DiscRete sEmi-empiricAl Model

DREAM Documentation

Hao Fu (h.fu@soton.ac.uk)
Chris Marsden (c.marsden@soton.ac.uk)
Francesco Shankar (f.shankar@soton.ac.uk)
Max Dickson (md2g17@soton.ac.uk)

Contents

1 Installation guide

1.1 Prerequisites

First of all, you need a clear installation of Python (version 3.0 or higher is recommended, though it works also with 2.0) with at least the `numpy`, `scipy`, `colossus` and `ctypes` module. This last one is to call some functions in C to speed up the code.

If you want to analyse the output data and produce the relative plots, you will also need to have the `matplotlib` module installed on your machine.

To test the presence of the aforementioned modules, you can type on the terminal:

```
$ python
$ >>> import numpy
$ >>> import scipy
$ >>> import colossus
$ >>> import ctypes
$ >>> import matplotlib
```

If one of these imports fails, go to the corresponding website, and follow the installation instructions there. If you have the `pip` installer on your machine, the command `pip install numpy` will do the job, as well as for the other packages.

A GNU C compiler, for example `gcc`, is also essential for compiling the code. For instructions on how to install the compiler visit https://gcc.gnu.org/install/. To test that it is correctly installed on the machine type on your terminal one of the following options:

```
$ gcc --version

$ gcc -v
```

1.2 Installation

Move release of DREAM to one of your folders, called e.g. `my-folder`, go to this directory on your terminal and type:

```
my-folder]$ tar -xvf DREAM.zip
my-folder]$ cd DREAM
```

The main code does not need any compilation. However, you need to compile the C-scripts. For this purpose, enter one of the following two option:

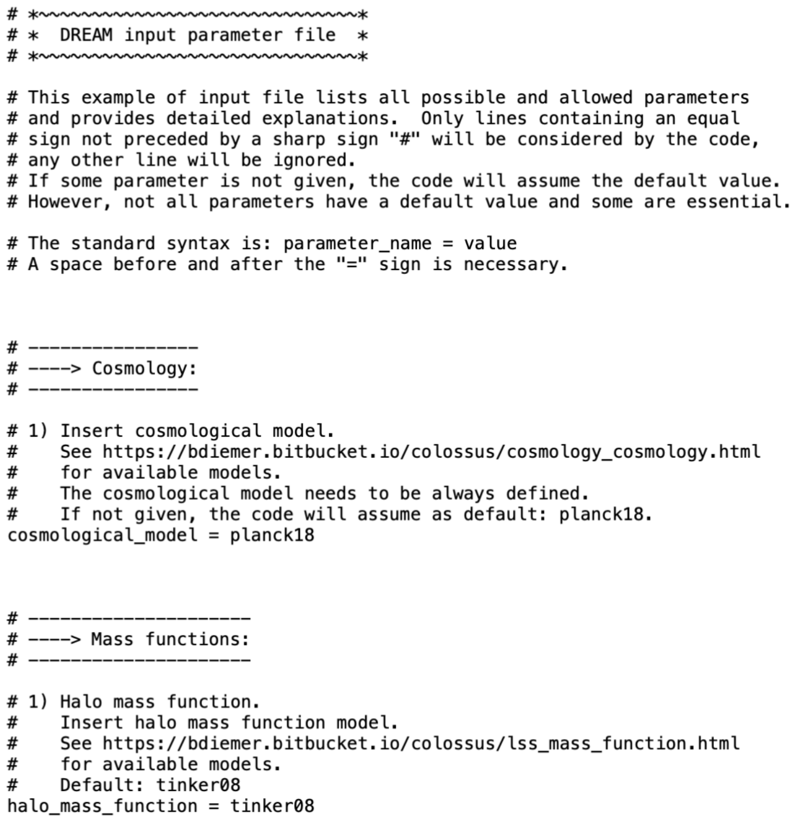
```
DREAM]$ make clean
DREAM]$ make _dream_

DREAM]$ make clean
DREAM]$ make
```

2 Getting started

2.1 Input parameter file

An example of the input parameter file, named `explanatory.ini`, is already within the downloaded code. The input files takes all the parameters necessary to run the code. Only lines containing an equal sign without a sharp sign # at the beginning will be considered by the code, and any other lines will be fully ignored. The standard syntax for a useful line is: `parameter_name = value`. A space before and after the equal sign is required. An example of the layout of the input parameter file is provided in Figure 1.



```
# *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
# *  DREAM input parameter file  *
# *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*

# This example of input file lists all possible and allowed parameters
# and provides detailed explanations.  Only lines containing an equal
# sign not preceded by a sharp sign "#" will be considered by the code,
# any other line will be ignored.
# If some parameter is not given, the code will assume the default value.
# However, not all parameters have a default value and some are essential.

# The standard syntax is: parameter_name = value
# A space before and after the "=" sign is necessary.



# ----------------
# ----> Cosmology:
# ----------------

# 1) Insert cosmological model.
#    See https://bdiemer.bitbucket.io/colossus/cosmology_cosmology.html
#    for available models.
#    The cosmological model needs to be always defined.
#    If not given, the code will assume as default: planck18.
cosmological_model = planck18



# ---------------------
# ----> Mass functions:
# ---------------------

# 1) Halo mass function.
#    Insert halo mass function model.
#    See https://bdiemer.bitbucket.io/colossus/lss_mass_function.html
#    for available models.
#    Default: tinker08
halo_mass_function = tinker08
```

Figure 1: Example of input parameter file.

Some parameters have a default set value, if one of these is not provided (i.e. line not present or commented out) the code will assume the default value automatically. While, other parameter does not have a default value and are compulsory, is not provided the code will return an error message.

In this Section there is a list of allowed parameters.

2.1.1 Cosmology

```
cosmological_model = planck18
```

The parameter `cosmological_model` is extremely explicit. The user needs to specify the cosmological model assumed for the simulation. This is implemented through the colossus.cosmology[1] package. If not given, the code will assume Planck 2018 [5] parameters by default and a warning message will be printed.

2.1.2 Mass functions

```
halo_mass_function = tinker08
```

The parameter halo_mass_function states the type of halo mass function used to generate the main progenitors. This is implemented via the colossus.lss.mass_function[2] module. If not given, the code will assume Tinker et al. 2008 [7] HMF by default.

2.1.3 Dark matter catalogue

```
cube_side = 100

z_min = 0
z_max = 6
z_bin = 0.1

M0_host_min = 11
M0_host_max = 15
M0_host_bin = 0.1

mass_definition = 200m

use_mean_track = no

M_sub_res = 1e-3

max_order = 3
```

The parameter cube_side takes the length of the box side where you want to generate the catalogue. This should be provided in units of [Mpc/h]

The second set of parameters constraints the redshift interval, within which the code will calculate the central halo accretion track and mergers. The parameters z_bin denotes the resolution for drawing the probability distribution of the redshifts first accretion.

Then, we have the parameters to set the mass interval (in units of $\log_{10}[M/M_\odot]$), within which the code will generate the parent haloes. The parameters M0_host_bin stays for the resolution for drawing the halo mass function distribution.

The parameter mass_definition indicates the mass definition. This is implemented via the Python colossus package[3].

[1]Visit https://bdiemer.bitbucket.io/colossus/cosmology_cosmology.html, for the available cosmologies
[2]Visit https://bdiemer.bitbucket.io/colossus/lss_mass_function.html, for the available types.
[3]Visit https://bdiemer.bitbucket.io/colossus/halo_mass.html for details.

The option use_mean_track states how to calculate the accretion track of the parent dark matter haloes. Allowed options are 'yes' or 'no'. If yes it will assume one single track per halo mass bin (set to $\log(M_{\rm bin}/M_{\odot}) = 0.5$), while if no it will calculate the track for each parent halo (not recommended for CPU-time reasons). If not provided, this is set to 'no' by default.

The parameter M_sub_res represent the subhaloes mass resolution. For example, if the resolution is set to a certain value $f_{\rm res}$, for each parent halo of mass $M_{\rm par}$ the code will generate the subhaloes with mass higher than $f_{\rm res} M_{\rm par}$.

The parameter max_order indicates the maximum order of the subhaloes the user wants to generate. Insert an integer higher or equal to 1. Generally subhaloes of order higher than 3 have a negligible contribution. If not provided, this is set to 3 by default.

2.1.4 Mergers

```
use_merger_tree = no

type_orbital_circularity = constant
orbital_circularity = 0.5
fudge = -1
want_z_at_merge = yes
```

The option use_merger_tree gives the choice between a full merger tree and a statistical approach. The first option is beyond the purpose of this model, since we want to keep a statistical-empirical approach, thus this parameter is set to 'no' by default and we recommend the user not to change it.

The last set of parameters are needed for calculating the subhaloes merging timescale. This is implemented according to Eq. (5) in Boylan-Kolchin et al. 2008 [1], making use of McCavana et al. 2012 [4] fitting parameters. The parameter type_orbital_circularity indicates whether the user wants to assign a constant orbital circularity to subhaloes or extract it from a Gaussian distribution. In the first case the constant value should be specified to the parameter orbital_circularity, while in the second case it will be extracted from a Gaussian distribution centered in 0.5 and $\sigma = 0.23$ (see Khochfar et al. 2006 [3] and Shankar et al. 2014 [6]).

A fudge factor is introduced to adapt the merging timescale to match the results from N-body simulations and semi-analytical works. The interested reader can find an overview in Sec. 3.1.4 of Grylls et al. 2019 [2]. This should be passed to the parameter fudge as constant number. Normally a correction of 0.5 broadly matches the simulations, while a new linear dependence of the fudge on the progenitor/subhalo mass ratio further improves the results (set fudge to any negative value for applying this). If no correction should be applied, set fudge to 1 and the McCavana et al. 2012 [4] merging timescale will be calculated. If not given, the fudge will be set to 0.5 by default.

The parameter want_z_at_merge allows to choose whether to calculate and print to file the redshift at which the mergers have fully merged. Allowed options are 'yes' or 'no'. If not provided, this is set to 'no' by default.

2.1.5 Galaxies catalogue

```
want_galaxies = yes

exist_DM = yes

satellites_redshift = 0.1

ignore_high_orders = yes

SMHM_file = Data/SMHM_relations/example_file.txt

include_quenching = yes

include_stripping = yes
```

The switch `want_galaxies` states if the user wants to generate the catalogue of galaxies at a specific redshift z. This will be done by assigning a galaxy mass to each dark matter halo and subhalo and evolve it in time. Therefore, the user needs to specify whether a dark matter objects catalogue exists yet via the `exist_DM` option. If not the code will generate the latter as well, otherwise will simply read from the output folder.

The redshift at which the user desires the galaxies catalogue is specified by the parameter `satellites_redshift`. The option `ignore_high_orders` allows the user to choose to ignore the satellites of order higher than 1. This will give the advantage of a faster computation at the expenses of losing a negligible amount of information on the mergers.

The parameter `SMHM_file` (see Appendix A), as clearly suggests, represents the file that contains the stellar-mass-halo-mass (SMHM) relation, which will be used to connect haloes to galaxies. Finally, the last two parameters states whether to include quenching and tidal stripping in the evolution of the satellite galaxies.

2.1.6 Output parameters

```
save_parameters = yes
```

If the user wants to keep track of the input parameters given, it must be states through the parameter `save_parameters`.

2.2 Output directory

For each set of input parameters, you should use a different output folder. In this way, the directory will keep track of the exact run of the code. Not existing folders will be created automatically.

Output data will be store automatically into two `txt` files in a subfolder named `data`. One, named `output_parents.txt`, contains the information on the central haloes, while the other, named `output_mergers.txt`, contains the information on the mergers. If you provide, not intentionally, an already existing folder, the code will overwrite the existing files.

File `output_parents.txt` is organized as follows: it contains two columns representing respectively the ID of the central halo and its mass in units of $\log_{10}[M/M_{\odot}]$.

File `output_mergers.txt` is organized as follows: the first column contains the ID of the corresponding central halo, the second column the order of the subhalo, the third column the mass of the subhalo in units of $\log_{10}[M/M_{\odot}]$, the forth column the redshift at first accretion, the fifth column the merging timescale in units of [Gyr], and eventually the sixth column the redshift at full infall.

File `output.log` contains information on the number of columns and rows of the aforementioned files. This is needed when reading the files. It also contains the total number of haloes and subhaloes generated within the selected box. This is useful for extracting the mass functions.

2.3 Analysis of outputs and plotting

Once you have got a run, you can analyse the output data to get information on it. All the plots will be stored as `pdf` files into a subfolder named `plots`. In the current version of DREAM, you will be able to get plots of the (un)evolved subhalo mass function compared to the analytical one, the redshift distribution and merger rates.

You will need to provide again an input parameter file with the subcommand '-p'. See also file `info_example.param`. Below is a list of allowed parameters:

```
cosmological_model = planck18

exist_z_at_merge = no

want_unevolved_shmf = yes
want_evolved_shmf = yes
want_mergers_rate = yes

halo_masses = [12, 13, 14]
halo_mass_bin = 0.2

SMHM_file = Data/SMHM_relations/example_file.txt

compute_SF = yes
redshifts_for_SF = [0.1, 1, 2]
stellar_masses_for_SFR = [10.5, 11, 11.5]
```

The cosmological model must be the same as that used to run the code. The parameter `exist_z_at_merge` states whether in file `data/output_mergers.txt` the redshift at merging exists, if yes the code will read it from the file, otherwise will calculate it by itself.

The parameter `halo_masses` indicates the mass (in units of $\log_{10}[M/M_{\odot}]$) of the parent haloes for which you want to calculate the subhalo mass functions, these need to be provided in square brackets separated by a comma. The parameter `halo_mass_bin` is the bin width within which the parent haloes will be identified, 0.2 is recommended.

The parameter `SMHM_file` is the same as already explained in Section 2.1.5.

The next set of parameters concern the star formation rate. The option `compute_SF` states whether to calculate the quantities related to the star formation. The parameter `redshifts_for_SF` indicates the redshifts at which the $\mathrm{SFR} - M_*$ relation should be calculated. The parameter `stellar_masses_for_SF` indicates the stellar masses (in units of $\log_{10}[M/M_\odot]$) for which the $\mathrm{SFR} - z$ and SFHs relation should be calculated.

All the outputs will be save as txt files in the same directory where the catalogue has been read. In folder `Scripts` the user can find some useful predefined scripts for plotting the outputs.

3 Example of work session

Just downloaded and installed DREAM and wish to launch and analyse the first run.

You can first create some folders in order to keep the DREAM directory always tidy. Do for example:

```
DREAM]$ mkdir catalogues
DREAM]$ mkdir catalogues/planck18
DREAM]$ mkdir input
DREAM]$ mkdir scripts
```

Then copy `explanatory.ini` in your input folder, with a customized name, e.g. `planck18.ini` if the run is based on the Planck 2018 cosmological parameters, and edit it if needed:

```
DREAM]$ cp -r explanatory.ini input/planck18.ini
```

Then launch a quick run over a small volume:

```
DREAM]$ python dream/DREAM.py run -p input/planck18.ini -o
catalogues/planck18
```

You should get a print at terminal similar to the following:

```
Running DREAM (DiscRete sEmi-empiricAl Model)

Cosmological model set to: planck18

Reading parameters from file:
path-to-DREAM/DREAM/explanatory.ini

Generating catalog for a volume of (20.00 Mpc/h)^3
Number of haloes generated: 52

Calculating accretion tracks...
Accretion tracks calculated

Generating halo IDs...
Halo IDs generated

Generating mergers...
100\%[############################################]
52/52 [00:00<00:00, 239.58it/s]

Data stored in:
path-to-DREAM/DREAM/outputs/data/

Process completed correctly
Time elapsed: 1 sec
```

Check that the output files have been correctly created by typing:

```
DREAM]$ ls -o catalogues/planck18/data/
```

If both parents and mergers file exist, now you can analyse these with the subcommand 'info'. Remember to provide the input parameter for the analysis. You need also to provide the folder where the data is stored, without specifying the subfolder `data`. Execute for example:

```
DREAM]$ python dream/DREAM.py info -p info_example.param
catalogues/planck18
```

3.1 Example of calculation of redshift at merging

As last remark, if you chose not to store the redshift at merging, you can calculate it from the merging timescale. Basically, you need to add the merging timescale to the age of the Universe at first accretion. Then get the redshift corresponding to the age of the Universe at full merge. If the merger fully infalls later than today ($z = 0$), then set `z_at_merge` to -1.

Below is an example of script:

```
$ python
$ >>> import numpy as np
$ >>> from colossus.cosmology import cosmology
$ >>> cosmology.setCosmology("planck18")
$ >>> Cosmo = cosmology.getCurrent()
$ >>> path = #insert path to output folder
$ >>> z_infall = np.loadtxt(path+"data/output_mergers.txt",
$ usecols=2)
$ >>> merging_timescale = np.loadtxt(path+"data/output_mergers.txt",
$ usecols=3)
$ >>> age_at_merge = Cosmo.age(z_infall) + merging_timescale
$ >>> z_at_merge = np.zeros(age_at_merge.size)
$ >>> for i,Age in enumerate(age_at_merge):
$ >>>     if Age > Cosmo.age(0):
$ >>>         z_at_merge[i] = -1
$ >>>     elif Age <= Cosmo.age(0):
$ >>>         z_at_merge[i] = Cosmo.age(Age, inverse=True)
```

Now you are ready to go!

4 Functions

4.1 The `Python` functions

4.1.1 SMHM Relation Matrix

For DREAM to populate halos accurately at varying redshift, it requires a SMHM relation for any given redshift. The function `writematrix` produces a matrix of SMHM relations for a set of redshifts, given the input parameters `SMF`, `sigma`, `z_range`, and `title`. `SMF` is a 2D array, where the first row contains the stellar masses, and the second row contains the number densities for each stellar mass. `sigma` is either a `float` or a 1D array, describing the scatter of the SMF at each redshift in `z_range`, which is a 1D array containing the values of `z` for the SMHM relations to be calculated at. The output `.txt` file will be titled `title`, which is a `string`.

4.2 The `C` functions

A SMHM file

The SMHM file includes information on the SMHM relation for a specified redshift grid, which will be taken by the code and interpolated to compute the relation at any wanted redshift. The file, therefore, should be given in the format as described in Figure 2. The redshift array contains the aforementioned predefined grid. The Mhalo array represents the corresponding halo mass at each redshift, while the Mstar array is fixed. Finally, N_rows and N_cols are the number of rows and columns of the matrix, and should be specified to ease the reading from a C script.

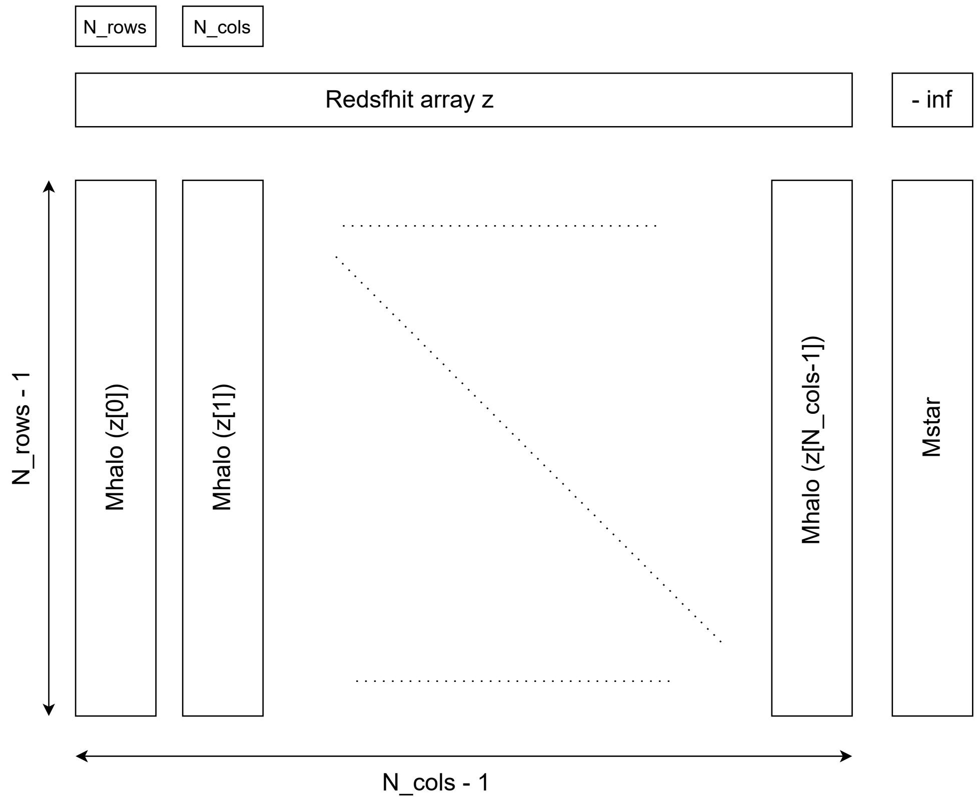


Figure 2: Format of the SMHM file.

References

[1] Michael Boylan-Kolchin, Chung-Pei Ma, and Eliot Quataert. "Dynamical friction and galaxy merging time-scales". In: 383.1 (Jan. 2008), pp. 93–101. DOI: 10.1111/j.1365-2966.2007.12530.x. arXiv: 0707.2960 [astro-ph].

[2] Philip J. Grylls et al. "A statistical semi-empirical model: satellite galaxies in groups and clusters". In: 483.2 (Feb. 2019), pp. 2506–2523. DOI: 10.1093/mnras/sty3281. arXiv: 1812.00015 [astro-ph.GA].

[3] S. Khochfar and A. Burkert. "Orbital parameters of merging dark matter halos". In: 445.2 (Jan. 2006), pp. 403–412. DOI: 10.1051/0004-6361:20053241. arXiv: astro-ph/0309611 [astro-ph].

[4] Tom McCavana et al. "The lives of high-redshift mergers". In: 424.1 (July 2012), pp. 361–371. DOI: 10.1111/j.1365-2966.2012.21202.x. arXiv: 1204.6319 [astro-ph.CO].

[5] Planck Collaboration et al. "Planck 2018 results. VI. Cosmological parameters". In: 641, A6 (Sept. 2020), A6. DOI: 10.1051/0004-6361/201833910. arXiv: 1807.06209 [astro-ph.CO].

[6] Francesco Shankar et al. "Environmental dependence of bulge-dominated galaxy sizes in hierarchical models of galaxy formation. Comparison with the local Universe". In: 439.4 (Apr. 2014), pp. 3189–3212. DOI: 10.1093/mnras/stt2470. arXiv: 1401.2460 [astro-ph.CO].

[7] Jeremy Tinker et al. “Toward a Halo Mass Function for Precision Cosmology: The Limits of Universality”. In: 688.2 (Dec. 2008), pp. 709–728. DOI: 10.1086/591439. arXiv: 0803.2706 [astro-ph].