ASLOTH

Release 1.0

Mattis Magg, Tilman Hartwig, Li-Hsin Chen, Yuta Tarumi

Jun 22, 2022

CONTENTS:

The semi-analytical model A-SLOTH (Ancient Stars and Local Observables by Tracing Halos) is the first public code that connects the formation of the first stars and galaxies to observables. The model is based on dark matter merger trees that can either be generated based on Extended Press-Schechter theory or that can be imported from dark matter simulations. On top of these merger trees, A-SLOTH applies analytical recipes for baryonic physics to model the formation of both metal-free and metal-poor stars and the transition between. A-SLOTH samples individual stars and includes radiative, chemical, and mechanical feedback. It is calibrated based on six observables, such as the optical depth to Thomson scattering, the stellar mass of the Milky Way and its satellite galaxies, the number of extremely-metal poor stars, and the cosmic star formation rate density at high redshift. A-SLOTH has versatile applications with moderate computational requirements. It can be used to constrain the properties of the first stars and high-z galaxies based on local observables, predicts properties of the oldest and most metal-poor stars in the Milky Way, can serve as a subgrid model for larger cosmological simulations, and predicts next-generation observables of the early Universe, such as supernova rates or gravitational wave events. More details on the astrophysical models can be found in our ApJ paper.

CHAPTER

ONE

GETTING STARTED

You can clone A-SLOTH to your machine with:

```
git clone https://gitlab.com/thartwig/asloth.git
```

If you encounter a problem or error at any point, you can consult the section on *debugging*, which contains a list of known problems and their solutions.

1.1 System Requirements and Dependencies

To run A-SLOTH you need a fortran compiler. We tested recent ifort and gfortran compilers, but you are welcone to try different ones. Set the correct compiler in the Makefile. A-SLOTH uses some features of the 2003 and 2008 standards of fortran, so very old compilers will probably not work. You will also need an Open-MP library if you want to run the code in parallel and python 3 and if you indend to use our plotting scripts. The Code is usually using a large amount of RAM. The fiducial EPS-based mode required about 4GB of RAM. We recommend 24-64 GB of RAM for modelling a full Milky Way based on the Caterpillar merger trees.

We have tested A-SLOTH and the provided tutorials on the following systems:

- Ubuntu 18.04, gfortran 7.5.0 / gfortran 10.3.0 / ifort 2021.5.0, python 3.8.3
- Ubuntu 20.04, gfortran 9.4.0, python 3.8.2
- CentOS Linux 7, ifort 2019.2, python 3.8.9
- MacOSX 10.14.6, gfortran 11.2.0 / gfortran 8.2.0, python 3.9.9 / python 3.6.13 / python 3.7.3
- MacOSX 10.15.7, gfortran 9.2.0, python 3.7.10

To execute the python analysis scripts, you might need to install the additional python packages (for example with `pip3 install PackageName`)

- numpy
- scipy
- matplotlib
- pandas
- psutil
- astropy

1.2 Tutorials

The following tutorials will guide you through the basic functions and applications of A-SLOTH. We recommand to complete these tutorials in order. A walk-through for the tutorials is provided on YouTube. The tutorials are based on the public release version of A-SLOTH with no other modifications. If a tutorial does not work, you might want to first reset A-SLOTH to the fiducial version, i.e., by checking `git status`, and by running `make clean`. Or you might need to first delete or rename previous output folders in the A-SLOTH directory. If not stated otherwise, all commands should be executed from the main A-SLOTH directory.

All python scripts in these tutorials should be run with python 3.

1.2.1 Tutorial 1: Hello A-SLOTH

Here, we summarize how you can run A-SLOTH with fiducial settings and check your outputs with standard plotting scripts. Change to the A-SLOTH folder and compile the code with

```
cd asloth/
make
```

Then you can run A-SLOTH with the command

```
./asloth.exe param_template.nml
```

If everything goes well, you should then see the statement

```
  ...
 Starting main star formation and feedback loop
 Current redshift:   0.0 PROGRESS: [################################]100.00% -- Complete
 A-SLOTH finished successfully.
```

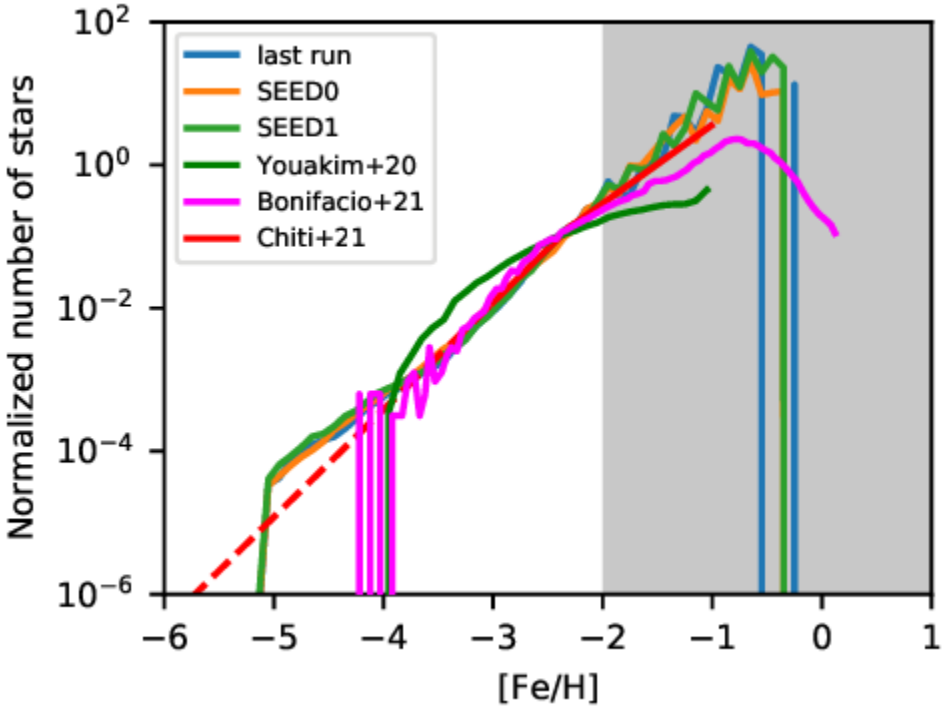
Now, we can have a look at the results by running python-based plotting scripts:

```
python scripts/plot_asloth.py
```

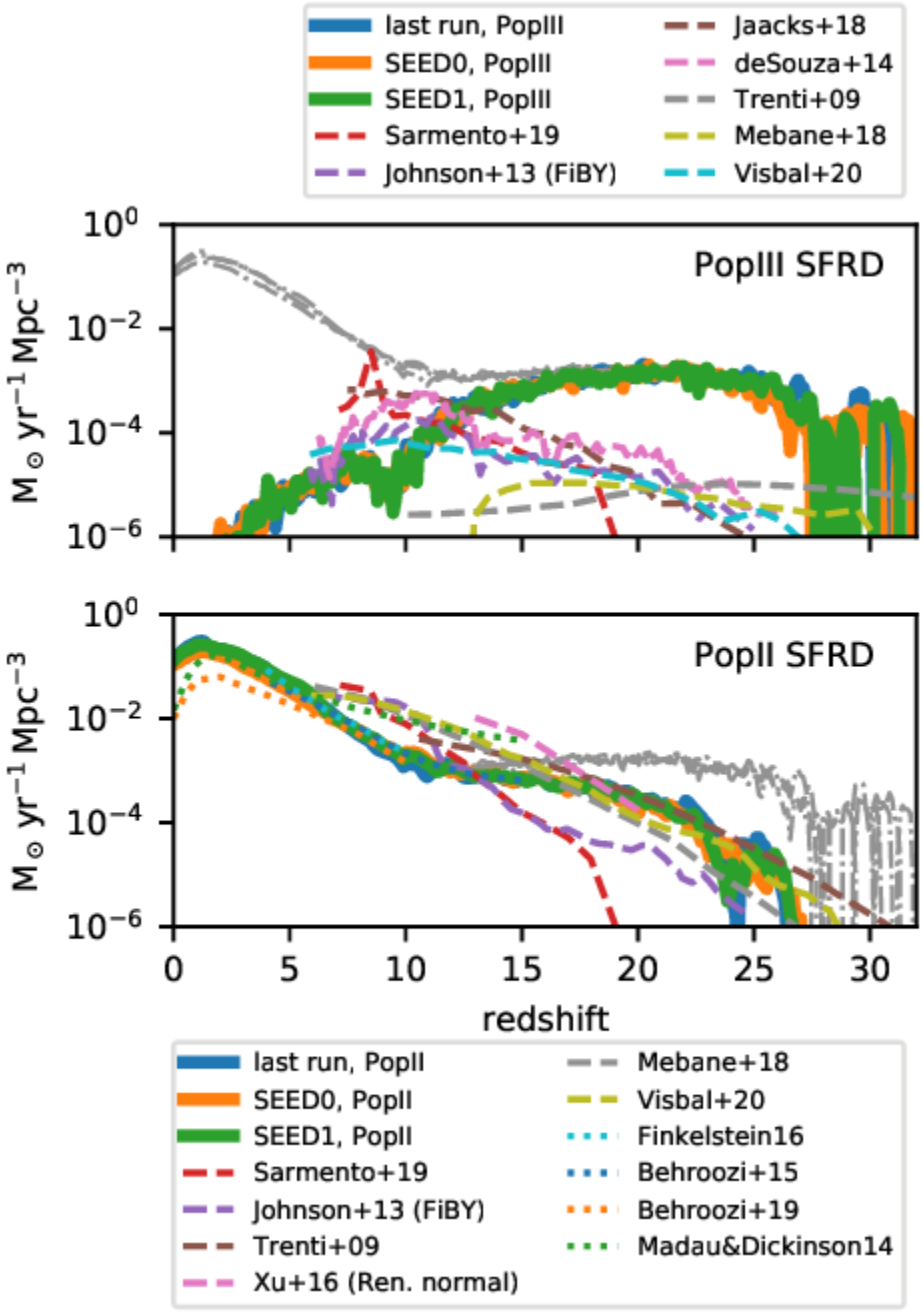
Ideally, the script will use the newest folder to visualize the results. It will also tell you in which folder the plots are saved:

```
...
The directory output_DATE_TIME exists and we will use it to create plots.
Plots will be saved in output_DATE_TIME/plots/
Adopted EMP fraction in MW: 2.314815e-05
...
```

The resulting figures should look like this:


last run
SEED0
SEED1
Youakim+20
Bonifacio+21
Chiti+21
Normalized number of stars
[Fe/H]

or this:



Please note: A-SLOTH is only deterministic if the same random seed, compiler, processor architecture, and optimisation flags are used. Therefore, your simulation output might look slightly different than this plot. As reference, we provide two additional reference values (SEED0, SEED1) with different random seeds to give you an idea how much the random seed can affect the results.

1.2.2 Tutorial 2: Vary one input parameter manually

In this tutorial, we investigate the effect of the Pop III IMF on the MDF. The fiducial model suggests Mmax=210Msun. Let's see how observables change if we modify this fiducial value to 100Msun or 300Msun (note that the mass range of PISNe is about 140-260Msun). This tutorial consists of two steps: first, we run the 3 different models (Mmax = [100,210,300]Msun). Then we will plot the MDFs from these 3 models

Step 1: Run the three models. To run the fiducial model with Mmax=210Msun, you need to run:

```
make clean
make
./asloth.exe param_template.nml
```

Once A-SLOTH has finished, you should have one folder named `output_YYYYMMDD_HHMMSS`. For better overview, you can rename this folder (not necessary, but helps later):

```
mv output_day_time output_Mmax210
```

Now, we want to run a second model with a different value for Mmax. For this purpose, we first copy the parameter file:

```
cp param_template.nml param_Mmax.nml
```

Then, we open the file `param_Mmax.nml` and modify the line with

```
IMF_max=210 ! maximum mass of PopIII IMF
```

to the desired value, let's say 100, and save the parameter file. Now we can run A-SLOTH again with this new parameter file:

```
./asloth.exe param_Mmax.nml
```

This should have generated another output folder that you can rename again. Now you can change `IMF_max` in `param_Mmax.nml` again and run A-SLOTH a third time. At the end of this step, you should have three folders with the results of the three runs that are named

```
output_Mmax100
output_Mmax210
output_Mmax300
```

Step 2: Plot the results. Open the file `scripts/plot_asloth.py` and scroll to the bottom. Change `UserFolder` to

```
UserFolder = ["output_Mmax100", "output_Mmax210", "output_Mmax300"]
```

and `UserLabels` to

```
UserLabels = ["Mmax=100Msun", "Mmax=210Msun", "Mmax=300Msun"]
```

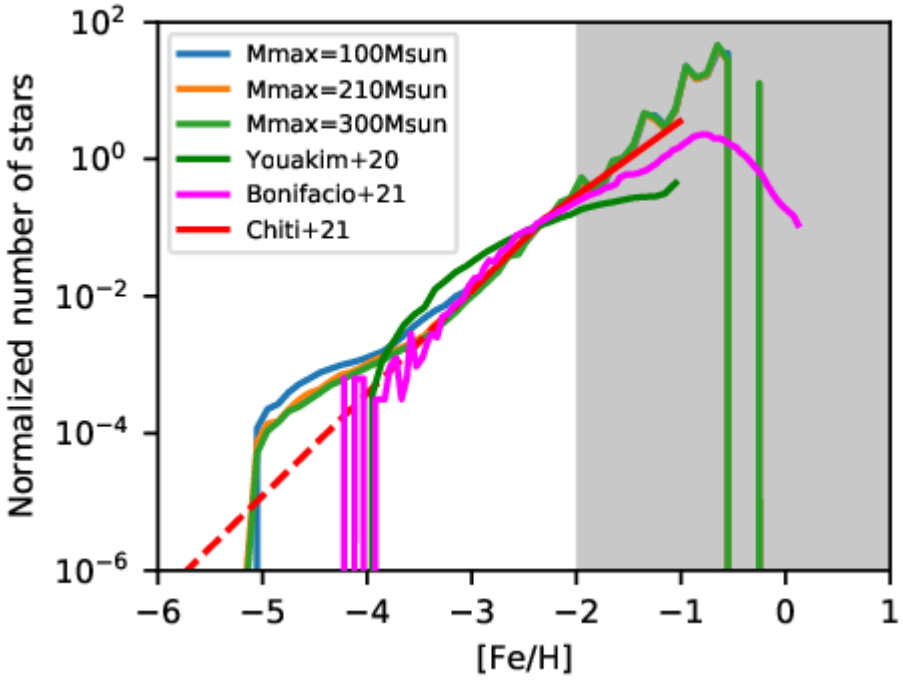
Furthermore, set `plot_newest_folder = False`. Save the file and run it with

```
python scripts/plot_asloth.py
```

The python scripts should have told you

```
...
Plots will be saved in output_Mmax100/plots/
...
```

(together with many other information). So let's look into this folder, which should contain the file `MW_MDF.pdf`, which should look like this (due to different random number generators or seeds, the figure might not look exactly like this



one):

We can see that the run with Mmax=100Msun (blue) differs from the other runs in the metallicity range below [Fe/H]=-3. Congratulations! You now know how to run and compare A-SLOTH with different input parameters manually.

1.2.3 Tutorial 3: Explore two input parameters automatically

In this tutorial we want to use A-SLOTH for a parameter exploration and see which observables are most sensitive to the details of PopIII star formation. Specifically, we want to vary the upper mass limit of the PopIII IMF and the PopIII star formation efficiency. Therefore, we run many realizations and sample random pairs of these two parameters. At the end, we visualize the effect of these parameters on the observables. Optional: you can set the variable `path_output` in `scripts/wrapper/loop_trees.py` and the variable `folder_name` in `scripts/wrapper/analyse_trees.py` to a folder in which you want to save the output. Otherwise, output will be written to your A-SLOTH folder. The following script will run for 15-30Min:

```
python scripts/wrapper/loop_trees.py
```

This script launches a python wrapper which will run 32 x 3 EPS merger trees:

1. Draws random values for Mmax and ETAIII
2. Checks if enough RAM and CPUs are available
3. Launches first tree
4. If sufficient resources are available, it also launches a second and third EPS tree with the same input parameters, but with a different random seed.

5. It continues with 1)-4) until either 32 x 3 trees are completed OR until 30Min are over.
6. If sufficient resources are not available (RAM, CPUs, disk I/O), it will wait a few seconds and try again.

Please note: the output to terminal might seem confusing because multiple trees are running at the same time. If the python script finishes before A-SLOTH, you might not get back to the command line prompt at the end of this run. If this is the case, you can press `Enter` once the last tree has finished.

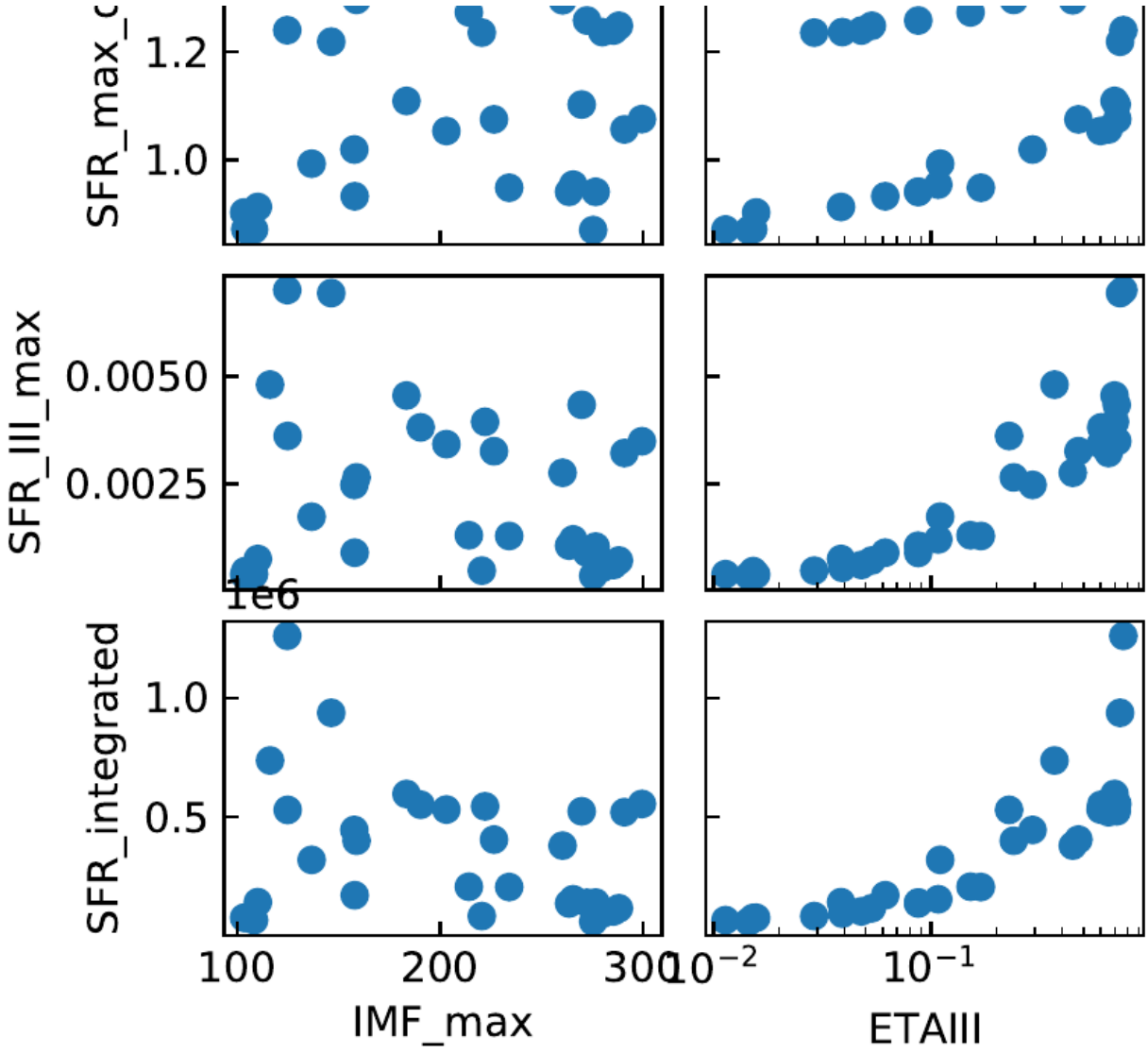
Once this script has finished, you should have many folders with the results in your `path_output` directory. In addition, the two figures `RAM_histogram.pdf` and `CPU_histogram.pdf` show you the available resources during the run. This is useful to check if any of these poses a bottleneck to the calculation. We have prepared another script to extract, digest, and compare the data of the output folders:

```
python scripts/wrapper/analyse_trees.py
```

This script goes through all the folders, collects the input parameters from each run, calculates the observables from A-SLOTH, and eventually compares them to real observables. During the run, plots are created in the output folders. The print messages will tell you in which folders you can find the individual plots. At the end, this script will produce the file `Parameter_Exploration.dat`. You can either look at this file manually, analyse it with your favourite software, or use a third script to visualize it:

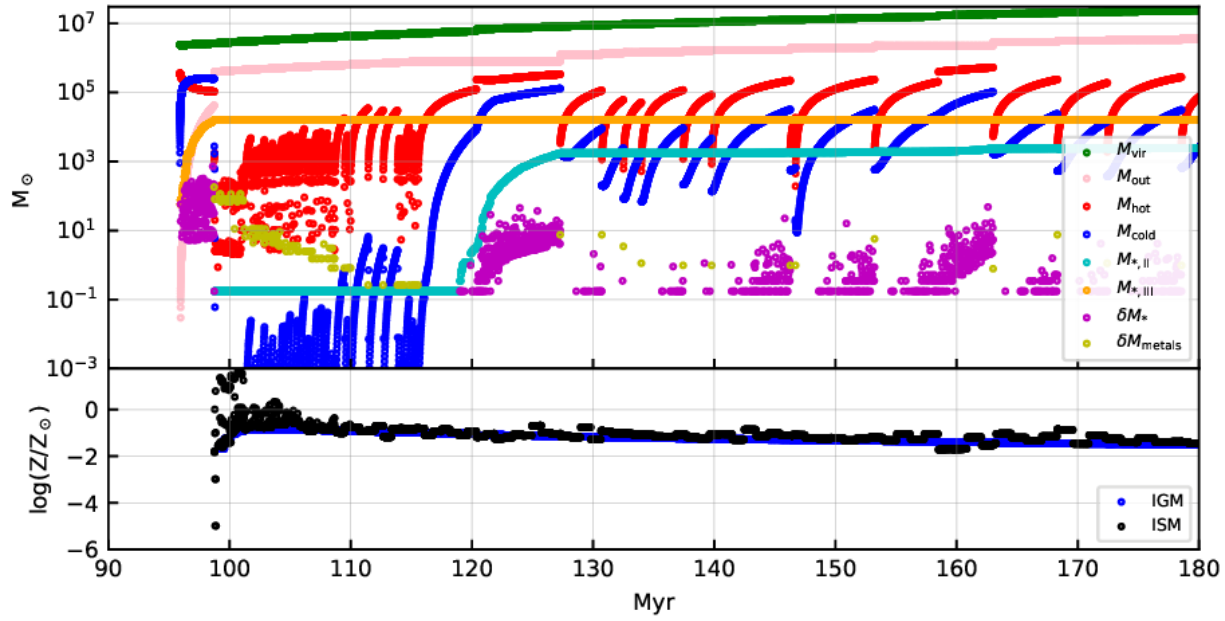
```
python scripts/wrapper/tutorial_GridPlot.py
```

The generated figure `Parameter_Exploration_tutorial.pdf` should look like this cut-out:


SFR_max_c
1.2
1.0
SFR_III_max
0.0050
0.0025
1e6
SFR_integrated
1.0
0.5
100
200
300
IMF_max
10^{-2}
10^{-1}
ETAIII

1.2.4 Tutorial 4: Analyse evolution of baryons in one merger tree branch

In this tutorial, we want to activate one additional compile time option in order to create a plot like this:



This plot shows the evolution of different baryonic quantities as function of cosmic time for one branch of the merger tree. Your plot might look slightly different depending on the random seed. For another example and explanations of the illustrated quantities, see Fig. 3 in Hartwig+22.

To generate the output for this plots, we have to activate the compiler flag `OUTPUT_GAS_BRANCH`. To do this, open the file `src/asloth.h` and activate the option by remeoving the first `!` in that line.

```
#define OUTPUT_GAS_BRANCH
```

You should recompile code with

```
make clean
make
```

Then, you can run the code with

```
./asloth.exe param_template.nml
```

In your `output_date_time` folder you should now have the additional file `t_gas_substeps.dat`. To illustrate these results, you can use a python script. But first, we have to set the output folder in the script. Open the file `scripts/utilities/plot_tgas.py` and in the very last line change `output_folder` to your output folder that was just created. Save the file and run

```
python scripts/utilities/plot_tgas.py
```

This should have generated the file `t_gas.pdf` in your output folder.

1.2.5 Tutorial 5: Output additional information from A-SLOTH

So far, we have used output files that were already implemented and written to the output folder. Now, we want to learn how users can implement their own desired output to a file. There are various options to write to a file. One can either keep a file open and write to it incrementally during the run. Or one can write all information to a file at once at the end of the run. In this example, we will look at the subroutine `write_files_reali()` in the file `src/to_file`, which is called at the end of the run to write outputs to files. In this file, you will find a section that starts with

```
!!! TUTORIAL 5: START !!!
```

At the moment, this code opens a file, writes a header, and closes the file. The do-loop iterates over all dark matter halos in this run and the write statement would write information for every halo. Such a file would be huge with millions of lines. Hence, it is currently deactivated by

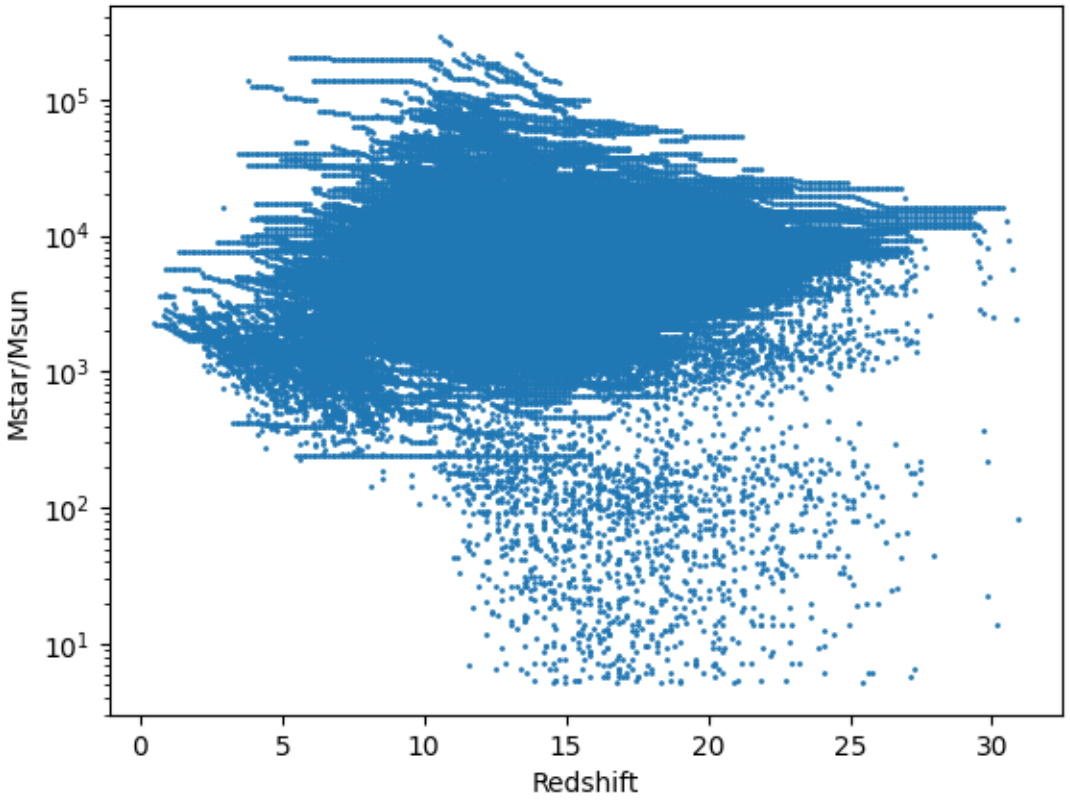
```
if(.False.)then
```

This routine should write the redshift and stellar mass of halos. Let's say we want this information only for PopIII-dominated halos, i.e., for halos that contain more PopIII stars than PopII stars.

Change the if-statement so that we only write information of PopIII-dominated halos to the file. Yes, you have to think how this if-statement looks like and can not simply copy it from the handbook as in previous tutorials. When you need additional information what the different node properties mean (Node%quantity), you can look in the file `src/defined_types.F90` under `type TreeNode`. Save the file once you have changed the if-statement. Then you can compile and run the code, as you have done in Tutorial 1. Again, you should get a new folder `output_date_time`.

To illustrate the results, you can open the python script `scripts/utilities/tutorial5.py`, set the correct `output_folder`, save it, and run it with

```
python scripts/utilities/tutorial5.py
```



Please note that the plot shows *cumulative* stellar masses, i.e., stellar masses that have formed until a certain redshift, but it does not account for stars that have already died based on their lifetime. While this plot is not immensly informative, this tutorial has shown you how to output and illustrate additional quantities from A-SLOTH. Happy coding!

1.3 Configure your simulation

The following files are relevant to change the general behaviour of the code:

- Parameter file. We provide the parameter file `param_template.nml`. We recommend to first create a copy of this file if you want to change these values. Numerical values in this file are calibrated to match observations, see Hartwig+22.
- Config file. You can find the available compile time options in the file `src/asloth.h`. You have to recompile the code if you change these compiler time options.
- Numerical parameters. You can find additional parameters in the file `src/num_pars.F90`. They are set to recommended values. You might want to change some of these parameters, such as the simulated redshift range, or the halos mass resolution.

CHAPTER

TWO

HOW TOS

2.1 Debugging

If A-SLOTH is not behaving as expected, this section might help. If the code is not compiling, please check the *code requirements*. A-SLOTH required several GB of RAM. Please check your available RAM before and during the run with `top` or `free -m`. If the code is running but crashes (e.g. segmentation fault) we recommend external debugging software such as valgrind. If the code is running, but the results seem strange to you, you can try the options below. If you think you found a bug, please report it via the GitLab issue tracker or by e-mail to hartwigATphys.s.uMINUStokyo.ac.jp.

2.1.1 Compiler Options

In the file `src/asloth.h` you can activate the compile time options `DEBUG`, `INFO`, and `MEM_REPORT`, which will provide additional outputs, especially if something is strange.

2.1.2 Python Scripts

The following python functions can be useful to identify problems with the code:

- `scripts/plot_asloth.py`
- `scripts/plot_tgas.py`

2.1.3 Additional Output

You can print additional information to the terminal or a file to check if variables during the run have the values that you expect. We provide the subroutine `print_all_props()` to print additional information of nodes (see file `src/defined_types.F90`).

2.2 Common Issues

2.2.1 Git clone does not work

If you get the error `SSL_ERROR_SYSCALL in connection to gitlab.com:443` or similar, you can also manually download the source code from the repository website.

2.2.2 The code does not compile

This can occur when you modified the 'asloth.h' or changed how you compile (e.g. the compiler and/or optimization flags) in the middle of compiling. It may also be helpful recompile the code entirely with

```
make clean
make
```

If the reason for being unable to compile is that the compiler is unable to find a module, please re-generate the dependency list via

```
python scripts/utilities/make_dependency_list.py
```

2.2.3 The generation of the dependency list fails

If the generation of the dependency list gets stuck and encounters a stack overflow after a while, most likely a circular dependency has been introduced. Be aware that if `module A` uses `module B`, neither `module B` nor any module that uses it can use `module A`

2.2.4 A-SLOTH does not accept my parameter.nml file

This file contains a set of fortran namelists. All namelists need to be present, even if they are empty. The names of all given parameters need to be identical to the names of the parameters in the specified namelist in the code. They also must have the correct type, and - if its an array - length. Missing quotation marks for a string or superflous quotation marks for a non-string variable are a likely reason.

2.2.5 The code crashes with a strange memory error

Make sure you compile with the options `-fbounds-check` (gfortran) or `-CB` (ifort). The most common reason for such memory errors is stepping out of bounds of an array. If the error only occurs with ifort but not gfortran, make sure to use the `-heap-arrays` flag for compiling. Compiling with `-g` (if you do not do so already) will give you slightly improved debugging information. If you recieve a segmentation fault while writing a very long line to a text file, try increasing the `recl` parameter when opening the offending file.

2.2.6 Compilation with ifort crashes

If your compilation crashes with

```
ifort -c  src/stellar/IMF.F90 -O3 -fpp -xHost -ipo -heap-arrays -g -r8 -D IFORT -shared-
↪intel -module mods -o build/stellar/IMF.o
src/stellar/IMF.F90(53): warning #6178: The return value of this FUNCTION has not been␣
↪defined.  [RES]
  function P_func_dummy(this, m1, m2) result(res)
-----------------------------------------^
```

you can try to use a newer version of the ifort compiler or the gfortran compiler instead.

2.2.7 A-SLOTH crashes with end of file error

```
At line 46 of file src/cosmic_time.F90
Fortran runtime error: End of file
```

Make sure that the value of `nlev` is correct in the parameter file.

2.2.8 The standard python plotting script does not work

If you try to plot output and it crashes with this error

```
asloth$ python scripts/plot_asloth.py
Traceback (most recent call last):
  File "scripts/plot_asloth.py", line 156, in <module>
    MyFit = plot_comparison(UserFolder, UserLabels=UserLabels)
  File "scripts/plot_asloth.py", line 51, in plot_comparison
    from utilities.utility import asloth_fit
ImportError: No module named utilities.utility
```

or similar, you can try to run it explicitely with python3 (you should always run the plotting scripts with python3).

2.3 Changing which chemical elements are traced

If you want to trace different elements, you can specify these in the parameter file under `ElementName`. You should then also update `N_ELEMENTS` so that N_ELEMENTS = len(ElementName)+1.

2.4 Get a tree

The code can be run in the EPS mode without any additional merger tree data. However, we recommend the EPS mode only for testing. To obtain reliable results, one should obtain dark matter merger trees from simulations. A-SLOTH uses its own binary format. Please contact the team on advice on how to obtain trees. The code includes a sample-script to convert merger trees from consistent-trees to our own format (reindex/reindex.F90, please look at the comments in the source code for instructions on how to use it).

2.5 Check Changes

If you have modified the source code and want to check the resulting differences, you can use the provided python scripts to do so. Let's say you ran the fiducial version of A-SLOTH and saved the output in the folder `output_v0`. Then you implement some new physics or improve some routine, run the code again and obtain a new output folder with the name `output_v1`.

You can visually compare the output of these two folders with the function `scripts/plot_asloth.py`. You should set

```
UserFolder = ["output_v0","output_v1"]
UserLabels = ["v0","v1"]
plot_newest_folder = False
```

and run it with `python3 scripts/plot_asloth.py`. You can use the produced plots to inspect the changes from your code update.

CHAPTER

THREE

REFERENCES AND LINKS

3.1 Articles by the ASLOTH collaboration

Hartwig et al. 2022
Magg et al. 2022
Chen et al. 2022
Tarumi et al. 2020
Hartwig et al. 2018
Magg et al. 2018
Magg et al. 2016
Hartwig et al. 2016b
Hartwig et al. 2016a
Hartwig et al. 2015

3.2 The Caterpillar Project

Caterpillar Website

CHAPTER

FOUR

LIST OF MODULES AND PROCEDURES

4.1 Main Code

4.1.1 ASLOTH main program

program asloth

Use

config, *defined_types*, *eps_wrapper*, *tree_memory_arrays_passable*, *numerical_parameters*, *resolution_parameters*, *to_file*, *input_files*, *check_flags*, *utility*, *volume_fractions*, *feedback_routines* (*node_feedback()*), *bubble_tree* (*start_btree()*), *tree_reader*, *tree_initialization*, *star_formation*, *read_yields*, *metal_functions*, *populations*, *sf_routines*, *crit_masses* (*set_mcrit()*), *cosmic_time* (*set_cosmic_time()*)

Call to

set_default_config(), *read_config()*, *check_compiler_flags()*, *init_outputs()*, *set_cosmic_time()*, *set_mcrit()*, *read_tree()*, *make_eps_tree()*, *init_stellar_populations()*, *assign_yields()*, *output_pop()*, *start_btree()*, *init_tree()*, *print_bar()*, *real_to_string3()*, *mem_report()*, *clean_tree()*, *get_fracs()*, *node_feedback()*, *sf_step()*, *add_parent_to_tree()*, *add_to_sfr()*, *add_to_feedback_volumes()*, *add_to_base_mdf()*, *add_to_base_mstariisurv()*, *write_files_reali()*, *outputs_satellites()*, *write_mw_properties()*, *write_files()*, *close_outputs()*

4.1.2 Configuration

config

Quick access

Variables

config_file, *cubic_box*, *nthreads*

Routines

read_config(), *set_default_config()*

Needed modules

- *numerical_parameters*
- *input_files*: List of input files
- *random_object*: This is an object oriented version of the ran3 algorithm from numerical recipes Before using a rng type object it needs to be initialized by calling rng%init(seed) where seed is an integer seed Afterwards rng%get_ran() will return a random real in the range [0,1)

Variables

- config/**config_file** *[character,protected]*
  e.g. param_template.nml
- config/**cubic_box** *[logical,protected]*
- config/**nthreads** *[integer,protected]*

Subroutines and functions

subroutine config/**set_default_config**()

Called from
asloth

subroutine config/**read_config**(*fname*)

Parameters
fname *[character,in]*

Use
omp_lib

Called from
asloth

4.1.3 Type definitions

defined_types

Quick access

Types
node_arrays, *stars_highmass*, *treenode*

Routines
add_baryons(), *del_high_mass()*, *make_stars_high_mass()*, *print_all_props()*, *set_baryons()*

Needed modules

- *metals* (*n_elements()*): main module for handling metals
- *utility*: General purpose utility functions

Types

- **type** defined_types/**treenode**

  class for storing information of each halo at each time

  Type fields

  - **% base** *[*treenode,*pointer]* :: which halo will this one be at at the the lowest redshift
  - **% child** *[*treenode,*pointer/optional/default=>null()]*
  - **% desc_id** *[integer]* :: descendent id
  - **% firstpopiisf** *[logical,optional/default=.true.]*
  - **% highstars** *[*stars_highmass,*pointer/optional/default=>]*
  - **% highstars_first** *[*stars_highmass,*pointer/optional/default=>]*
  - **% highstars_last** *[*stars_highmass,*pointer/optional/default=>]*
  - **% i_checked** *[logical]* :: has this halo already been checked for stochastic feedback?
  - **% i_enr** *[logical,optional/default=.false.]* :: is this halo enriched?
  - **% i_ion** *[logical,optional/default=.false.]* :: is the halo affected by ionising radiation
  - **% i_mcrit** *[logical,optional/default=.false.]* :: is the halo above the critical mass?
  - **% i_sf** *[logical,optional/default=.false.]* :: this flag controls whether the SF_step subroutine is called
  - **% i_sub** *[logical,optional/default=.false.]* :: is this a subhalo?
  - **% id** *[integer]* :: halo id
  - **% igm_z** (*) *[real,allocatable]* :: ambient metallicity
  - **% jlevel** *[integer]* :: which time-step is the halo at
  - **% l_ion** *[real]* :: ionising radiation emission rate [Photons/second]
  - **% llstars** *[*stars_highmass,*pointer/optional/default=>]*
  - **% llstars_first** *[*stars_highmass,*pointer/optional/default=>]*
  - **% llstars_last** *[*stars_highmass,*pointer/optional/default=>]*
  - **% lp_id** *[integer]* :: last progenitor depthfirst id
  - **% m_cold** *[real]* :: cold gas mass [M_sun]
  - **% m_filter** *[real,optional/default=0]*
  - **% m_hot** *[real]* :: hot gas mass [M_sun]
  - **% m_metals** (,) *[real,allocatable]*
  - **% m_out** *[real]* :: outflow mass [M_sun]
  - **% m_peak** *[real]* :: peak mass [Msun]

- **% m_peak_filter** *[real,optional/default=0]*
- **% m_starii** *[real]* :: stellar mass Pop II [M_sun]
- **% m_starii_surv** *[real,optional/default=0.0]* :: Msun
- **% m_stariii** *[real]* :: stellar mass Pop III [M_sun]
- **% mhalo** *[real]* :: mass of halo
- **% nchild** *[integer]* :: number of child nodes
- **% null** *[*node_arrays*,pointer]*
- **% num_popiiprogenitor** *[integer,optional/default=0]* :: how many Pop II forming progenitors
- **% parent** *[*treenode*,pointer/optional/default=>null()]*
- **% pop** *[integer]*
- **% r_en** *[real,optional/default=0]* :: enrichment radius [physical Mpc]
- **% r_ion** *[real,optional/default=0]* :: ionised radius [physical Mpc]
- **% sibling** *[*treenode*,pointer/optional/default=>null()]*
- **% this_array** *[*node_arrays*,pointer/optional/default=>]*
- **% x** (3) *[real]* :: position

- **type** defined_types/**node_arrays**

  Type fields

  - **% mdf_array** (,) *[real,allocatable]*

- **type** defined_types/**stars_highmass**

  Type fields

  - **% bin_id** *[integer,optional/default=0]* :: which bin it belongs, should be consistent with IMF arrays
  - **% next** *[*stars_highmass*,pointer/optional/default=>]*
  - **% null** *[*stars_highmass*,pointer]*
  - **% num** *[integer,optional/default=0]*
  - **% pop** *[integer,optional/default=0]* :: PopII (2) or PopIII (3) stars
  - **% prev** *[*stars_highmass*,pointer/optional/default=>]*
  - **% timeborn** *[real,optional/default=0.0]* :: in order to know when the stars die
  - **% z** (*) *[real,allocatable]* :: Zgas [Z/Zsun] when the star is born. Related with SN yields.

Subroutines and functions

function defined_types/**make_stars_high_mass**(*n*, *i*, *timeborn*, *zgas*, *pop*)

Parameters

- **n** *[integer,in]*
- **i** *[integer,in]*
- **timeborn** *[real,in]*
- **zgas** (*) *[real,in]*
- **pop** *[integer,in]*

Return
res *[*stars_highmass*]*

subroutine defined_types/**del_high_mass**(*this*)

Parameters
this *[*stars_highmass,*inout]*

subroutine defined_types/**set_baryons**(*this*, *other*)

copies baryonic properties from one halo to another (overwrites)

Parameters

- **this** *[real]*
- **other** *[real]*

subroutine defined_types/**add_baryons**(*this*, *other*)

adds baryonic properties of one halo to another

Parameters

- **this** *[real]*
- **other** *[real]*

subroutine defined_types/**print_all_props**(*a*)

Printing function that prints various porperties of a halo in order to allow easier tracing of errors and unexpected changes

Parameters
a *[*treenode,*in]*

4.1.4 Tree Reader

tree_reader

Quick access

Routines
read_tree()

Needed modules

- *defined_types*
- *input_files*: List of input files
- *numerical_parameters*
- *tree_memory_arrays*: This module is use for managing the memory of the merger tree
- *tree_memory_arrays_passable*: This module is used to reference the merger tree
- *utility*: General purpose utility functions
- *cosmic_time*

Subroutines and functions

subroutine tree_reader/**read_tree**(*count*, *read_mass*)

Parameters

- **count** *[integer,out]* :: reading tree
- **read_mass** *[real,out]*

Called from
asloth

Call to
print_bar(), *mem_report()*

4.1.5 Tree initialization

tree_initialization

Description

prepares tree for running asloth

Quick access

Routines
filter_virial_masses(), *init_tree()*, *set_tree_params()*

Subroutines and functions

subroutine tree_initialization/**init_tree**(*count*, *n_jcurrent*)

Parameters

- **count** *[integer,in]*
- **n_jcurrent** (*nlev_max*) *[integer,out]*

Use
defined_types, *numerical_parameters*, *tree_memory_arrays_passable*, *utility*, *crit_masses*, *cosmic_time*, *resolution_parameters*, *metalmix_dz*, *filter_module*, *trace_mdf*, *metals*

Called from
asloth

Call to
print_bar(), *mem_report()*, *read_lookups()*, *filter_virial_masses()*, *init_mdf()*

subroutine tree_initialization/**filter_virial_masses**()

Use
defined_types, *numerical_parameters*, *tree_memory_arrays_passable*, *filter_module*

Called from
init_tree()

Call to
compute_savgol_coeff(), *print_bar()*, *savgol_filter()*, *savgol_free()*

subroutine tree_initialization/**set_tree_params**()

Use
power_spectrum_parameters, modified_merger_tree, *numerical_parameters*, *input_files*, time_parameters

Called from
make_eps_tree()

4.1.6 numerical and resolution parameters

numerical parameters

Quick access

Variables
a_radiation, *ab_mag_norm*, *ab_norm*, *alpha_b*, *alpha_ion*, *alpha_outflow*, *ang2m*, *atomic_mass_helium*, *atomic_mass_hydrogen*, *bt_delta*, *bt_max*, *bt_min*, *btnbin*, *btnpad*, *c_cgs*, *c_hii*, *c_light*, *c_light_angstroms*, *c_light_cm*, *clump*, *cm32m3*, *crit_freq*, *deca*, *delta200*, *deltaeds*, *e_51*, *em_rdisk_half_scale_o2*, *eps3*, *eps4*, *eps6*, *epsm*, *epsr*, *erg2j*, *escape_fraction*, *etaii*, *etaiii*, *ev2erg*, *ev2j*, *f_escii*, *f_esciii*, *fx_peak_nfw*, *g*, *g_cgs*, *g_gyrkms3*, *g_mpcgyr*, *g_mpcgyr2*, *g_si*, *gpi*, *gyr2s*, *gyr2yr*, *h0100*, *h0100pgyr*, *h_planck*, *h_planck_erg*, *hecto*, *hubble*, *hubble_cgs*, *imf_max*, *imf_min*, *iso_fac*, *j2erg*, *k_boltzmann*, *k_boltzmann_erg*, *khorizon*, *kilo*, *kind99*, *km2m*, *kms2mpcgyr*, *kom*, *kpc2cm*, *kstrc_1*, *kstrc_2*, *l_ab0*, *l_box*, *ln10*, *ln2*, *log4*, *log_lsun_erg*, *logang2m*, *logc_light*, *logc_light_angstroms*, *logdeca*, *logev2erg*, *loggyr2s*, *loghecto*, *logj2erg*, *logk_boltzmann*, *logkilo*, *logl_ab0*, *loglsun*, *loglsun_bc*, *logm2cm*, *logmega*, *logmpc2asat10pc*, *logpc2m*, *logpi*, *lograd2as*, *long_bn*, *lsun*, *lsun40_bc*, *lsun_bc*, *lsun_erg*, *lw_mode*, *m2cm*, *m32cm3*, *m8crit*, *m_atomic*, *m_atomic_g*, *m_be*, *m_cha_out*, *m_electron*, *mean_mol*, *mega*, *mend*, *milli*, *mp_cgs*, *mpc2asat10pc*, *mpc2cm*, *mpc2m*, *mpc_cgs*, *mpckm2gyr*, *mres*, *msolar*, *msolar_1030kg*, *msolar_g*, *mstart*, *msun_cgs*, *msun_mpc3_cgs*, *mu_primordial*, *myr2s*, *n_b_cgs*, *n_cloud_default*, *n_cold_default*, *n_h_cgs*, *n_ionii*, *n_ioniii*, *n_spec*, *nextoutputid*, *nlev*, *nlev_max*, *nstarmassbinii*, *nstarmassbiniii*, *omega_b*, *omega_l*,

omega_m, *pc2cm*, *pc2m*, *pi*, *pi4*, *pio2*, *pio4*, *pisq*, *popii_slope*, *rad2as*, *rad2degrees*, *rdisk_half_scale*, *rho_b_ast*, *rho_b_cgs*, *rho_crit_cgs*, *rho_m_cgs*, *rhocrit*, *right_angle_degrees*, *semi_circle_degrees*, *sigma_8*, *sigma_t*, *sigma_thomson*, *slope*, *sqrt2*, *sqrt2opi*, *sqrt2pi*, *sqrtatomic_mass_hydrogen*, *sqrtdelta200*, *sqrtg*, *sqrtg_si*, *sqrtk_boltzmann*, *sqrtm_atomic*, *sqrtmpc2m*, *sqrtmsolar*, *sqrtpi*, *sqrtrhocrit*, *stdout_bn*, *surf_dens_norm_half*, *t_crit*, *t_ion*, *t_univ*, *tree_mass*, *v_com*, *v_enrich_popiii*, *v_out*, *vbc*, *x_hydrogen*, *x_hydrogen_solar*, *xh*, *y_helium*, *y_helium_solar*, *year_cgs*, *yhe*, *z_crit*, *z_metals*, *z_metals_solar*

Variables

- numerical_parameters/**a_radiation** *[real,parameter*=7.565743419463647e-16*]*
- numerical_parameters/**ab_mag_norm** *[real,parameter*=48.6*]*
- numerical_parameters/**ab_norm** *[real,parameter*=3.6307805477e-20*]*
- numerical_parameters/**alpha_b** *[real,parameter*=2.6e-13*]*
- numerical_parameters/**alpha_ion** *[real,parameter*=2.6e-13*]*
- numerical_parameters/**alpha_outflow** *[real]*
- numerical_parameters/**ang2m** *[real,parameter*=1e-10*]*
- numerical_parameters/**atomic_mass_helium** *[real,parameter*=4.002602*]*
- numerical_parameters/**atomic_mass_hydrogen** *[real,parameter*=1.00794*]*
- numerical_parameters/**bt_delta** *[real,optional/default=0.1]*
- numerical_parameters/**bt_max** *[real,optional/default=log10(4.e17)]*
- numerical_parameters/**bt_min** *[real,optional/default=log10(4.e15)]*
- numerical_parameters/**btnbin** *[integer,optional/default=10]*
- numerical_parameters/**btnpad** *[integer,optional/default=2]*
- numerical_parameters/**c_cgs** *[real,parameter*=29980000000.0*]*
- numerical_parameters/**c_hii** *[real,parameter*=3*]*
- numerical_parameters/**c_light** *[real,parameter*=299792458.0*]*
- numerical_parameters/**c_light_angstroms** *[real,parameter*=2.99792458e+18*]*
- numerical_parameters/**c_light_cm** *[real,parameter*=29979245800.0*]*
- numerical_parameters/**clump** *[real,parameter*=4.0*]*
- numerical_parameters/**cm32m3** *[real,parameter*=1e-06*]*
- numerical_parameters/**crit_freq** *[real,parameter*=3.969127815104263e-18*]*
- numerical_parameters/**deca** *[real,parameter*=10.0*]*
- numerical_parameters/**delta200** *[real,parameter*=200.0*]*

- numerical_parameters/**deltaeds** *[real,parameter*=177.65287922076283*]*
- numerical_parameters/**e_51** *[real,parameter*=1.2*]*
- numerical_parameters/**em_rdisk_half_scale_o2** *[real,parameter*=0.432067481*]*
- numerical_parameters/**eps3** *[real,parameter*=0.001*]*
- numerical_parameters/**eps4** *[real,parameter*=0.0001*]*
- numerical_parameters/**eps6** *[real,parameter*=1e-06*]*
- numerical_parameters/**epsm** *[real,parameter*=0.0001*]*
- numerical_parameters/**epsr** *[real,parameter*=1e-06*]*
- numerical_parameters/**erg2j** *[real,parameter*=1e-07*]*
- numerical_parameters/**escape_fraction** *[real]*
- numerical_parameters/**etaii** *[real]*
- numerical_parameters/**etaiii** *[real]*
- numerical_parameters/**ev2erg** *[real,parameter*=1.60217733e-12*]*
- numerical_parameters/**ev2j** *[real,parameter*=1.60217733e-19*]*
- numerical_parameters/**f_escii** *[real]*
- numerical_parameters/**f_esciii** *[real]*
- numerical_parameters/**fx_peak_nfw** *[real,parameter*=0.2162165956*]*
- numerical_parameters/**g** *[real,parameter*=4.301307710181788e-09*]*
- numerical_parameters/**g_cgs** *[real,parameter*=6.674e-08*]*
- numerical_parameters/**g_gyrkms3** *[real,parameter*=4.205785220992724e-06*]*
- numerical_parameters/**g_mpcgyr** *[real,parameter*=4.398999721935946e-12*]*
- numerical_parameters/**g_mpcgyr2** *[real,parameter*=4.498910530810333e-15*]*
- numerical_parameters/**g_si** *[real,parameter*=6.67259e-11*]*
- numerical_parameters/**gpi** *[real,parameter*=1.3212863952890194e-05*]*
- numerical_parameters/**gyr2s** *[real,parameter*=3.15576e+16*]*
- numerical_parameters/**gyr2yr** *[real,parameter*=1000000000.0*]*
- numerical_parameters/**h0100** *[real,parameter*=100.0*]*
- numerical_parameters/**h0100pgyr** *[real,parameter*=0.10227121653079844*]*
- numerical_parameters/**h_planck** *[real,parameter*=6.6260755e-34*]*
- numerical_parameters/**h_planck_erg** *[real,parameter*=6.6260755e-27*]*
- numerical_parameters/**hecto** *[real,parameter*=100.0*]*

- numerical_parameters/**hubble** *[real,parameter*=0.6774*]*
- numerical_parameters/**hubble_cgs** *[real,parameter*=2.1952879411478756e-18*]*
- numerical_parameters/**imf_max** *[real]*
- numerical_parameters/**imf_min** *[real]*
- numerical_parameters/**iso_fac** *[real,parameter*=1.085736204755322*]*
- numerical_parameters/**j2erg** *[real,parameter*=10000000.0*]*
- numerical_parameters/**k_boltzmann** *[real,parameter*=1.3806503e-23*]*
- numerical_parameters/**k_boltzmann_erg** *[real,parameter*=1.3806503e-16*]*
- numerical_parameters/**khorizon** *[real,parameter*=0.00033356409519815205*]*
- numerical_parameters/**kilo** *[real,parameter*=1000.0*]*
- numerical_parameters/**kind99** *[integer,parameter*=selected_real_kind(p,*r=99)]*
- numerical_parameters/**km2m** *[real,parameter*=1000.0*]*
- numerical_parameters/**kms2mpcgyr** *[real,parameter*=0.0010227121653079844*]*
- numerical_parameters/**kom** *[real,parameter*=2.9434011554855882e-21*]*
- numerical_parameters/**kpc2cm** *[real,parameter*=3.0856775807e+21*]*
- numerical_parameters/**kstrc_1** *[real,parameter*=0.6251543028*]*
- numerical_parameters/**kstrc_2** *[real,parameter*=1.191648617*]*
- numerical_parameters/**l_ab0** *[real,parameter*=43442114331434.14*]*
- numerical_parameters/**l_box** *[real]*
  (linear box size)
- numerical_parameters/**ln10** *[real,parameter*=2.302585093*]*
- numerical_parameters/**ln2** *[real,parameter*=0.6931471806*]*
- numerical_parameters/**log4** *[real,parameter*=0.602059991*]*
- numerical_parameters/**log_lsun_erg** *[real,parameter*=33.5848963*]*
- numerical_parameters/**logang2m** *[real,parameter*=-10.0*]*
- numerical_parameters/**logc_light** *[real,parameter*=8.4768207*]*
- numerical_parameters/**logc_light_angstroms** *[real,parameter*=18.476820699999998*]*
- numerical_parameters/**logdeca** *[real,parameter*=1.0*]*
- numerical_parameters/**logev2erg** *[real,parameter*=-11.7952894176*]*
- numerical_parameters/**loggyr2s** *[real,parameter*=16.499103967*]*
- numerical_parameters/**loghecto** *[real,parameter*=2.0*]*

- numerical_parameters/**logj2erg** *[real,parameter*=7.0*]*
- numerical_parameters/**logk_boltzmann** *[real,parameter*=-22.859916308*]*
- numerical_parameters/**logkilo** *[real,parameter*=3.0*]*
- numerical_parameters/**logl_ab0** *[real,parameter*=13.637910954000002*]*
- numerical_parameters/**loglsun** *[real,parameter*=26.5848963*]*
- numerical_parameters/**loglsun_bc** *[real,parameter*=26.5868122*]*
- numerical_parameters/**logm2cm** *[real,parameter*=2.0*]*
- numerical_parameters/**logmega** *[real,parameter*=6.0*]*
- numerical_parameters/**logmpc2asat10pc** *[real,parameter*=10.31442513287*]*
- numerical_parameters/**logpc2m** *[real,parameter*=16.489350545*]*
- numerical_parameters/**logpi** *[real,parameter*=0.497149873*]*
- numerical_parameters/**lograd2as** *[real,parameter*=5.31442513287*]*
- numerical_parameters/**long_bn** *[integer,parameter*=selected_int_kind(16)*]*
- numerical_parameters/**lsun** *[real,parameter*=3.845e+26*]*
- numerical_parameters/**lsun40_bc** *[real,parameter*=3.862e-07*]*
- numerical_parameters/**lsun_bc** *[real,parameter*=3.862e+26*]*
- numerical_parameters/**lsun_erg** *[real,parameter*=3.8450000000000004e+33*]*
- numerical_parameters/**lw_mode** *[integer]*
- numerical_parameters/**m2cm** *[real,parameter*=100.0*]*
- numerical_parameters/**m32cm3** *[real,parameter*=1000000.0*]*
- numerical_parameters/**m8crit** *[real,parameter*=595167835572453.2*]*
- numerical_parameters/**m_atomic** *[real,parameter*=1.66053873e-27*]*
- numerical_parameters/**m_atomic_g** *[real,parameter*=1.66053873e-24*]*
- numerical_parameters/**m_be** *[real,parameter*=1000.0*]*
- numerical_parameters/**m_cha_out** *[real]*
- numerical_parameters/**m_electron** *[real,parameter*=9.10938188e-31*]*
- numerical_parameters/**mean_mol** *[real,parameter*=1.2281617489023304*]*
- numerical_parameters/**mega** *[real,parameter*=1000000.0*]*
- numerical_parameters/**mend** *[real,parameter*=2.0*]*
  , Mcutoff !, MLongLive
- numerical_parameters/**milli** *[real,parameter*=0.001*]*

- numerical_parameters/**mp_cgs** *[real,parameter*=1.673e-24*]*
- numerical_parameters/**mpc2asat10pc** *[real,parameter*=20626480624.64262*]*
- numerical_parameters/**mpc2cm** *[real,parameter*=3.0856775807e+24*]*
- numerical_parameters/**mpc2m** *[real,parameter*=3.0856775807e+22*]*
- numerical_parameters/**mpc_cgs** *[real,parameter*=3.0857e+24*]*
- numerical_parameters/**mpckm2gyr** *[real,parameter*=977.79222143002*]*
- numerical_parameters/**mres** *[real,parameter*=500000.0*]*
- numerical_parameters/**msolar** *[real,parameter*=1.9891e+30*]*
- numerical_parameters/**msolar_1030kg** *[real,parameter*=1.9890999999999999*]*
- numerical_parameters/**msolar_g** *[real,parameter*=1.9891e+33*]*
- numerical_parameters/**mstart** *[real,parameter*=-2.0*]*
- numerical_parameters/**msun_cgs** *[real,parameter*=1.989e+33*]*
- numerical_parameters/**msun_mpc3_cgs** *[real,parameter*=6.769766377778569e-41*]*
- numerical_parameters/**mu_primordial** *[real,parameter*=0.5847493469151946*]*
- numerical_parameters/**myr2s** *[real,parameter*=31557600000000.0*]*
- numerical_parameters/**n_b_cgs** *[real,parameter*=2.038744909036926e-07*]*
- numerical_parameters/**n_cloud_default** *[real,parameter*=1000*]*
  ! dense, and diffuse density in the cold phase
- numerical_parameters/**n_cold_default** *[real,parameter*=100.0*]*
- numerical_parameters/**n_h_cgs** *[real,parameter*=1.5337477950684792e-07*]*
- numerical_parameters/**n_ionii** *[real,parameter*=4000.0*]*
- numerical_parameters/**n_ioniii** *[real,parameter*=90000.0*]*
- numerical_parameters/**n_spec** *[real,parameter*=0.9667*]*
- numerical_parameters/**nextoutputid** *[integer]*
- numerical_parameters/**nlev** *[integer]*
- numerical_parameters/**nlev_max** *[integer,parameter*=400*]*
- numerical_parameters/**nstarmassbinii** *[integer,parameter*=64*]*
- numerical_parameters/**nstarmassbiniii** *[integer,parameter*=64*]*
- numerical_parameters/**omega_b** *[real,parameter*=0.0486*]*
- numerical_parameters/**omega_l** *[real,parameter*=0.6911*]*
- numerical_parameters/**omega_m** *[real,parameter*=0.3089*]*

- numerical_parameters/**pc2cm** *[real,parameter*=3.0856775807e+18*]*
- numerical_parameters/**pc2m** *[real,parameter*=3.0856775807e+16*]*
- numerical_parameters/**pi** *[real,parameter*=3.1415926536*]*
- numerical_parameters/**pi4** *[real,parameter*=12.5663706144*]*
- numerical_parameters/**pio2** *[real,parameter*=1.5707963268*]*
- numerical_parameters/**pio4** *[real,parameter*=0.7853981634*]*
- numerical_parameters/**pisq** *[real,parameter*=9.86960440115349*]*
- numerical_parameters/**popii_slope** *[real,parameter*=2.3*]*
- numerical_parameters/**rad2as** *[real,parameter*=206264.80624642622*]*
- numerical_parameters/**rad2degrees** *[real,parameter*=57.29577951289617*]*
- numerical_parameters/**rdisk_half_scale** *[real,parameter*=1.67834699*]*
- numerical_parameters/**rho_b_ast** *[real,parameter*=6187863374.547873*]*
- numerical_parameters/**rho_b_cgs** *[real,parameter*=4.189038942330162e-31*]*
- numerical_parameters/**rho_crit_cgs** *[real,parameter*=8.619421692037372e-30*]*
- numerical_parameters/**rho_m_cgs** *[real,parameter*=2.6625393606703443e-30*]*
- numerical_parameters/**rhocrit** *[real,parameter*=277511434571.3454*]*
- numerical_parameters/**right_angle_degrees** *[real,parameter*=90.0*]*
- numerical_parameters/**semi_circle_degrees** *[real,parameter*=180.0*]*
- numerical_parameters/**sigma_8** *[real,parameter*=0.8159*]*
- numerical_parameters/**sigma_t** *[real,parameter*=6.65e-25*]*
- numerical_parameters/**sigma_thomson** *[real,parameter*=6.65245854e-29*]*
- numerical_parameters/**slope** *[real]*
- numerical_parameters/**sqrt2** *[real,parameter*=1.4142135624*]*
- numerical_parameters/**sqrt2opi** *[real,parameter*=0.7978845608*]*
- numerical_parameters/**sqrt2pi** *[real,parameter*=2.5066282746*]*
- numerical_parameters/**sqrtatomic_mass_hydrogen** *[real,parameter*=1.00396215*]*
- numerical_parameters/**sqrtdelta200** *[real,parameter*=14.142135623*]*
- numerical_parameters/**sqrtg** *[real,parameter*=6.558435567945511e-05*]*
- numerical_parameters/**sqrtg_si** *[real,parameter*=8.16859228998e-06*]*
- numerical_parameters/**sqrtk_boltzmann** *[real,parameter*=3.715710295489e-12*]*
- numerical_parameters/**sqrtm_atomic** *[real,parameter*=4.07497083425e-14*]*

- numerical_parameters/**sqrtmpc2m** *[real,parameter*=175660968365.0*]*
- numerical_parameters/**sqrtmsolar** *[real,parameter*=1410354565300000.0*]*
- numerical_parameters/**sqrtpi** *[real,parameter*=1.7724538509*]*
- numerical_parameters/**sqrtrhocrit** *[real,parameter*=526793.5407696816*]*
- numerical_parameters/**stdout_bn** *[integer,parameter*=10*]*
- numerical_parameters/**surf_dens_norm_half** *[real,parameter*=0.0297114*]*
- numerical_parameters/**t_crit** *[real,parameter*=2200.0*]*
- numerical_parameters/**t_ion** *[real,parameter*=10000.0*]*
- numerical_parameters/**t_univ** *[real,parameter*=4.3549644e+17*]*
- numerical_parameters/**tree_mass** *[real]*
- numerical_parameters/**v_com** *[real]*
- numerical_parameters/**v_enrich_popiii** *[real,parameter*=11000000.0*]*
- numerical_parameters/**v_out** *[real,parameter*=11000000.0*]*
- numerical_parameters/**vbc** *[real]*
- numerical_parameters/**x_hydrogen** *[real,parameter*=0.778*]*
- numerical_parameters/**x_hydrogen_solar** *[real,parameter*=0.7381*]*
- numerical_parameters/**xh** *[real,parameter*=0.7523*]*
- numerical_parameters/**y_helium** *[real,parameter*=0.222*]*
- numerical_parameters/**y_helium_solar** *[real,parameter*=0.2485*]*
- numerical_parameters/**year_cgs** *[real,parameter*=31560000.0*]*
- numerical_parameters/**yhe** *[real,parameter*=0.2477*]*
- numerical_parameters/**z_crit** *[real,parameter*=-5.0*]*
- numerical_parameters/**z_metals** *[real,parameter*=5.36e-10*]*
- numerical_parameters/**z_metals_solar** *[real,parameter*=0.0134*]*

resolution parameters

Quick access

Variables
m_ccsn_max, *m_ccsn_min*, *m_ion*, *m_pisn_max*, *m_pisn_min*, *m_survive*, *zmax*, *zmin*

Variables

- `resolution_parameters/`**`m_ccsn_max`** *[real,parameter*=40*]*
  Mass range for CCSNe
- `resolution_parameters/`**`m_ccsn_min`** *[real,parameter*=10*]*
- `resolution_parameters/`**`m_ion`** *[real,parameter*=5*]*
  Stellar mass above which we account for ionizing radition
- `resolution_parameters/`**`m_pisn_max`** *[real,parameter*=260*]*
  Mass range for PISNe
- `resolution_parameters/`**`m_pisn_min`** *[real,parameter*=140*]*
- `resolution_parameters/`**`m_survive`** *[real,parameter*=0.81*]*
  Stellar mass with lifetime 13.8Gyr.
- `resolution_parameters/`**`zmax`** *[real,parameter*=35.0*]*
- `resolution_parameters/`**`zmin`** *[real,parameter*=0.0*]*

4.1.7 Precomputed time-dependent quantities

crit_masses

Quick access

Variables
atomic_cooling_mass, *mcrit*

Routines
get_atomic_cooling_mass(), *get_crit_mass()*, *get_m_vir()*, *get_mcrit_schauer()*, *get_mcrit_stacy()*, *get_t_vir()*, *set_mcrit()*

Needed modules

- *numerical_parameters*
- *cosmic_time*

Variables

- `crit_masses/`**`atomic_cooling_mass`** (*nlev_max*) *[real,protected]*
- `crit_masses/`**`mcrit`** (*nlev_max*) *[real,protected]*

Subroutines and functions

subroutine crit_masses/**set_mcrit**()

Called from
asloth

Call to
get_crit_mass(), *get_atomic_cooling_mass()*

function crit_masses/**get_crit_mass**(*z*)

Determines critical mass for star formation at current redshift

Parameters
z *[real,in]* :: redshift

Return
get_crit_mass *[real]*

Called from
set_mcrit()

Call to
get_mcrit_schauer(), *get_mcrit_stacy()*, *get_atomic_cooling_mass()*

function crit_masses/**get_mcrit_stacy**(*z*)

Modified to also check Jeans Mass calculated with modified speed of sound See Stacy et al. 2011

Parameters
z *[real,in]* :: redshift

Return
get_mcrit_stacy *[real]*

Called from
get_crit_mass()

Call to
get_m_vir()

function crit_masses/**get_mcrit_schauer**(*z*)

based on Schauer+21, MNRAS, Volume 507, Issue 2, pp.1775-1787 Critical mass for halo collapse with LW feedback and baryonic streaming

Parameters
z *[real,in]* :: redshift

Return
get_mcrit_schauer *[real]*

Called from
get_crit_mass()

function crit_masses/**get_atomic_cooling_mass**(*z*)

the atomic cooling mass is the virial mass of 10^4 K

Parameters
z *[real,in]* :: redshift

Return
get_atomic_cooling_mass *[real]*

Called from
set_mcrit(), *get_crit_mass()*

Call to
get_m_vir()

function crit_masses/**get_m_vir**(*z*, *t*)

function determines the halo mass corresponding to a virial temperature we use the definition of Hummel et al 2012

Parameters

- **z** *[real,in]* :: redshift
- **t** *[real,in]* :: virial temperature

Return
get_m_vir *[real]*

Called from
get_mcrit_stacy(), *get_atomic_cooling_mass()*

function crit_masses/**get_t_vir**(*m*, *z*)

Parameters

- **m** *[real,in]*
- **z** *[real,in]*

Return
t *[real]*

Called from
sf_step()

cosmic_time

Quick access

Variables
a3_inv, *alev*, *tlev*, *zlev*

Routines
cosmic_dt(), *distribute_cosmic_time()*, *dtdz()*, *read_scale_file()*, *set_cosmic_time()*

Needed modules

- *numerical_parameters* (*nlev()*, *nlev_max()*)
- *resolution_parameters* (*zmin()*, *zmax()*)
- *input_files* (*scale_file()*): List of input files

Variables

- cosmic_time/**a3_inv** (*nlev_max*) *[real,protected]*
  1/(a^3)
- cosmic_time/**alev** (*nlev_max*) *[real,protected]*
  cosmic expansion factor
- cosmic_time/**tlev** (*nlev_max*) *[real,protected]*
  time since big bang
- cosmic_time/**zlev** (*nlev_max*) *[real,protected]*
  redshift

Subroutines and functions

subroutine cosmic_time/**set_cosmic_time**()

subroutine to set redshift steps and compute expansion factors and cosmic time

Called from
asloth

Call to
read_scale_file(), *distribute_cosmic_time()*, *cosmic_dt()*

subroutine cosmic_time/**distribute_cosmic_time**()

subroutine to distribute redshift steps (either linearly or logarithmically)

Called from
set_cosmic_time()

subroutine cosmic_time/**read_scale_file**()

subroutine to read the redshift steps from scale file

Called from
set_cosmic_time()

function cosmic_time/**dtdz**(*a*)

Subroutine to calculate absolute value of dt/dz in seconds for density field normalized at reference epoch a0=1.

Notation:

Parameters
a *[real,in]* :: expansion factor rel to a0=1

Uses

Return
res *[real]*

Use
numerical_parameters (*hubble_cgs()*, *omega_l()*, *omega_m()*)

Called from
cosmic_dt()

function cosmic_time/**cosmic_dt**(*z1*, *z2*)

determines the time in seconds between redshift z1 and z2

Parameters

- **z1** *[real,in]*
- **z2** *[real,in]*

Return
res *[real]*

Called from
set_cosmic_time()

Call to
dtdz()

4.1.8 EPS wrapper

eps_wrapper

Description

module to interface with the Parkinson et al. 2008 tree generation routines

Quick access

Routines
make_eps_tree()

Needed modules

- power_spectrum_parameters: Variables used to hold properties of the power spectrum
- make_tree_module
- *tree_memory_arrays*: This module is use for managing the memory of the merger tree
- *tree_memory_arrays_passable*: This module is used to reference the merger tree
- tree_routines: % A set of subroutines and functions for manipulating and utilizing the merger trees.
- modified_merger_tree
- *defined_types* (*treenode()*)
- *numerical_parameters* (*mres()*)
- *resolution_parameters* (*zmin()*)
- *tree_initialization* (*set_tree_params()*): prepares tree for running asloth

Subroutines and functions

subroutine eps_wrapper/**make_eps_tree**()

Called from
asloth

Call to
set_tree_params(), memory(), make_tree(), walk_tree()

4.1.9 Chemistry

chemistry

Quick access

Variables
abhe, *x_ion*

Routines
ch21(), *ch22()*, *cl4()*, *cl64()*, *fh2_crit()*, *h2_cooling_rate()*, *h2_form_rate()*, *ph4()*, *t_hot2cold()*

Needed modules

- *numerical_parameters*

Variables

- chemistry/**abhe** *[real,parameter=y_helium/x_hydrogen/4.0]*
- chemistry/**x_ion** *[real,parameter=0.0002]*
  residual ionization fraction

Subroutines and functions

function chemistry/**t_hot2cold**(*n*, *t*, *z*)

Parameters

- **n** *[real,in]*
- **t** *[real,in]*
- **z** *[real,in]*

Return
res *[real]*

Called from
timescaledetermination()

Call to
fh2_crit(), *h2_form_rate()*

function chemistry/**h2_form_rate**(*n*, *t*, *z*)

Parameters

- **n** *[real,in]*
- **t** *[real,in]*
- **z** *[real,in]*

Return
res *[real]*

Called from
t_hot2cold()

Call to
ch21(), *ph4()*, *ch22()*

function chemistry/**fh2_crit**(*n*, *t*, *z*)

Parameters

- **n** *[real,in]*
- **t** *[real,in]*
- **z** *[real,in]*

Return
res *[real]*

Called from
t_hot2cold()

Call to
h2_cooling_rate()

function chemistry/**h2_cooling_rate**(*n*, *t*)

Parameters

- **n** *[real,in]*
- **t** *[real,in]*

Return
res *[real]*

Called from
fh2_crit()

Call to
cl4(), *cl64()*

function chemistry/**ch22**(*temp*)

Parameters
temp *[real,in]*

Return
res *[real]*

Called from
h2_form_rate()

function chemistry/**ch21**(*temp*)

Parameters
temp *[real,in]*

Return
res *[real]*

Called from
h2_form_rate()

function chemistry/**ph4**(*z*)

Parameters
z *[real,in]*

Return
res *[real]*

Called from
h2_form_rate()

function chemistry/**cl64**(*temp*)

Parameters
temp *[real,in]*

Return
res *[real]*

Called from
h2_cooling_rate()

function chemistry/**cl4**(*temp*)

Parameters
temp *[real,in]*

Return
res *[real]*

Called from
h2_cooling_rate()

4.1.10 Input files

Description

List of input files

Quick access

Variables
lines_lt_popii, *lines_mmetals_popii*, *lines_qion_popii*, *mtree_file*, *output_string*, *pkinfile_dummy*, *popii_lt_file*, *popii_mmetals_file*, *popii_qion_file*, *popii_yields_file*, *popiii_yields_file*, *scale_file*, *tree_file*, *tree_name*, *tree_path*

Variables

- input_files/**lines_lt_popii** *[integer]*
- input_files/**lines_mmetals_popii** *[integer,parameter*=27*]*
- input_files/**lines_qion_popii** *[integer]*
- input_files/**mtree_file** *[character,parameter*='tree_files/mass_estimates_lx13.dat'*]*
- input_files/**output_string** *[character]*
- input_files/**pkinfile_dummy** *[character,parameter*='data/kvector_z1.txt'*]*
  is later passed on to pkinfile
- input_files/**popii_lt_file** *[character]*
- input_files/**popii_mmetals_file** *[character,parameter*='data/kobayashi2006_z1e-3_mmetal.txt'*]*
- input_files/**popii_qion_file** *[character]*
- input_files/**popii_yields_file** *[character,parameter*='data/kobayashi2006_z1e-3.txt'*]*
- input_files/**popiii_yields_file** *[character,parameter*='data/yields_isotopes_sum_tr.dat'*]*
- input_files/**scale_file** *[character]*
- input_files/**tree_file** *[character]*
- input_files/**tree_name** *[character]*
- input_files/**tree_path** *[character]*

4.1.11 Memory modules

Tree_memory_arrays

Description

This module is use for managing the memory of the merger tree

Quick access

Variables
mergertree_aux

Needed modules

- *defined_types*

Variables

- tree_memory_arrays/**mergertree_aux** (*) *[*treenode,*target/allocatable]*

Tree_memory_arrays_passable

Description

This module is used to reference the merger tree

Quick access

Variables
mergertree, *number_of_nodes*

Routines
tree_index()

Needed modules

- *defined_types*

Variables

- tree_memory_arrays_passable/**mergertree** (*) *[*treenode,*pointer]*
- tree_memory_arrays_passable/**number_of_nodes** *[integer]*

Subroutines and functions

function tree_memory_arrays_passable/**tree_index**(*node*)

Parameters
node *[*treenode,*pointer]*

Return
tree_index *[integer,pure]*

Called from
associate_siblings()

4.1.12 Star formation

SF_routines

Quick access

Routines

add_hmstars(), *add_to_base_mstariisurv()*, *coldgasbindingenergy_cold()*, *coldgasbindingenergy_hot()*, *coldgasbindingenergy_nfw()*, *coldgasbindingenergy_stellar()*, *dm_concen()*, *get_n_ion()*, *hms_feedback()*, *hotgasbindingenergy_disk()*, *hotgasbindingenergy_hot()*, *hotgasbindingenergy_nfw()*, *imfsampling()*, *nfw_const()*, *rm_hmstars()*, *snedrivenoutflow()*, *survivingstarlist()*, *timescaledetermination()*, *writellstarstofileanddel()*

Needed modules

- *defined_types*
- *numerical_parameters*
- *resolution_parameters*
- *random_object*: This is an object oriented version of the ran3 algorithm from numerical recipes Before using a rng type object it needs to be initialized by calling rng%init(seed) where seed is an integer seed Afterwards rng%get_ran() will return a random real in the range [0,1)
- *utility*: General purpose utility functions
- *metal_functions*: Helper functions for handling metals and computing abundances and metallicities
- *trace_mdf*
- *metals*: main module for handling metals
- *populations*: This module contains the type for storing IMFs, and the properties of the stellar populations
- *chemistry*
- *cosmic_time* (*zlev()*, *tlev()*)

Subroutines and functions

subroutine sf_routines/**add_hmstars**(*p_node*, *ii*, *timeborn*, *num_*, *zgas*, *pop*)

Parameters

- **p_node** *[*treenode*,inout]*
- **ii** *[integer,in]*
- **timeborn** *[real,in]*
- **num** *[integer,in]*
- **zgas** (*n_elements*) *[real,in]*
- **pop** *[integer,in]*

Called from

imfsampling()

subroutine sf_routines/**rm_hmstars**(*p_node*, *highstar*)

Parameters

- **p_node** *[*treenode*,inout]*
- **highstar** *[*stars_highmass*,inout,pointer]*

Called from
hms_feedback()

function sf_routines/**get_n_ion**(*p_node*, *t_current*, *del_t*)

This routine calculates the total number of ionizing photons emitted during this time-step

Parameters

- **p_node** *[*treenode*,inout]* :: the halo
- **t_current** *[real,in]* :: beginning of the sub-step
- **del_t** *[real,in]* :: time-step

Return
n_ion_tot *[real]* :: number of total ionizing photons

Called from
sf_step()

subroutine sf_routines/**hms_feedback**(*p_node*, *t_current*, *totalmetalejectathisstep*, *totalsneenergythisstep*, *m_heat*, *rel_del_t*)

This routine calculates the cold gas mass that will be heated/ blown out from the halo and the explosion energy/ ejecta from SNe.

Parameters

- **p_node** *[*treenode*,inout]* :: the halo
- **t_current** *[real,in]* :: beginnng of the sub-step
- **totalmetalejectathisstep** (*n_elements*) *[real,inout]* :: metals produced in this time-step
- **totalsneenergythisstep** *[real,out]* :: Msun cm^2 s^-2
- **m_heat** *[real,out]* :: mass that will trasfer from cold to hot in this time-step
- **rel_del_t** *[real,in]* :: time-step

Called from
sf_step()

Call to
add_to_m_metals(), *rm_hmstars()*

subroutine sf_routines/**add_to_base_mstariisurv**(*this_node*)

Inherit M_star_II_surv to the base node at the final redshift

Parameters
this_node *[*treenode*,inout]*

Called from
asloth

subroutine sf_routines/**survivingstarlist**(*p_node*, *num*, *ii*, *z*, *timeborn*, *pop*)

Non-standard routine if users are interested in stellar information of survival stars in satellites

Parameters

- **p_node** *[*treenode*,inout]* :: the halo
- **num** *[integer,in]* :: number of stars in this IMF mass bin
- **ii** *[integer,in]* :: index of the IMF mass bin
- **z** (*n_elements*) *[real,in]* :: metallicity of the star
- **timeborn** *[real,in]* :: time of star formation
- **pop** *[integer,in]* :: stellar population

Called from
imfsampling()

subroutine sf_routines/**writellstarstofileanddel**(*p_node*)

Parameters
p_node *[*treenode*,inout]*

Called from
sf_step()

subroutine sf_routines/**imfsampling**(*p_node*, *totalmassnewstars*, *m_star_form*, *t_current*, *this_rng*, *zgas*, *pop*)

This routine samples individual stars from the IMF and store massive stars in the list.

Parameters

- **p_node** *[*treenode*,inout]* :: the halo
- **totalmassnewstars** *[real,inout]* :: total mass of newly formed stars
- **m_star_form** *[real,in]* :: estimated total stellar mass based on cold gas mass and star formation efficiency
- **t_current** *[real,in]* :: the time in the beginning of the time-step
- **this_rng** *[*rng*,inout]* :: random number generator
- **zgas** (*n_elements*) *[real,in]* :: gas metallicity
- **pop** *[*population*,in]* :: stellar population

Use
trace_mdf

Called from
sf_step()

Call to
get_mdf_bin(), *poisson()*, *add_hmstars()*, *survivingstarlist()*, *add_to_node_mdf()*

subroutine sf_routines/**dm_concen**(*m_halo*, *z*, *dm_concentration*)

This routine determins the concentration of the NFW profile. We adopt the fitting curve by Correa et al. 2015 (doi:10.1093/mnras/stv1363) Planck cosmology (appendix B) Note that the fitting functions are valid for z=0-10 based on private conversation

Parameters

- **m_halo** *[real,in]* :: virial mass of the halo
- **z** *[real,in]* :: current redshift

- **dm_concentration** *[real,out]* :: (arbitrary) constant value, reference from Fig. 7 in Correa et al. 2015 for haloes > 10^5 Msun

Called from
sf_step()

subroutine sf_routines/**timescaledetermination**(*p_node*, *v_dyn*, *t_dyn*, *ncold*, *n_hot*, *coldgas_tff*, *r_vir_current*, *dm_concentration*, *t_vir*, *t_cool*)

This routines determines all the relative timescales that we need. Note that the time scales we eventually use are dimensionless.

Parameters

- **p_node** *[*treenode*,inout]* :: This also includes "dead" stars
- **v_dyn** *[real,inout]* :: s; dynamical time of the halo center
- **t_dyn** *[real,inout]* :: dynamical timescale of central part of the halo
- **ncold** *[real,inout]* :: number density of cold gas
- **n_hot** *[real,inout]* :: number density of hot gas
- **coldgas_tff** *[real,inout]* :: free fall time of cold gas
- **r_vir_current** *[real,in]* :: the current virial raius of the halo
- **dm_concentration** *[real,in]* :: dark matter halo concentration
- **t_vir** *[real,in]* :: virial temperature of the halo
- **t_cool** *[real,inout]* :: cooling time of hot gas

Called from
sf_step()

Call to
t_hot2cold()

function sf_routines/**nfw_const**(*r_s*, *r_vir_current*)

Parameters

- **r_s** *[real,in]*
- **r_vir_current** *[real,in]*

Return
a *[real]*

Called from
hotgasbindingenergy_nfw(), *coldgasbindingenergy_nfw()*

function sf_routines/**hotgasbindingenergy_nfw**(*r_s*, *r_vir_current*, *m_hot*, *m_peak*)

Parameters

- **r_s** *[real,in]*
- **r_vir_current** *[real,in]*
- **m_hot** *[real,in]*
- **m_peak** *[real,in]*

Return
be_hot_nfw *[real]*

Called from
snedrivenoutflow()

Call to
nfw_const()

function sf_routines/**hotgasbindingenergy_disk**(*r_s*, *r_vir_current*, *m_hot*, *m_disk*)

Parameters

- **r_s** *[real,in]*
- **r_vir_current** *[real,in]*
- **m_hot** *[real,in]*
- **m_disk** *[real,in]*

Return
be_hot_disk *[real]*

Called from
snedrivenoutflow()

function sf_routines/**hotgasbindingenergy_hot**(*r_vir_current*, *m_hot*)

Parameters

- **r_vir_current** *[real,in]*
- **m_hot** *[real,in]*

Return
be_hot_hot *[real]*

Called from
snedrivenoutflow()

function sf_routines/**coldgasbindingenergy_nfw**(*r_s*, *r_vir_current*, *m_cold*, *m_peak*)

Parameters

- **r_s** *[real,in]*
- **r_vir_current** *[real,in]*
- **m_cold** *[real,in]*
- **m_peak** *[real,in]*

Return
be_cold_nfw *[real]*

Called from
snedrivenoutflow()

Call to
nfw_const()

function sf_routines/**coldgasbindingenergy_stellar**(*r_s*, *m_cold*, *m_stellar*)

Parameters

- **r_s** *[real,in]*
- **m_cold** *[real,in]*

- **m_stellar** *[real,in]*

Return
be_cold_stellar *[real]*

Called from
snedrivenoutflow()

function sf_routines/**coldgasbindingenergy_hot**(*r_s*, *r_vir_current*, *m_cold*, *m_hot*)

Parameters

- **r_s** *[real,in]*
- **r_vir_current** *[real,in]*
- **m_cold** *[real,in]*
- **m_hot** *[real,in]*

Return
be_cold_hot *[real]*

Called from
snedrivenoutflow()

function sf_routines/**coldgasbindingenergy_cold**(*r_s*, *m_cold*)

Parameters

- **r_s** *[real,in]*
- **m_cold** *[real,in]*

Return
be_cold_cold *[real]*

Called from
snedrivenoutflow()

subroutine sf_routines/**snedrivenoutflow**(*p_node*, *r_vir_current*, *dm_concentration*, *totalsneenergythisstep*, *del_m_hot*, *totalmassnewstars*, *m_out_hot*, *m_out_cold*)

This routines determines the binding energy of hot gas and cold gas. It then compare the SNe energy and the binding energy to determine how much gas SNe will blow out of the halo.

Parameters

- **p_node** *[*treenode*,inout]* :: the halo
- **r_vir_current** *[real,in]* :: current virial radius of the halo
- **dm_concentration** *[real,in]* :: dark matter halo's concentration
- **totalsneenergythisstep** *[real,inout]* :: sum of SNe energy that occur in this time-step
- **del_m_hot** *[real,in]* :: smoothly accreted hot gas in this time-step
- **totalmassnewstars** *[real,in]* :: total newly formed stellar mass in this time-step
- **m_out_hot** *[real,inout]* :: hot gas mass that ends up in outflow in this time-step
- **m_out_cold** *[real,inout]* :: cold gas mass that ends up in outflow in this time-step

Called from
sf_step()

Call to
hotgasbindingenergy_nfw(), *hotgasbindingenergy_disk()*, *hotgasbindingenergy_hot()*, *coldgasbindingenergy_nfw()*, *coldgasbindingenergy_stellar()*, *coldgasbindingenergy_hot()*, *coldgasbindingenergy_cold()*

Star_formation

Quick access

Routines
add_parent_to_tree(), *dt_adaptive()*, *m_star_dot()*, *prepare_node()*, *sf_step()*

Needed modules

- *bubble_tree*: This module containes routines for adding feedback of haloes to the tracing system and for testing whether a halo or a specific point in space is affected by the feedback
- *defined_types*
- *numerical_parameters*
- *resolution_parameters*
- *metal_functions*: Helper functions for handling metals and computing abundances and metallicities
- *snowplow*
- *random_object*: This is an object oriented version of the ran3 algorithm from numerical recipes Before using a rng type object it needs to be initialized by calling rng%init(seed) where seed is an integer seed Afterwards rng%get_ran() will return a random real in the range [0,1)
- *sf_routines*
- *input_files*: List of input files
- *crit_masses* (*get_t_vir()*)
- *utility* (*iunit_list()*): General purpose utility functions
- *virial_radius* (*rvir_mpc()*)
- *igm_z*: utility routine for inheriting the IGM metallicity accross time
- *metalmix_dz*

Subroutines and functions

subroutine star_formation/**prepare_node**(*p_node*, *m_star_max*, *is_ion*)

Parameters

- **p_node** *[*treenode*,inout]* :: parent node
- **m_star_max** *[real,out]*
- **is_ion** *[logical,out]* :: set from most massive child

Called from
sf_step()

Call to
check_m_metals(), *inherit_igm_z()*

subroutine star_formation/**sf_step**(*node*)

Parameters
node *[*treenode*,inout,target]*

Called from
asloth

Call to
prepare_node(), *rvir_mpc()*, *dm_concen()*, *get_t_vir()*, *check_m_metals()*, *get_metallicity()*, *calc_dz()*, *metalpoor()*, *timescaledetermination()*, *dt_adaptive()*, *m_star_dot()*, *imfsampling()*, *get_n_ion()*, *hms_feedback()*, *snedrivenoutflow()*, *assert()*, *v_plow_ad()*, *r_ion_ad()*, *writellstarstofileanddel()*

subroutine star_formation/**add_parent_to_tree**(*node*)

Parameters
node *[*treenode*,inout]*

Called from
asloth

Call to
add_bubble_to_tree(), *get_m_metals()*

function star_formation/**dt_adaptive**(*p_node*, *coldgas_tff*, *t_dyn*, *del_m_hot*, *del_t*, *t_step*, *pop_now*)

timestep in sec

Parameters

- **p_node** *[*treenode*,in]*
- **coldgas_tff** *[real,in]*
- **t_dyn** *[real,in]*
- **del_m_hot** *[real,in]*
- **del_t** *[real,in]*
- **t_step** *[real,in]*
- **pop_now** *[integer,in]*

Return
dt_adaptive *[real]*

Called from
sf_step()

Call to
m_star_dot()

function star_formation/**m_star_dot**(*m_cold*, *coldgas_tff*, *pop_now*)

star formation rate in this substep (Msun/s)

Parameters

- **m_cold** *[real,in]*
- **coldgas_tff** *[real,in]*

- **pop_now** *[integer,in]*

Return
m_star_dot *[real]*

Use
numerical_parameters (*etaii()*, *etaiii()*)

Called from
sf_step(), *dt_adaptive()*

4.1.13 Stellar feedback

feedback_routines

Quick access

Routines
node_feedback(), *self_enrichment()*, *statistical_feedback()*

Needed modules

- *bubble_tree*: This module containes routines for adding feedback of haloes to the tracing system and for testing whether a halo or a specific point in space is affected by the feedback
- *defined_types*
- *feedback_arrays* (*v_enriched()*, *v_ionized()*)
- *crit_masses*

Subroutines and functions

subroutine feedback_routines/**node_feedback**(*this_node*)

Parameters
this_node *[*treenode,*target]*

Called from
asloth

Call to
self_enrichment(), *statistical_feedback()*, *bubble_check()*, *ion_check()*

subroutine feedback_routines/**self_enrichment**(*node*)

This routine computes feedback exactly as the bubble-tree in the case that there is only self-enrichment

Parameters
node *[*treenode*]*

Called from
node_feedback()

subroutine feedback_routines/**statistical_feedback**(*this_node*)

Parameters
this_node *[*treenode,*target]*

Use
random_object

Called from
node_feedback()

Call to
get_ran()

snowplow

Quick access

Routines
r_ion_ad(), *v_plow_ad()*

Needed modules

- *numerical_parameters*
- *virial_radius*

Subroutines and functions

`function` snowplow/**v_plow_ad**(*m*, *z*, *r*, *r_v*)

Parameters

- **m** *[real,in]*
- **z** *[real,in]*
- **r** *[real,in]*
- **r_v** *[real,in]*

Return
v *[real]*

Called from
sf_step()

`function` snowplow/**r_ion_ad**(*nion*, *r_old*, *z*, *del_t*, *mhalo*)

Parameters

- **nion** *[real,in]*
- **r_old** *[real,in]*
- **z** *[real,in]*
- **del_t** *[real,in]*
- **mhalo** *[real,in]*

Return
r_new *[real]*

Called from
sf_step()

Call to
rvir_mpc(), *rvir()*

4.1.14 Feedback tracing

feedback_arrays

Quick access

Variables
sfr, *v_enriched*, *v_ionized*

Routines
add_to_feedback_volumes(), *add_to_sfr()*

Needed modules

- *numerical_parameters* (*nlev_max()*, *year_cgs()*, *pi()*)
- *defined_types*
- *cosmic_time* (*tlev()*)

Variables

- feedback_arrays/**sfr** (2,*nlev_max*) *[real,protected/optional/default=0]*
  SFR: population, timestep
- feedback_arrays/**v_enriched** (*nlev_max*) *[real,protected/optional/default=0]*
- feedback_arrays/**v_ionized** (*nlev_max*) *[real,protected/optional/default=0]*
  (in physical Mpc^3)

Subroutines and functions

subroutine feedback_arrays/**add_to_feedback_volumes**(*node*)

Parameters
node *[*treenode*]*

Called from
asloth

subroutine feedback_arrays/**add_to_sfr**(*node*)

Parameters
node *[*treenode*]*

Called from
asloth

volume_fractions

Quick access

Routines
get_fracs()

Needed modules

- *cosmic_time* (*alev()*, *zlev()*)
- *bubble_tree*: This module containes routines for adding feedback of haloes to the tracing system and for testing whether a halo or a specific point in space is affected by the feedback
- *input_files*: List of input files
- *numerical_parameters*
- *feedback_arrays* (*v_ionized()*, *v_enriched()*)
- *config* (*cubic_box()*)
- *utility* (*iunit_list()*): General purpose utility functions

Subroutines and functions

subroutine volume_fractions/**get_fracs**(*jlevel*)

Parameters
jlevel *[integer,in]*

Use
random_object

Called from
asloth

Call to
get_ran(), *position_check()*

4.1.15 Outputs

to_file

Quick access

Types
file_specs

Routines
add_timestamp(), *close_outputs()*, *init_outputs()*, *outputs_satellites()*, *write_file()*, *write_files()*, *write_files_reali()*, *write_mw_properties()*

Needed modules

- *resolution_parameters*
- *numerical_parameters*
- *input_files*: List of input files
- *utility*: General purpose utility functions
- *converters*: Module for converting numerical values to strings
- *config* (*config_file()*)
- *cosmic_time*
- *defined_types*
- *feedback_arrays*
- *virial_radius* (*rvir_mpc()*)

Types

- **type** to_file/**file_specs**

  Type fields

  - **% file_string** *[character]*
  - **% head1** *[character]*
  - **% head2** *[character]*
  - **% out_dir** *[character]*
  - **% stuff** (30,*nlev_max*) *[real]*
  - **% var_name** *[character]*

Variables

Subroutines and functions

subroutine to_file/**init_outputs**()

Called from
asloth

Call to
add_timestamp()

subroutine to_file/**write_file**(*fs*, *var_name*, *ncoll*)

Parameters

- **fs** *[*file_specs*]*
- **var_name** *[character]*
- **ncoll** *[integer]* :: loop writes one line

Called from
write_files()

subroutine to_file/**outputs_satellites**(*count*)

Parameters
count *[integer]*

Use
tree_memory_arrays_passable, *trace_mdf*

Called from
asloth

Call to
rvir_mpc(), *add_to_node_mdf()*

subroutine to_file/**write_files_reali**()

Use
tree_memory_arrays_passable, *input_files*, *metal_functions*, *trace_mdf*

Called from
asloth

Call to
rvir_mpc(), *add_to_node_mdf()*

subroutine to_file/**write_files**(*output*)

Parameters
output *[*file_specs*,inout]*

Called from
asloth

Call to
real_to_string(), *write_file()*

subroutine to_file/**close_outputs**()

go through list of possible open file IDs

Called from
asloth

subroutine to_file/**add_timestamp**(*output_path*)

Parameters
output_path *[character,inout]*

Called from
init_outputs()

Call to
int_to_string4(), *int_to_string2()*

subroutine to_file/**write_mw_properties**()

Use
tree_memory_arrays_passable, *input_files*

Called from
asloth

4.2 Utility modules

4.2.1 Converters

Description

Module for converting numerical values to strings

Quick access

Routines
bool2int(), *int_to_string2()*, *int_to_string3()*, *int_to_string4()*, *real_to_string()*, *real_to_string3()*

Subroutines and functions

function converters/**real_to_string**(*numb*)

Parameters
numb *[real,in]*

Return
string *[real]*

Called from
write_files()

function converters/**real_to_string3**(*numb*)

Parameters
numb *[real,in]*

Return
string *[real]*

Called from
asloth

function converters/**int_to_string2**(*numb*)

Parameters
numb *[integer,in]*

Return
string *[real]*

Called from
add_timestamp()

function converters/**int_to_string3**(*numb*)

Parameters
numb *[integer,in]*

Return
string *[real]*

function converters/**int_to_string4**(*numb*)

Parameters
numb *[integer,in]*

Return
string *[real]*

Called from
add_timestamp()

function converters/**bool2int**(*boo*)

Parameters
boo *[logical,in]*

Return
int_out *[integer]*

4.2.2 Savitzki-Golay filter

Description

Fortran module with Savitzki-Golay filter to remove high-frequency noise from data The module is optimized for a case in which many arrays need to be smoothed with the same filter. It is intended for smoothing noisy virial masses dependencies: lapack

Quick access

Variables
coeff_array, *coeff_matrix*, *deriv*, *half_window*, *order*, *rhs_array*, *window_size*

Routines
compute_savgol_coeff(), *fact()*, *pad_data()*, *savgol_alloc()*, *savgol_filter()*, *savgol_free()*, *set_params()*, *set_up_lin_eqs()*

Needed modules

- *utility*: General purpose utility functions

Variables

- filter_module/**coeff_array** (*) *[real,private/allocatable/save]*
- filter_module/**coeff_matrix** (*,*) *[real,private/allocatable/save]*
- filter_module/**deriv** *[integer,private/save]*
- filter_module/**half_window** *[integer,private/save]*
- filter_module/**order** *[integer,private/save]*
- filter_module/**rhs_array** (*,*) *[real,private/allocatable/save]*
- filter_module/**window_size** *[integer,private/save]*

Subroutines and functions

subroutine filter_module/**compute_savgol_coeff**(*window_size_in*, *order_in*, *deriv_in*)

Parameters

- **window_size_in** *[integer,in]*
- **order_in** *[integer,in]*
- **deriv_in** *[integer,in]*

Called from
filter_virial_masses()

Call to
assert(), *set_params()*, *savgol_free()*, *savgol_alloc()*, *set_up_lin_eqs()*, *fact()*

subroutine filter_module/**savgol_filter**(*y*, *ylen*, *y_out*)

Parameters

- **y** (ylen) *[real,in]*
- **y_out** (ylen) *[real,out]*

Options
ylen *[integer,in,optional/default=len(y)]*

Called from
filter_virial_masses()

Call to
assert(), *pad_data()*

subroutine filter_module/**pad_data**(*y*, *ylen*, *half_window_pass*, *y_padded*)

Parameters

- **y** (ylen) *[real,in]*
- **half_window_pass** *[integer,in]*
- **y_padded** (ylen+half_window_pass-(-half_window_pass+1)+1) *[real,out]*

Options
ylen *[integer,in,optional/default=len(y)]*

Called from
savgol_filter()

subroutine filter_module/**savgol_alloc**()

Called from
compute_savgol_coeff()

subroutine filter_module/**savgol_free**()

Called from
filter_virial_masses(), *compute_savgol_coeff()*

subroutine filter_module/**set_params**(*window_size_in*, *order_in*, *deriv_in*)

Parameters

- **window_size_in** *[integer,in]*
- **order_in** *[integer,in]*
- **deriv_in** *[integer,in]*

Called from
compute_savgol_coeff()

subroutine filter_module/**set_up_lin_eqs**()

Called from
compute_savgol_coeff()

function filter_module/**fact**(*n*)

Parameters
n *[integer,in]*

Return
fact *[integer]*

Called from
compute_savgol_coeff()

Call to
assert()

4.2.3 Utility

Description

General purpose utility functions

Quick access

Variables
current_cpu_time, *getpid*, *iunit_list*, *start_cpu_time*

Routines
assert(), *i_realloc_1d()*, *mem_report()*, *print_bar()*, *r_realloc_1d()*, *r_realloc_2d()*

Needed modules

- *input_files*: List of input files
- *numerical_parameters* (stdout())
- ifport

Variables

- utility/**current_cpu_time** *[real]*
- utility/**getpid** *[intrinsic]*
- utility/**iunit_list** (16) *[integer]*
  list of file IDs
- utility/**start_cpu_time** *[real]*

Subroutines and functions

subroutine utility/**mem_report**()

Called from
init_tree(), *read_tree()*, *asloth*

subroutine utility/**assert**(*test_passed*, *message*)

Parameters

- **test_passed** *[logical,in]*
- **message** *[character,in]*

Called from
metalpoor(), *get_element_info()*, *sf_step()*, *kroupa_p_func()*, *kroupa_m_func()*, *massprobabilityimf()*, *compute_savgol_coeff()*, *savgol_filter()*, *fact()*

subroutine utility/**r_realloc_1d**(*a*, *n_new*)

Parameters

- **a** (*) *[real,inout,allocatable]*
- **n_new** *[integer,in]*

subroutine utility/**i_realloc_1d**(*a*, *n_new*)

Parameters

- **a** (*) *[integer,inout,allocatable]*
- **n_new** *[integer,in]*

subroutine utility/**r_realloc_2d**(*a*, *n1_new*, *n2_new*)

Parameters

- **a** (,) *[real,inout,allocatable]*
- **n1_new** *[integer,in]*
- **n2_new** *[integer,in]*

subroutine utility/**print_bar**(*prog*, *msg*[, *complete*])

Parameters

- **prog** *[real]*
- **msg** *[character,in]*

- **complete** *[logical]*

Called from
init_tree(), *filter_virial_masses()*, *read_tree()*, *asloth*

4.2.4 Virial radius

Quick access

Routines
rvir(), *rvir_mpc()*

Needed modules

- *numerical_parameters*

Subroutines and functions

`function` virial_radius/**rvir**(*mhalo*, *z*)

Parameters

- **mhalo** *[real,in]*
- **z** *[real,in]*

Return
r *[real]*

Called from
r_ion_ad(), *rvir_mpc()*

`function` virial_radius/**rvir_mpc**(*mhalo*, *z*)

Parameters

- **mhalo** *[real,in]* :: halo mass
- **z** *[real,in]* :: redshift

Return
r *[real]*

Called from
r_ion_ad(), *sf_step()*, *outputs_satellites()*, *write_files_reali()*

Call to
rvir()

4.2.5 Check flags

Description

Module to check whether the flags defined and undefined in asloth.h are an ok combination

Quick access

Routines
check_compiler_flags()

Subroutines and functions

subroutine check_flags/**check_compiler_flags**()

Called from
asloth

4.2.6 Random number generator

Description

This is an object oriented version of the ran3 algorithm from numerical recipes Before using a rng type object it needs to be initialized by calling rng%init(seed) where seed is an integer seed Afterwards rng%get_ran() will return a random real in the range [0,1)

Quick access

Types
rng

Variables
rng_seed

Routines
get_ran(), *init()*, *poisson()*

Types

- **type** random_object/**rng**

  Type fields

  - **% iff** *[integer]*
  - **% inext** *[integer]*
  - **% inextp** *[integer]*
  - **% ma** (55) *[integer]*
  - **% mj** *[integer]*

Variables

- random_object/**rng_seed** *[integer]*

Subroutines and functions

subroutine random_object/**init**(*this*, *idum*)

initializes the RNG

Parameters

- **this** *[real]*
- **idum** *[integer,in]*

function random_object/**get_ran**(*this*)

Retuns uniformly distributed random numbers in the range [0,1)

Parameters
this *[real]*

Return
get_ran *[real]*

Called from
split(), *statistical_feedback()*, *dz_external()*, *dz_internal()*, *poisson()*, *get_fracs()*

function random_object/**poisson**(*this*, *mu*)

Returns poisson distributed random numbers. Works only for small numbers (mu < 10)

Parameters

- **this** *[real]*
- **mu** *[real,in]* :: parameter of the poisson distribution

Return
poisson *[integer]*

Called from
imfsampling()

Call to
get_ran()

4.3 Metal modules

Modules and functions for the handling of metals

4.3.1 Metals

Description

main module for handling metals

Quick access

Types
element, *element_list*

Variables
index_carbon, *index_iron*, *n_elements*, *tracked_elements*

Routines
append(), *get_element_index()*, *get_element_info()*, *init_element()*

Needed modules

- *utility* (*assert()*): General purpose utility functions

Types

- **type** metals/**element**
  container type for individual elements
  Type fields
  - **% atomic_mass** *[real]*
  - **% element_name** *[character]*
  - **% solar_abund** *[real]*
- **type** metals/**element_list**
  this list is a container for all
  Type fields
  - **% list** (*) *[*element*,allocatable]*
  - **% n** *[integer,optional/default=0]*

Variables

- metals/**index_carbon** *[integer,optional/default=0]*
- metals/**index_iron** *[integer,optional/default=0]*
  index of carbon and iron in yields and metal arrays
- metals/**n_elements** *[integer]*
- metals/**tracked_elements** *[*element_list*]*

Subroutines and functions

subroutine metals/**append**(*this*, *el*)

appends an element to an element list

Parameters

- **this** *[real]* :: the list of elements
- **el** *[element]* :: the element to be appended

function metals/**get_element_index**(*this*, *el_name*)

returns the the index at which an element is found in the yield table

Parameters

- **this** *[real]* :: list of tracked elements
- **el_name** *[character,in]* :: name of the element to look up

Return
i *[integer]*

Called from
readelementid()

function metals/**init_element**(*el_name*)

Parameters
el_name *[character,in]*

Return
this *[element]*

Call to
get_element_info()

subroutine metals/**get_element_info**(*el_name*, *atomic_mass*, *solar_abund*)

Parameters

- **el_name** *[character,in]* :: which element are we looking up?
- **atomic_mass** *[real,out]* :: mean atomic weight
- **solar_abund** *[real,out]* :: converting from the log(X/H)+12 scale

Called from
init_element()

Call to
assert()

4.3.2 Read Yields

Description

module for reading yields from the appropirate files

Quick access

Routines
assign_yields(), *assign_yields_general()*, *get_elem_num()*, *readelementid()*

Needed modules

- *populations*: This module contains the type for storing IMFs, and the properties of the stellar populations
- *config* (*config_file()*)
- *metals*: main module for handling metals

Subroutines and functions

subroutine read_yields/**assign_yields**()

Called from
asloth

Call to
readelementid(), *assign_yields_general()*

subroutine read_yields/**readelementid**()

Called from
assign_yields()

Call to
get_element_index()

subroutine read_yields/**assign_yields_general**(*fileyields*, *pop*)

Parameters

- **fileyields** *[character,in]*
- **pop** *[*population*,in]* :: go through IMF bins

Called from
assign_yields()

Call to
get_elem_num()

subroutine read_yields/**get_elem_num**(*f*, *num_ele*, *mass_num*)

defines the number of elements and the number of different stellar masses in the yields file hard-coded file names. If you want to use a different table, add it here.

Parameters

- **f** *[character,in]* :: file name

- **num_ele** *[integer,out]* :: number of elements in the file
- **mass_num** *[integer,out]* :: number of masses in the table

Called from
assign_yields_general()

4.3.3 Metal functions

Description

Helper functions for handling metals and computing abundances and metallicities

Quick access

Routines
add_to_m_metals(), *cfe()*, *check_m_metals()*, *coh()*, *feh()*, *get_m_metals()*, *get_melement()*, *get_metallicity()*, *metalpoor()*, *zcal()*

Needed modules

- *numerical_parameters*
- *defined_types* (*treenode()*)
- *utility* (*assert()*): General purpose utility functions
- *metals*: main module for handling metals

Subroutines and functions

function metal_functions/**get_metallicity**(*m_h*, *mmetals*, *ii*)

gas metallicity of this halo This function assumed homogeneous mixing of metals with gas Because this homogeneous metallicity is needed as input to calculate dZ in order to corret to an inhomogeneous metallicity

Parameters

- **m_h** *[real,in]*
- **mmetals** *[real,in]* :: mass hydrogen, mass metals
- **ii** *[integer,in]* :: index in abundance array

Return
get_metallicity *[real]*

Called from
sf_step()

subroutine metal_functions/**check_m_metals**(*thishalo*)

if not yet allocated, allocate ThisHalo%m_metals and set to zero

Parameters
thishalo *[*treenode*]*

Called from
add_to_m_metals(), *prepare_node()*, *sf_step()*

subroutine metal_functions/**add_to_m_metals**(*thishalo*, *mmetal*, *iadd*)

add metals to %m_metals node property :p integer iadd [in]: is index of which metal type should be added

Parameters

- **thishalo** *[*treenode*]*
- **mmetal** (*n_elements*) *[real,in]*

Called from
hms_feedback()

Call to
check_m_metals()

function metal_functions/**get_m_metals**(*thishalo*, *iget*)

get metal mass array from %m_metals node property :p integer iget [in]: is index of which metal type is wanted

Parameters
thishalo *[*treenode*]*

Return
get_m_metals (*n_elements*) *[real]* :: function returns an array

Called from
get_m_metals(), *add_parent_to_tree()*

Call to
get_m_metals()

function metal_functions/**get_melement**(*thishalo*, *index_metal*)

get mass of this halo of one specific element

Parameters

- **thishalo** *[*treenode*]*
- **index_metal** *[integer,in]*

Return
get_melement *[real]*

Called from
feh(), *coh()*, *cfe()*

function metal_functions/**feh**(*thishalo*, *dz_now*)

gas metallicity of this halo [Fe/H]

Parameters

- **thishalo** *[*treenode*]*
- **dz_now** *[real,in]*

Return
feh *[real]*

Called from
metalpoor()

Call to
get_melement()

function metal_functions/**coh**(*thishalo*, *dz_now*)

gas metallicity of this halo [C/H]

Parameters

- **thishalo** *[*treenode*]*
- **dz_now** *[real,in]*

Return
coh *[real]*

Called from
metalpoor()

Call to
get_melement()

function metal_functions/**cfe**(*thishalo*)

gas metallicity of this halo [C/Fe]

Parameters
thishalo *[*treenode*]*

Return
cfe *[real]*

Call to
get_melement()

function metal_functions/**metalpoor**(*thishalo*, *z_in*, *dz_now*)

Parameters

- **thishalo** *[*treenode*]*
- **z_in** *[real,in]*
- **dz_now** *[real,in]*

Return
metalpoor *[logical]*

Called from
sf_step()

Call to
feh(), *assert()*, *coh()*

subroutine metal_functions/**zcal**(*metallicity*, *gas*, *metals*)

20200820 Li-Hsin This routine also calculates the metallciity but in linear scale. This is simply for the convenience when we want to keep information from the last time step.

Parameters

- **metallicity** *[real,out]*
- **gas** *[real,in]*
- **metals** *[real,in]*

4.3.4 IGM Metallicity routines

Description

utility routine for inheriting the IGM metallicity accross time

Quick access

Routines
inherit_igm_z()

Needed modules

- *defined_types*

Subroutines and functions

subroutine igm_z/**inherit_igm_z**(*p_node*)

Parameters
p_node *[*treenode,*inout]* :: parent node

Called from
prepare_node()

4.3.5 Metal Mix dZ

Quick access

Variables
dztable_12, *dztable_23*, *dztable_34*, *dztable_45*, *dztable_58*

Routines
calc_dz(), *dz_external()*, *dz_internal()*, *read_lookups()*

Needed modules

- *numerical_parameters*
- *defined_types*
- *random_object*: This is an object oriented version of the ran3 algorithm from numerical recipes Before using a rng type object it needs to be initialized by calling rng%init(seed) where seed is an integer seed Afterwards rng%get_ran() will return a random real in the range [0,1)

Variables

- metalmix_dz/**dztable_12** (1000) *[real,protected]*
- metalmix_dz/**dztable_23** (1000) *[real,protected]*
- metalmix_dz/**dztable_34** (1000) *[real,protected]*
- metalmix_dz/**dztable_45** (1000) *[real,protected]*
- metalmix_dz/**dztable_58** (1000) *[real,protected]*

Subroutines and functions

subroutine metalmix_dz/**read_lookups**()

Called from
init_tree()

function metalmix_dz/**dz_external**(*rng_obj*, *z*)

dilution from Yuta Tarumi: ApJ, Volume 897, Issue 1, id.58 based on Renaissance Simulation dZ = (Mmetals/Mgas)_densestGas / (Mmetals/Mgas)_halo hydrogen mass in this halo / dZ = hydrogen mass with which metals mix in densest region

Parameters

- **rng_obj** *[*rng*,inout]*
- **z** *[real,in]*

Return
dz_external *[real]*

Called from
calc_dz()

Call to
get_ran()

function metalmix_dz/**dz_internal**(*rng_obj*)

Parameters
rng_obj *[*rng*,inout]*

Return
dz_internal *[real]* :: to be consistent with previous use of dZ

Called from
calc_dz()

Call to
get_ran()

function metalmix_dz/**calc_dz**(*m_metals_int*, *rng_obj*, *z_now*)

metal mixing correction based on Tarumi+20, ApJ, Volume 897, Issue 1, id.58 dZ: log-differenze between metallicity of all gas in a halo and the dense gas

Parameters

- **m_metals_int** *[real,in]*
- **rng_obj** *[*rng*,inout]*

- **z_now** *[real,in]*

Return
calc_dz *[real]*

Called from
sf_step()

Call to
dz_external(), *dz_internal()*

4.3.6 Metallicity distribution function

Quick access

Variables
mdf_zarray, *mdfdz*, *mdfnbin*, *mdfnpad*, *mdfzmax*, *mdfzmin*

Routines
add_to_base_mdf(), *add_to_node_mdf()*, *get_mdf_bin()*, *init_mdf()*

Needed modules

- *defined_types*

Variables

- trace_mdf/**mdf_zarray** (*) *[real,allocatable/protected]*
- trace_mdf/**mdfdz** *[real,parameter*=0.1*]*
  MDF range, bin width
- trace_mdf/**mdfnbin** *[integer]*
- trace_mdf/**mdfnpad** *[integer,parameter*=2*]*
- trace_mdf/**mdfzmax** *[real,parameter*=1*]*
- trace_mdf/**mdfzmin** *[real,parameter*=-9*]*

Subroutines and functions

subroutine trace_mdf/**init_mdf**()

Called from
init_tree()

Call to
get_mdf_bin()

function trace_mdf/**get_mdf_bin**(*zgas*)

Parameters
zgas *[real]*

Return
get_mdf_bin *[integer]* :: [Fe/H]

Called from
imfsampling(), *init_mdf()*

subroutine trace_mdf/**add_to_node_mdf**(*this_node*, *mdf_nstar*, *mdf_index*, *idx2*)

add MDF_Nstar stars to the MDF of This_Node

Parameters

- **this_node** *[*treenode*,inout]*
- **mdf_nstar** *[real,in]*
- **mdf_index** *[integer,in]*
- **idx2** *[integer,in]*

Called from
imfsampling(), *outputs_satellites()*, *write_files_reali()*

subroutine trace_mdf/**add_to_base_mdf**(*this_node*)

Inherit MDF to the base node at the final redshift

Parameters
this_node *[*treenode*,inout]*

Called from
asloth

4.4 Spatially resolved feedback

Modules and routines for efficiently tracing spatially resolved feedback

4.4.1 Bubble Tree

Description

This module containes routines for adding feedback of haloes to the tracing system and for testing whether a halo or a specific point in space is affected by the feedback

Quick access

Variables
head_nodes, *head_nodes_ion*, *l_box_max*, *mass_diff*, *max_level*

Routines
add_bubble_to_tree(), *bubble_check()*, *clean_bnode()*, *clean_tree()*, *copy_props_to_bubble()*, *count_ion()*, *dist_check()*, *dist_check_allow_same()*, *find_bnode()*, *grid_check()*, *ion_check()*, *merge_into_bubble()*, *mv_bubble()*, *position_check()*, *rm_bubble()*, *start_btree()*

Needed modules

- *numerical_parameters*
- *defined_types*
- *config* (*cubic_box()*)
- *cosmic_time* (*alev()*, *a3_inv()*)
- *crit_masses* (*atomic_cooling_mass()*)
- *metal_functions*: Helper functions for handling metals and computing abundances and metallicities
- *bubble_tree_types*

Variables

- bubble_tree/**head_nodes** (*nlev_max*) *[*n_pointer*]*
- bubble_tree/**head_nodes_ion** (*nlev_max*) *[*n_pointer*]*
- bubble_tree/**l_box_max** *[real]*
- bubble_tree/**mass_diff** *[real]*
- bubble_tree/**max_level** *[integer,parameter*=20*]*

Subroutines and functions

subroutine bubble_tree/**start_btree**()

initializes tree and sets up the head nodes

Called from
asloth

subroutine bubble_tree/**add_bubble_to_tree**(*nodeadd*, *mmetal_add*)

Adds ionized an enriched bubble of a node to the tree

Parameters

- **nodeadd** *[*treenode*,inout]*
- **mmetal_add** (*n_elements*) *[real,in]* :: metal mass in this SN shell

Called from
add_parent_to_tree()

Call to
find_bnode(), *increase_size()*, *copy_props_to_bubble()*

subroutine bubble_tree/**find_bnode**(*jlevel*, *x*, *bnode*, *r*[, *ion_tree*])

Parameters

- **jlevel** *[integer,in]*
- **x** (3) *[real,in]*
- **bnode** *[*bubble_node*,out,pointer]* :: associate parent node
- **r** *[real,in]*

- **ion_tree** *[logical,in,]*

Called from
add_bubble_to_tree()

Call to
grid_check()

subroutine bubble_tree/**rm_bubble**(*node*, *to_remove*)

Parameters

- **node** *[*bubble_node*,pointer]*
- **to_remove** *[integer]*

Call to
mv_bubble()

subroutine bubble_tree/**mv_bubble**(*n_from*, *i_from*, *n_to*, *i_to*)

moves the bubble to_move to the node move_to and fixes all the pointers afterwards

Parameters

- **n_from** *[*bubble_node*,inout,pointer]*
- **i_from** *[integer,in]*
- **n_to** *[*bubble_node*,inout,pointer]*
- **i_to** *[integer,in]*

Called from
rm_bubble()

subroutine bubble_tree/**copy_props_to_bubble**(*bnode*, *bubble*, *nodeadd*, *mmetal_add*)

Parameters

- **bnode** *[*bubble_node*,pointer]*
- **bubble** *[integer,in]* :: Index of Bubble
- **nodeadd** *[*treenode*,inout]*
- **mmetal_add** (*) *[real,in]* :: metal mass in this SN shell

Called from
add_bubble_to_tree()

subroutine bubble_tree/**bubble_check**(*this_node*)

routine to check whether This_Node is externally enricher and/or ionized

Parameters
this_node *[*treenode*,inout,target]*

Called from
node_feedback()

Call to
dist_check_allow_same(), *grid_check()*

subroutine bubble_tree/**ion_check**(*this_node*)

routine to check whether This_Node is ionized the treatment of haloes above the atomic cooling limit is based on Visbal et al. 2016 (MNRAS 460, L59-L63)

Parameters
this_node *[*treenode*,inout,target]*

Called from
node_feedback()

Call to
dist_check(), *grid_check()*

subroutine bubble_tree/**clean_tree**(*jlevel*)

Parameters
jlevel *[integer,in]*

Called from
asloth

Call to
clean_bnode()

subroutine bubble_tree/**clean_bnode**(*jlevel*, *bnode*, *n_merge*)

Parameters

- **jlevel** *[integer,in]*
- **bnode** *[*bubble_node*,inout,pointer]*
- **n_merge** *[integer,inout]*

Called from
clean_tree()

subroutine bubble_tree/**merge_into_bubble**(*jlevel*, *x_in*, *v_in*, *l_in*, *x_out*, *v_ion_current*, *l_out*, *bnode*, *bubble_in*)

Parameters

- **jlevel** *[integer,in]*
- **x_in** (3) *[real,in]*
- **v_in** *[real,in]*
- **l_in** *[real,in]*
- **x_out** (3) *[real,out]*
- **v_ion_current** *[real,out]*
- **l_out** *[real,out]*
- **bnode** *[*bubble_node*,pointer]*
- **bubble_in** *[integer,in]*

subroutine bubble_tree/**position_check**(*x*, *j_now*, *is_ion*, *is_enr*)

routine to check whether This_Node is externally enricher and/or ionized

Parameters

- **x** (3) *[real,in]*
- **j_now** *[integer,in]*
- **is_ion** *[logical,out]*

- **is_enr** *[logical,out]*

Called from
get_fracs()

Call to
grid_check()

function bubble_tree/**count_ion**(*x*, *j_now*, *ion_tree*)

routine to check whether This_Node is externally enricher and/or ionized

Parameters

- **x** (3) *[real,in]*
- **j_now** *[integer,in]*
- **ion_tree** *[logical,in]*

Return
n_ion *[integer]*

Call to
grid_check()

subroutine bubble_tree/**grid_check**(*ijk*, *bnode*, *x_in*, *caller*)

Parameters

- **ijk** (3) *[integer,in]*
- **bnode** *[*bubble_node*,pointer]*
- **x_in** (3) *[real,in]*
- **caller** *[character,in]*

Called from
find_bnode(), *bubble_check()*, *ion_check()*, *position_check()*, *count_ion()*

function bubble_tree/**dist_check_allow_same**(*x_ref*, *x*, *r*)

Parameters

- **x_ref** (3) *[real,in]*
- **x** (3) *[real,in]*
- **r** *[real,in]*

Return
dist_check_allow_same *[logical,pure]*

Called from
bubble_check()

function bubble_tree/**dist_check**(*x_ref*, *x*, *r*)

Parameters

- **x_ref** (3) *[real,in]*
- **x** (3) *[real,in]*
- **r** *[real,in]*

Return
dist_check *[logical,pure]*

Called from
ion_check()

4.4.2 Bubble tree types

Quick access

Types
bubble_node, *n_pointer*

Routines
increase_size(), *resize()*

Needed modules

- *utility* (realloc()): General purpose utility functions
- *metals* (*n_elements()*): main module for handling metals

Types

- **type** bubble_tree_types/**n_pointer**

  Type fields

  - **% p** *[*bubble_node,*pointer/optional/default=>null()]*

- **type** bubble_tree_types/**bubble_node**

  Type fields

  - **% l_box** *[real]* :: size of the node
  - **% l_ion** (*) *[real,allocatable]* :: ionizing photon emission rate
  - **% level** *[integer]* :: tree-level where the node is located at
  - **% m_out** (*) *[real,allocatable]* :: metal mass
  - **% mmetals_bub** (*) *[real,allocatable]* :: metal mass
  - **% n_bub** *[integer,optional/default=0]*
  - **% n_max** *[integer,optional/default=0]*
  - **% null** *[*bubble_node,*pointer]*
  - **% parent_node** *[*bubble_node,*pointer/optional/default=>]*
  - **% pop** (*) *[integer,allocatable]* :: enriching population
  - **% r_en** (*) *[real,allocatable]* :: enriched radius
  - **% r_ion** (*) *[real,allocatable]* :: ionized radius
  - **% sub_nodes** (8) *[*n_pointer*]* :: pointers to the eight sub-nodes
  - **% x** (,) *[real,allocatable]* :: position
  - **% x_node** (3) *[real]* :: position of the corner of the node
  - **% yields_bub** (,) *[real,allocatable]*

Subroutines and functions

subroutine bubble_tree_types/**resize**(*this*)

resizes bubble storage for one node within the bubble tree

Parameters
this *[real]*

subroutine bubble_tree_types/**increase_size**(*this*)

Makes room for one additional bubble in a node of the bubble-tree

Parameters
this *[real]*

Called from
add_bubble_to_tree()

4.5 Stellar populations

4.5.1 Populations

Description

This module contains the type for storing IMFs, and the properties of the stellar populations

Quick access

Types
population

Variables
popii, *popiii*

Routines
compute_ionizing_mass_rate(), *create_pop()*, *get_index_of_mass_cutoff()*, *init_stellar_populations()*, *massprobabilityimf()*, *output_pop()*, *set_yields()*

Needed modules

- *numerical_parameters*
- *resolution_parameters*
- *stellar_props*
- *utility*: General purpose utility functions
- *imfs*: This module contains intial mass functions
- *metals*: main module for handling metals

Types

- **type** populations/**population**

  Type fields

  - **% f_esc** *[real]* :: ionizing photon escape fraction
  - **% ion_m_rate** (*) *[real,allocatable]* :: mass heating rate (M_sun/s) due to ionizing radiation
  - **% mass** (*) *[real,allocatable]* :: stellar mass in the bin
  - **% n_bins** *[integer]* :: number of bins
  - **% n_cutoff** *[integer]* :: Index in IMF array at which "massive" stars begin
  - **% n_ion** *[real]* :: Number of ionizing photons per stellar baryon
  - **% p** (*) *[real,allocatable]* :: relative likelihood to form star in the bin
  - **% pop** *[integer]* :: what stellar population is this?
  - **% q** (*) *[real,allocatable]* :: ionizing photon emission rate
  - **% sn_energy** (*) *[real,allocatable]* :: Energy of a SN in this bin (erg)
  - **% sn_momentum** (*) *[real,allocatable]* :: mass blown out SN (Msun cm/s)
  - **% tlife** (*) *[real,allocatable]* :: lifetime
  - **% yields** (,) *[real,allocatable]* :: mass blown out SN (Msun cm/s)

Variables

- populations/**popii** *[*population*,protected]*
- populations/**popiii** *[*population*,protected]*

Subroutines and functions

function populations/**create_pop**(*n*, *pop*)

Parameters

- **n** *[integer,in]*
- **pop** *[integer,in]*

Return
this *[*population*]*

Call to
massprobabilityimf(), *get_index_of_mass_cutoff()*, *q_popii()*, *lifetime_popii()*, *e_sn_popii()*, *momentum_sn()*, *q_popiii()*, *lifetime_popiii()*, *e_sn_popiii()*

subroutine populations/**init_stellar_populations**()

Called from
asloth

function populations/**get_index_of_mass_cutoff**(*nstarmassbin*, *mstarpop*)

This function determins the index where the desired mass is at in the mass bins.

Options
nstarmassbin *[integer,in,optional/default=len(mstarpop)]*

Parameters
mstarpop (nstarmassbin) *[real,in]*

Return
n_cutoff *[integer]*

Called from
create_pop()

subroutine populations/**massprobabilityimf**(*p_starmassbin*, *mstarpop*, *pop*, *n_starmassbin*)

This routine determins the mass probability of a mass bin. We determine the total stellar mass form in one step first and then draw individual stars from the IMF.

Parameters

- **p_starmassbin** (n_starmassbin) *[real,out]*
- **mstarpop** (n_starmassbin) *[real,out]*
- **pop** *[integer,in]* :: PopII (2) or PopIII (3)?
- **n_starmassbin** *[integer,in]*

Called from
create_pop()

Call to
assert()

subroutine populations/**compute_ionizing_mass_rate**(*this*)

Gas mass heating/ionizing rate from massive stars throughout their lifetimes. Gas density dependent.

Parameters
this *[real]*

subroutine populations/**set_yields**(*this*[, *yields*])

Parameters

- **this** *[real]*
- **yields** (,) *[real,in,allocatable]*

subroutine populations/**output_pop**(*pop*, *fname*)

Parameters

- **pop** *[real]*
- **fname** *[character,in]*

Called from
asloth

4.5.2 Initial mass functions

Description

This module contains intial mass functions

Quick access

Types
initial_mass_function, *unknown_type*

Routines
kroupa_cumulative_mass_df(), *kroupa_cumulative_pdf()*, *kroupa_m_func()*, *kroupa_p_func()*, *kroupa_pdf()*, *m_func_dummy()*, *p_func_dummy()*, *power_law_mass_func()*, *power_law_p_func()*

Needed modules

- *utility* (*assert()*): General purpose utility functions

Types

- **type** imfs/**initial_mass_function**

  This type acts as a framework to implement additional IMFs. It cannot directly be used Rather, the idea is to create new types and inherit the basic structure from this one Note: This should be implemented as abstract type, but f2py.crackfortrant does not support these yet, which would break the documentation with sphinxfortran

- **type** imfs/**unknown_type**

  Type fields

  - **% m_max** *[real]*
  - **% m_min** *[real]*
  - **% slope** *[real]*

- **type** imfs/**unknown_type**

  Type fields

  - **% m_max** *[real]*
  - **% m_min** *[real]*
  - **% m_turn1** *[real,optional/default=0.08]*
  - **% m_turn2** *[real,optional/default=0.5]*
  - **% slope1** *[real,optional/default=-0.3]*
  - **% slope2** *[real,optional/default=-1.3]*
  - **% slope3** *[real,optional/default=-2.3]*

Subroutines and functions

function imfs/**m_func_dummy**(*this*, *m1*, *m2*)

Parameters

- **this** *[real]*
- **m1** *[real,in]*
- **m2** *[real,in]*

Return
res *[real]*

function imfs/**p_func_dummy**(*this*, *m1*, *m2*)

Parameters

- **this** *[real]*
- **m1** *[real,in]*
- **m2** *[real,in]*

Return
res *[real]*

function imfs/**power_law_mass_func**(*this*, *m1*, *m2*)

Parameters

- **this** *[real]*
- **m1** *[real,in]*
- **m2** *[real,in]*

Return
res *[real]*

function imfs/**power_law_p_func**(*this*, *m1*, *m2*)

Parameters

- **this** *[real]*
- **m1** *[real,in]*
- **m2** *[real,in]*

Return
res *[real]*

function imfs/**kroupa_p_func**(*this*, *m1*, *m2*)

Parameters

- **this** *[real]*
- **m1** *[real,in]*
- **m2** *[real,in]*

Return
res *[real]*

Call to
assert(), *kroupa_cumulative_mass_df()*

function imfs/**kroupa_m_func**(*this*, *m1*, *m2*)

Parameters

- **this** *[real]*
- **m1** *[real,in]*
- **m2** *[real,in]*

Return
res *[real]*

Call to
assert(), *kroupa_cumulative_mass_df()*, *kroupa_cumulative_pdf()*

function imfs/**kroupa_cumulative_mass_df**(*this*, *m*)

Parameters

- **this** *[real]*
- **m** *[real,in]*

Return
res *[real]*

Called from
kroupa_p_func(), *kroupa_m_func()*

function imfs/**kroupa_pdf**(*this*, *m*)

Parameters

- **this** *[real]*
- **m** *[real,in]*

Return
res *[real]*

function imfs/**kroupa_cumulative_pdf**(*this*, *m*)

Parameters

- **this** *[real]*
- **m** *[real,in]*

Return
res *[real]*

Called from
kroupa_m_func()

4.5.3 Stellar Properties

Quick access

Routines
e_sn_popii(), *e_sn_popiii()*, *lifetime_popii()*, *lifetime_popiii()*, *momentum_sn()*, *q_popii()*, *q_popiii()*

Needed modules

- *numerical_parameters*
- *resolution_parameters*

Subroutines and functions

function stellar_props/**q_popiii**(*m*)

Parameters
m *[real,in]*

Return
q *[real]*

Called from
create_pop()

function stellar_props/**lifetime_popiii**(*m*)

Parameters
m *[real,in]*

Return
lt *[real]*

Called from
create_pop()

function stellar_props/**q_popii**(*m*)

Parameters
m *[real,in]*

Return
q *[real]*

Called from
create_pop()

function stellar_props/**lifetime_popii**(*m*)

Parameters
m *[real,in]*

Return
lt *[real]*

Called from
create_pop()

function stellar_props/**e_sn_popii**(*m*)

Parameters
m *[real,in]*

Return
e_sn *[real]*

Called from
create_pop()

function stellar_props/**e_sn_popiii**(*m*)

Parameters
m *[real,in]*

Return
e_sn *[real]*

Called from
create_pop()

function stellar_props/**momentum_sn**(*e_sn*)

Parameters
e_sn *[real,in]*

Return
p_sn *[real]*

Called from
create_pop()

4.6 Python Scripts

4.6.1 scripts/plot_asloth.py

scripts.plot_asloth.**plot_comparison**(*UserFolder*, *UserLabels=None*)

Function to plot results from A-SLOTH and compare them to literature or other runs.

We use the most recent folder to plot and compare to literature. If one or more folders are specified, they will all be plotted. This can also be used to plot just one specific folder.

Parameters

- **UserFolder** (*list[str]*) – List of folder names to be plotted.
- **UserLabels** (*list[str]*) – List of labels. If none, UserFolders are used as labels

Returns: Beautiful plots

scripts.plot_asloth.**readMWproperties**(*UserFolder*)

4.6.2 scripts/tau/plot_Vion.py

tau.plot_Vion.**plot_Vion**(*UserFolder*, *UserLabels*, *folder_output*)

Function to plot ionisation histories and calculate tau

Parameters

- **UserFolder** (*List[str]*) – List of folder names to be plotted
- **UserLabels** (*List[str]*) – List of labels. If none, UserFolders are used as labels
- **folder_output** (*str*) – Folder in which plots will be saved

Returns

- **tau0** (*float*) – optical depth to Thomson scattering
- **tau_sigma** (*float*) – tau uncertainty

tau.plot_Vion.**get_tau_sigma**(*folder*)

Function to calculate tau without plotting ionisation history

Parameters
folder (*str*) – Folder in which plots will be saved

Returns

- **tau0** (*float*) – optical depth to Thomson scattering
- **tau_sigma** (*float*) – tau uncertainty

4.6.3 scripts/wrapper

This script allows to explore input parameters by automatically launching various runs. Users can set their specific path_output variable. If you want to run with NBODY merger trees, also set path_tree. If you think that disk I/O might be a bottleneck, you should also set your username in the function N_disk_sleep() to work correctly.

wrapper.loop_trees.**RunTrees**(*cats*, *NBODY*)

This function loops over a set of merger trees and explores one combination of input parameters

We first define input parameters that are constant within this loop. Then, we define tree-specific input parameters. Then, we check if sufficient computing resources are available. Eventually, we launch the A-SLOTH run.

Parameters

- **cats** (*list*) – catalogue of merger trees [[name, nlev, RAM]]
- **NBODY** (*bool*) – N-Body or EPS-generated merger trees

wrapper.loop_trees.**Get_CatList**(*NBODY=False*, *N=3*)

Function to generate list of catalogues to loop over

Parameters

- **NBODY** (*bool*) – N-Body or EPS-generated merger trees
- **N** (*int*) – number of random realisations in case of EPS-generated trees

Returns
cats – list of lists of type [[name, nlev, RAM]]

Return type
list

wrapper.loop_trees.**avail_mem_GB**()

Funtion returns available RAM in GB

wrapper.loop_trees.**CPU_use**()

Funtion returns available CPUs in %

wrapper.loop_trees.**N_disk_sleep**()

Funtion returns number of processes that are idle due to I/O.

Change username accordingly

wrapper.analyse_trees.**get_p_from_folders**(*dir_prefix*, *ifEPS=True*)

Function that calculates the p-value (and additional values).

Based on the directory prefix for the A-SLOTH output folder that should be analysed Function will you all folders that match this directory prefix.

Parameters
dir_prefix (*str*) – directory prefix that should be used to search for data

Returns
fit_asloth – class object that contains fit results

Return type
asloth_fit

wrapper.tutorial_GridPlot.**plot_tutorial2**()

Plotting script to visualise results of the two scripts loop_scripts.py and analysis_trees.py.

The plot layout is not optimized for aesthetics, but it should simply demonstrate the versatility of A-SLOTH. The resulting plot is a modified version of Fig. 17 in Hartwig+22

4.6.4 scripts/tau/plot_tgas.py

utilities.plot_tgas.**plot_tgas**(*output_folder*)

Illustrates time evolution of different baryonic quantities based on output file t_gas_substeps.dat. outout_folder has to be set manually.

Parameters
output_folder (*str*) – folder that contains results of run that should be analysed

4.6.5 scripts/utility.py

class utilities.utility.**asloth_config**

container class for writing config namelist files for ASLOTH

set_config_to_default()

sets default parameters for the main configuration

Parameters
self (asloth_config) – the configuration to be restored to defaults

set_metals(*metals*)

sets metal configuration for a specified list of elements

Parameters

- **self** (asloth_config) – configuration

- **metals** (*list*) – list of strings of the elements to be traced

to_file(*fname*)

class utilities.utility.**asloth_fit**

class for handling fitting parameters and results of A-SLOTH

get_p()

‘ Calculates the goodness-of-fit parameters based on 6 different observables. Please see Hartwig+22 for more details.

print_fit()

‘ Prints all class properties

set_fit_to_default()

sets default parameters for the fit object

Parameters

self (asloth_fit) – set all parameters to default values

utilities.utility.**get_p_from_sigma**(*sigma*)

Get p-value from “how many sigma away”

utilities.utility.**write_namelist**(*f*, *list_name*, *var_dict*)

function to write fortran namelists

Parameters

- **f** (*file*) – opened file to write the namelist to
- **list_name** (*str*) – name of the namelist. Note that this must be the exact name used in the fortran code
- **var_dict** (*dictionary*) – dictionary of variables to be written to the file

4.6.6 scripts/SMHM/plot_scatterSMHM.py

SMHM.plot_scatterSMHM.**plot_scatterSMHM**(*UserFolder*, *UserLabels*, *folder_output*)

Function to read data and plot scatter SMHM relation.

If one or more folders are specified, they will all be plotted. This can also be used to plot just one specific folder.

Parameters

- **UserFolder** (*str*) – List of folder names to be plotted.
- **UserLabels** (*str*) – List of labels. If none, UserFolders are used as labels.
- **folder_output** (*str*) – where the plot is saved.

Returns

None

SMHM.plot_scatterSMHM.**plot_CumuSMF**(*UserFolder*, *UserLabels*, *folder_output*, *if_plot*)

Function to return ks statistics, p-values, and MW stellar masses

If one or more folders are specified, they will all be plotted. This can also be used to plot just one specific folder.

Parameters

- **UserFolder** (*str*) – List of folder names to be plotted.

- **UserLabels** (*str*) – List of labels. If none, UserFolders are used as labels.
- **folder_output** (*str*) – where the plot is saved.

Returns
Lists of ks statistic, p-value and MW stellar mass.

SMHM.plot_scatterSMHM.**read_data**(*UserFolder*, *UserLabels*, *folder_output*)

Function to read data to plot SMHM relation

If one or more folders are specified, they will all be plotted. This can also be used to plot just one specific folder.

Parameters

- **UserFolder** (*str*) – List of folder names to be plotted.
- **UserLabels** (*str*) – List of labels. If none, UserFolders are used as labels
- **folder_output** (*str*) – where the plot is saved

Returns
Virial masses and stellar masses of galaxies in pandas.DataFrame format

SMHM.plot_scatterSMHM.**M_AM**(*M*)

Function to compute expected stellar mass of the galaxy at given virial mas (Garrison-Kimmel et al. 2014).

Parameters
M (*float*) – virial mass of the halo

Returns
expected stellar mass

SMHM.plot_scatterSMHM.**f_bwc**(*x*)

SMHM.plot_scatterSMHM.**AMtwodhistogram**(*df_list*, *y0*, *p_dir*, *output_name*, *WhichMass*, *climmax*, *if_imfana*)

The actual function to plot scatter SMHM relation.

Parameters

- **df_list** (*list of pd.DataFrame*) – input data
- **y0** (*float*) – the observational completeness
- **p_dir** (*str*) – output folder

Returns:

SMHM.plot_scatterSMHM.**CumulativeNumberOfSatellites**(*df_list*, *output_name*, *WhichMass*, *if_plot*, *p_dir*)

The actual function to plot cumulative stellar mass function and compute ks statistic, p-value from the KS test.

Parameters

- **df_list** (*list of pd.DataFrame*) – input data
- **output_name** (*str*) – figure name
- **WhichMass** (*float*) – which stellar mass to use (the total or survival)
- **if_plot** (*bool*) – whether to make plots or not
- **p_dir** (*str*) – output folder

Returns
lists of ks statistic, p-value, MW stellar mass

4.6.7 scripts/SMHM/plot_binnedSMHM.py

SMHM.plot_binnedSMHM.**plot_binnedSMHM**(*UserFolder*, *UserLabels*, *folder_output*)

The function reads data from multiple folders and plot the binned SMHM relations

If one or more folders are specified, they will all be plotted as separate SMHM relations. This can also be used to plot just one specific folder.

Parameters

- **UserFolder** (*str*) – List of folder names to be plotted.
- **UserLabels** (*str*) – List of labels. If none, UserFolders are used as labels
- **folder_output** (*str*) – where the plot is saved

Returns

plot of SMHM relation comparison

SMHM.plot_binnedSMHM.**IMFANA**(*AMBound*, *MX*, *MAMX*, *imf_list*, *y0*, *p_dir*, *imfname*, *WhichMass*, *climmax*, *cps*, *cs*, *ilabel*, *fname*)

The function plots binned SMHM relation.

Parameters

- **AMBound** – boundary of the SMHM relation from abundance matching (AM)
- **MX** – virial masses
- **MAMX** – stellar masses obtained from the AM technique at given virial masses
- **imf_list** (*list of pd.DataFrame*) – input data
- **y0** (*float*) – observational completeness
- **p_dir** (*str*) – output folder

Returns

plot of SMHM relation

SMHM.plot_binnedSMHM.**AM_BinMeanstd**(*df_list*, *WhichMass*, *tname*, *index_data*, *c=None*)

The actual function to plot binned SMHM relation.

Parameters

- **df_list** (*list of pd.DataFrame*) – input data
- **WhichMass** (*float*) – which stellar mass to use (the total or survival)
- **tnmae** (*str*) – labels
- **index-data** – index of the data
- **c** – specific color for the SMHM relation

Returns

adding a binned SMHM relation to the plot

SMHM.plot_binnedSMHM.**PlotNadler20SMHM**()

The function plots SMHM relation from Nadler et al. 2020.

4.6.8 scripts/plot_mcrit.py

utilities.plot_mcrit.**plot_mcrit**()

Function to plot ciritical mass of a halo at certain redshift, given by different models, and at different initial streaming velocities.

Returns
Figure that shows M_crit v.s. z at different initial streaming velocities and from different models

utilities.plot_mcrit.**crit_mass**(*z*, *lw_mode*, *VBC*)

Function to obtain ciritical mass of a halo at certain redshift, given by different models

We use lw_mode=5 (Schauer et al. 2021) as the fiducial model. Use can also choose lw_mode=4, which is a combined model from O'Shea et al. 2008, Stacy et al. 2011, and Hummel et al. 2012. lw_mode=7 corresponds to a model from Fialkov et al. 2013.

Parameters

- **z** (*float*) – redshift
- **lw_mode** (*int*) – which M_crit model
- **VBC** (*float*) – initial streaming velocity, in units of sigma_v

Returns
Critical mass of a halo to have star formation

4.6.9 scripts/plot_stellarprop.py

utilities.plot_stellarprop.**plot_stellarproperties**()

Function to reproduce the stellar properties plot

Returns
Stellar masses v.s. stellar properties, e.g. stellar lifetime, carbon yields, iron yields, supernova energie.

CHAPTER

FIVE

USAGE POLICY

You can use A-SLOTH freely for scientific research and other applications in accordance with the license. If you publish results for which you have used A-SLOTH, you must cite the following two papers:

- Hartwig et al. 2022, submitted to ApJ
- Magg et al. 2022, submitted to JOSS

You do not have to invite the A-SLOTH developers as co-authors on your publications. If you wish to contribute to A-SLOTH, please get in contact with us and/or create a pull-request.

CHAPTER

SIX

HELP

If you encounter any problems with A-SLOTH, there are several options to help you:

- Check the section on *debugging*.
- Create an issue on GitLab.
- Contact the developers: hartwigATu-tokyo.ac.jp

PYTHON MODULE INDEX

FORTRAN MODULE INDEX

INDEX

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z