Model` with a value of *`num_sage_output_files`* that is different to the value specified in the HDF5 file.

`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.

Python Module Index

S

Index

Symbols

A

B

C

D

F

G

H

I

L

M

N

O

P

R

S

U

V