Simbelmynë Documentation

Release 0.2.0

Florent Leclercq

Jul 19, 2019

Getting started

1 Installation 3

2 Examples 7

3 Parameter file 11

4 Internal structures 21

5 Reference 27

Simbelmynë is a cosmological software package written in C and Python. It is a hierarchical probabilistic simulator to generate synthetic galaxy survey data.

For more information on the code's purpose and features, please see its .

CHAPTER 1

Installation

This page contains instructions for installing the **Simbelmynë** code.

1.1 Installing the dependencies

1.1.1 Mandatory dependencies

The code requires the following software packages:

- (version 1.10 or higher)
- (version 3.3.6 or higher)
- (version 2.3 or higher)
- python 3, as provided for example by , with classic extensions such as NumPy, Matplotlib, etc.

The mandatory dependencies should be compiled using the following flags:

gsl

```
./configure --prefix=PATH/TO/gsl
make
make install
```

fftw

```
./configure --prefix=PATH/TO/fftw --enable-openmp --enable-shared
make
make install
./configure --prefix=PATH/TO/fftw --enable-openmp --enable-shared --enable-float
make
make install
```

Warning: Make sure to install *both* the double-precision version (first part) and the float-precision version (second part).

hdf5

```
./configure --prefix=PATH/TO/hdf5 --enable-shared
make
make install
```

anaconda3

Anaconda3 (see its documentation) comes with a bash installer, for instance:

```
bash Anaconda3-5.2.0-Linux-x86_64.sh
```

In case of compiler version conflict one may have to remove anaconda's gcc, and it may be necessary to install packages such as matplotlib:

```
conda remove libgcc
conda install matplotlib
```

1.1.2 Setting the environment variable

As the code links the dynamic libraries at compilation, you need to add the libraries' paths to your `LD_LIBRARY_PATH` before running:

```
# add some libraries to $LD_LIBRARY_PATH
export LD_LIBRARY_PATH=PATH/TO/gsl/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=PATH/TO/fftw/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=PATH/TO/hdf5/lib:$LD_LIBRARY_PATH
```

For convenience, the code above can be added to your `$HOME/.bashrc` (or `.bash_profile`, `.profile`, `.zshrc` etc.).

1.1.3 Optional dependencies

The following dependencies are optional but recommended; they are necessary to some features of the Simbelmynë code:

- and its python wrapper - used as an option to produce the input power spectrum of simulations
- - used anywhere projections on the sky are involved

1.2 Cloning the repository

Simbelmynë includes a main piece of code in the , as well as several optional "extensions" (to be released).

To download the main code, clone the repository using:

```
git clone git@bitbucket.org:florent-leclercq/simbelmyne.git SBMY_ROOT/
```

where you can replace `SBMY_ROOT/` with your desidered installation directory.

Note: Note that if you want to modify the code and/or some of the extensions, you shall first fork the corresponding repositories.

1.3 Adjusting the compilation options

Go to `SBMY_ROOT/` and open `config.mk` to edit your compilation options. You can add a "`SYSTYPE`" for your machine and adapt (in particular) the `LDFLAGS` and `INCLUDES` to match the path where you have installed your dependencies above. The following template can be used.

```
ifeq ($(SYSTYPE),"YourSystem")
    CC              = gcc
    RM              = rm -rf
    ECHO            = echo
    MKDIR           = @mkdir -p
    LDFLAGS         = -L PATH/TO/fftw/lib -L PATH/TO/gsl/lib -L PATH/TO/hdf5/lib
    LDLIBS          = -lm -lgsl -lgslcblas -lfftw3 -lfftw3_threads -lfftw3f -lfftw3f_
↪threads -lhdf5
    INCLUDES        = -I PATH/TO/fftw/include -I PATH/TO/gsl/include -I PATH/TO/hdf5/
↪include
    PYTHONFLAGS     = -fPIC -shared
    DEBUG           = "false"
endif
```

The following options (optimization flags depending on the C-compiler that is used, and code verbose level) can also be adjusted:

```
ifeq ($(DEBUG),"true")
    OPTIMIZE        = -fopenmp -g3 -ggdb -W -Wall -pedantic -pedantic-errors -std=c99
    OPT             += -DDEBUG
    OPT             += -DSCREEN_VERBOSE_LEVEL=7 -DLOGS_VERBOSE_LEVEL=7
else
    OPTIMIZE        = -fopenmp -O3 -std=c99
    OPT             += -DSCREEN_VERBOSE_LEVEL=4 -DLOGS_VERBOSE_LEVEL=6
endif
```

Then, under "Select target computer and optimization mode", add `SYSTYPE="YourSystem"` and comment out the other lines.

A number of compilation-time options are contained in `config.mk` and will determine the types of simulations that can be run with a given version of Simbelmynë. These options are documented in the template provided. In principle, it should not be necessary to modify the `makefile`.

Note: If you are a Simbelmynë developer, please do not include your private version of `config.mk` in your pull requests.

1.4 Recommended modifications of .bashrc

To run smoothly, the code generally requires to increase the stack size. We therefore suggest to add the following to your `$HOME/.bashrc` (or `.bash_profile`, `.profile`, `.zshrc` etc.)

```
# set stack size to unlimited
ulimit -s unlimited
```

1.5 Compiling the C code

Once everything is set up you should be able to compile:

```
make
```

An executable (`build/simbelmyne`) and a shared object (`pysbmy/build/simbelmyne.so`) are produced.

1.6 Installing the python wrapper package (pysbmy)

After successfully compiling the C code, install the python package using

```
pip install .
```

The list of python packages required for the installation can be found in `setup.py`.

1.7 Running the code

The code is run by using the entry point `simbelmyne` created by the python package, or by using directly the executable:

```
simbelmyne PATH/TO/config.sbmy PATH/TO/logs.txt
```

or

```
SBMY_ROOT/build/simbelmyne PATH/TO/config.sbmy PATH/TO/logs.txt
```

The first mandatory argument is the *Simbelmynë parameter file* (which can be generated using python), the second optional argument is the path for the output log file. If the second argument is omitted then logs will not be saved.

CHAPTER 2

Examples

This page contains instructions for running the **Simbelmynë** examples, assuming that the code has already been *installed*.

All the necessary scripts are in `SBMY_ROOT/examples/`. Typing `make` runs all the example of this page. At any point, one can restore a clean state by typing `make clean`.

2.1 Setting up the run

The script `example_setup.py` contains the configuration and can be adjusted (see the *parameter file* for a description of all the input parameters):

```
L0=L1=L2=250.
corner0=corner1=corner2=-125.
N0=N1=N2=128
Np0=Np1=Np2=128
Npm0=Npm1=Npm2=128
N_CAT=1
cosmo={'h':0.6774, 'Omega_r':0., 'Omega_q':0.6911, 'Omega_b':0.0486, 'Omega_m':0.3089,
↪ 'm_ncdm':0., 'Omega_k':0., 'tau_reio':0.066, 'n_s':0.9667, 'sigma8':0.8159, 'w0_fld
↪':-1., 'wa_fld':0., 'k_max':10.0, 'WhichSpectrum':"EH"}
```

The default setup will result in simulations that run easily on a standard laptop.

2.2 Preparing the inputs

2.2.1 Generating the parameter files

The first preparatory step is to generate the parameter files for the examples runs, using python. To do so, type `make parfiles`, or equivalently, run the python script `setup_parfiles.py`.

The following files are written as a result: `example_lpt.sbmy`, `example_pm.sbmy`, `example_tcola.sbmy`, `example_rsd.sbmy`, `example_mock.sbmy`.

2.2.2 Generating the input power spectrum

The second preparatory step is to compute the initial power spectrum to be used in the simulations, given the cosmological parameters and prescription specified in `example_setup.py`. This is achieved by typing `make power` or running the python script `setup_power.py`. The Fourier space k-binning to be used for final power spectra is also computed.

The following files are written as a result: `input_power.h5` and `input_ss_k_grid.h5`.

2.2.3 Generating the survey geometry

The third preparatory step is to prepare a simple synthetic survey geometry. To do so, type `make survey` or equivalently, run the python script `setup_survey_geometry.py`.

The following file is written as a result: `input_survey_geometry.h5`.

2.3 Running the simulations

We are now ready to run the actual simulations using the Simbelmynë executable. This is achieved by typing `make sims` or running the python script `run_examples.py`, which essentially executes the following commands:

```
pySbmy("example_lpt.sbmy", "logs_lpt.txt")
pySbmy("example_pm.sbmy", "logs_pm.txt")
pySbmy("example_tcola.sbmy", "logs_tcola.txt")
pySbmy("example_rsd.sbmy", "logs_rsd.txt")
pySbmy("example_mock.sbmy", "logs_mock.txt")
```

All the simulations start from the same initial conditions, generated by the first command and stored as `initial_density.h5`. The first run uses Lagrangian perturbation theory (LPT), the second uses a Particle-Mesh code, and the third uses tCOLA. The output density contrast fields are respectively `lpt_density.h5`, `final_density_pm.h5`, and `final_density_tcola.h5`. The fourth simulation uses tCOLA a forward model and additionnally puts dark matter particles in redshift space. The last simulation takes as input this redshift-space dark matter field (`rs_density_tcola.h5`) to produce a synthetic catalog (`output_mock_c0_n0.h5`) and compute its power spectrum as a summary statistic (`output_ss.h5`).

For the LPT run, dark matter particles are also stored in the file `lpt_particles.gadget3` (in GADGET HDF file format, see the).

The logs can be checked in the corresponding files (`logs_*.txt`).

2.4 Displaying the density fields

The command `make slices` uses the tool `slice` to display the density contrast in logarithmic scale (i.e. $\log(1+\delta)$). The result should look like this:

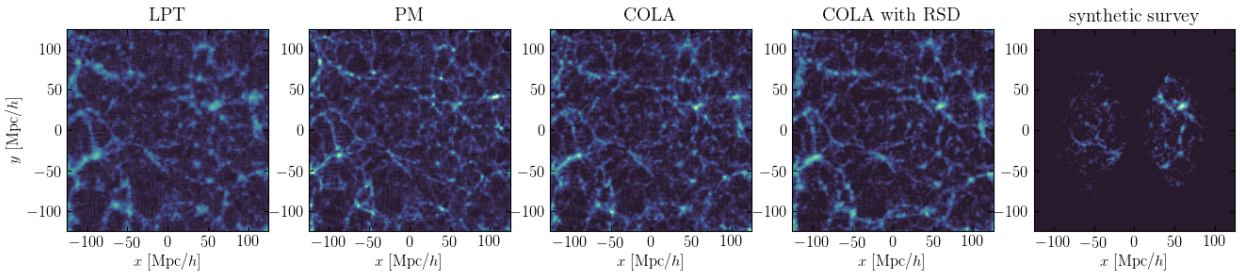

Fig. 1: Simbelmynë examples: from left to right: dark matter density evolved with LPT, dark matter density evolved with PM, dark matter density evolved with tCOLA, dark matter density evolved with tCOLA in redshift space, synthetic galaxy catalog.

2.5 Computing and plotting final power spectra

Using the outputs, the command `make test_power` (or directly the script `test_power.py`) produces a pdf plot of power spectra of the various fields, and a plot of dark matter power spectra divided by the linear power spectrum. The result should look like this:

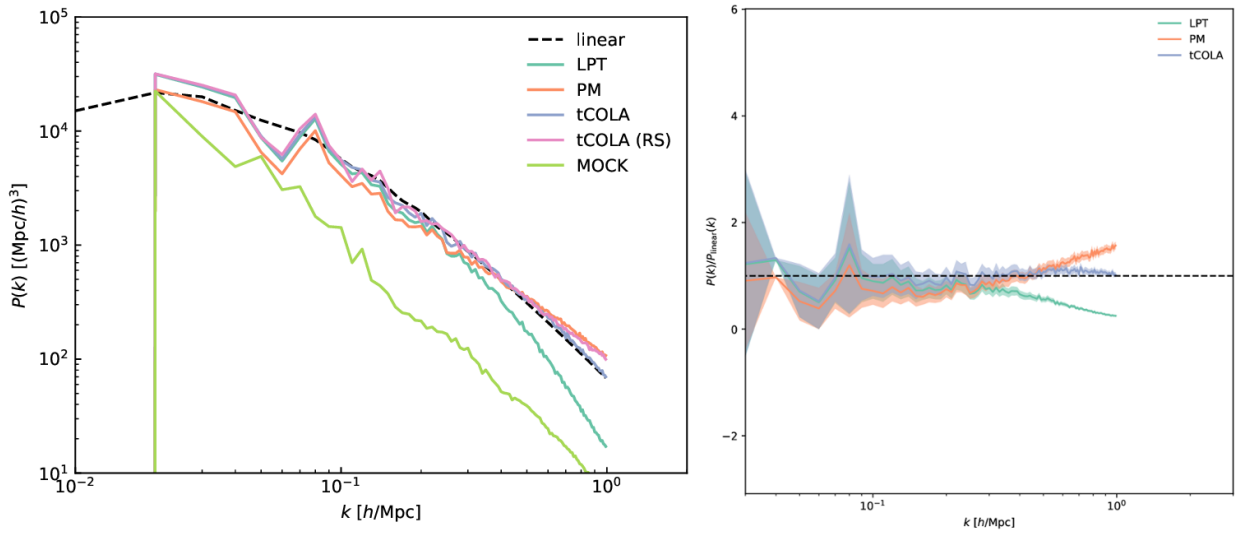

Fig. 2: Simbelmynë examples: left: power spectra of various fields; right: dark matter power spectra divided by the linear power spectrum.

CHAPTER 3

Parameter file

The **configuration file for Simbelmynë** is an ASCII text file. The recommended extension is `.sbmy`. A template is provided below, and it can also be generated using python.

3.1 Template

```
## ------------------------------------------------------------------------------------------
↪---------------------------------- ##
## ------------------------------      _                _           _                       ␣
↪      ---------------------------------- ##
## ------------------------------     (_)              | |         | |                      ␣
↪      ---------------------------------- ##
## ------------------------------  ___ _ _ __ ___ ___  | |__   ___| |_ __ ___   _   _ _ __  ␣
↪___   ---------------------------------- ##
## ------------------------------ / __| | '_ ` _ \| '_ \ / _ \ | '_ ` _ \| | | | '_ \ /
↪ _ \ ---------------------------------- ##
## ------------------------------ \__ \ | | | | | | |_) |  __/ | | | | | | |_| | | | |␣
↪ __/ ---------------------------------- ##
## ------------------------------ |___/_|_| |_| |_|_.__/ \___|_|_|_| |_| |_|\__, |_| |_
↪|\___| ---------------------------------- ##
## ------------------------------                                                __/ |      ␣
↪      ---------------------------------- ##
## ------------------------------                                               |___/       ␣
↪      ---------------------------------- ##
## ------------------------------------------------------------------------------------------
↪---------------------------------- ##

## ------------------------------------------------------------------------------------------
↪---------------------------------- ##
## Setup -----------------------------------------------------------------------------------
↪---------------------------------- ##
## ------------------------------------------------------------------------------------------
↪---------------------------------- ##
```

(continues on next page)

(continued from previous page)

```
SnapFormat                                      3
NumFilesPerSnapshot                             1

## ------------------------------------------------------------------------------
↪---------------------------------- ##
## Module LPT -----------------------------------------------------------------
↪---------------------------------- ##
## ------------------------------------------------------------------------------
↪---------------------------------- ##
ModuleLPT                                       1
InputRngStateLPT                                0
# Set this to '0' to reset the random number generator
OutputRngStateLPT                               dummy.rng

# Basic setup: -------------------------
Particles                                       128
# Number of particles in each dimension of the initial grid
Mesh                                            128
# The grid on which densities are computed
BoxSize                                         200.0
corner0                                         -100.0
corner1                                         -100.0
corner2                                         -100.0

# Initial conditions: ------------------
ICsMode                                         0
# 0 : the codes generates white noise, then initial conditions
# 1 : external white noise specified, the code multiplies by the power spectrum
# 2 : external initial conditions specified
WriteICsRngState                                0
OutputICsRngState                               dummy.rng
WriteWhiteNoise                                 0
OutputWhiteNoise                                initial_density_white.h5
# Only used if ICsMode=0
InputWhiteNoise                                 initial_density_white.h5
# Only used if ICsMode=1
InputInitialConditions                          initial_density.h5
# Only used if ICsMode=2
WriteInitialConditions                          0
OutputInitialConditions                         initial_density.h5
# At a scale factor of a=1e-3

# Power spectrum: ----------------------
InputPowerSpectrum                              input_power.h5
# Only used if ICsMode=0 or 1

# Final conditions: --------------------
RedshiftLPT                                     0.0
WriteLPTSnapshot                                1
OutputLPTSnapshot                               lpt_particles.gadget3
WriteLPTDensity                                 1
OutputLPTDensity                                lpt_density.h5

## ------------------------------------------------------------------------------
↪---------------------------------- ##
## Module PM/COLA -------------------------------------------------------------
↪---------------------------------- ##
```

(continues on next page)

(continued from previous page)

```
## ------------------------------------------------------------------------------------
↪---------------------------------- ##
ModulePMCOLA                                    0
EvolutionMode                                   2
# 1 : PM
# 2 : COLA
ParticleMesh                                    128
# The grid used in the particle-mesh code
NumberOfTimeSteps                               10
TimeStepDistribution                            0
# 0 : linear in the scale factor
# 1 : logarithmic in the scale factor
# 2 : exponential in the scale factor
ModifiedDiscretization                          1
# Use modified discretization of Kick and Drift operators in COLA
n_LPT                                           -2.5
# Exponent for the Ansatz in the modified Kick and Drift operators in COLA, only used␣
↪if ModifiedDiscretization=1

# Intermediate snapshots: --------------
WriteSnapshots                                  0
OutputSnapshotsBase                             particles_
OutputSnapshotsExt                              .gadget3
WriteDensities                                  0
OutputDensitiesBase                             density_
OutputDensitiesExt                              .h5

# Final snapshot: ----------------------
RedshiftFCs                                     0.0
WriteFinalSnapshot                              0
OutputFinalSnapshot                             final_particles.gadget3
WriteFinalDensity                               1
OutputFinalDensity                              final_density.h5

## ------------------------------------------------------------------------------------
↪---------------------------------- ##
## Module RSD ------------------------------------------------------------------------
↪---------------------------------- ##
## ------------------------------------------------------------------------------------
↪---------------------------------- ##
ModuleRSD                                       0
DoNonLinearMapping                              0
vobs0                                           0.0
vobs1                                           0.0
vobs2                                           0.0
# Velocity of the observer with respect to the CMB/galaxies frame
alpha1RSD                                       1.0
DoLPTSplit                                      0
alpha2RSD                                       1.0
# RSD model:
# if doLPTSplit=false:
#   z_obs = z_cosmo + (1+z_cosmo) * alpha1RSD * z_pec
# else:
#   z_obs = z_cosmo + (1+z_cosmo) * (alpha1RSD * z_pec_LPT + alpha2RSD * (z_pec-z_pec_
↪LPT))
# alpha1RSD=alpha2RSD=1 in LCDM
```

(continues on next page)

(continued from previous page)

```
# Reshift-space snapshot: --------------
WriteRSSnapshot                                 0
OutputRSSnapshot                                rs_particles.gadget3
WriteRSDensity                                  1
OutputRSDensity                                 rs_density.h5

## ------------------------------------------------------------------------------------
↪---------------------------------- ##
## Module Mock Catalogs ----------------------------------------------------------------
↪---------------------------------- ##
## ------------------------------------------------------------------------------------
↪---------------------------------- ##
ModuleMocks                                     0
InputRngStateMocks                              0
# Set this to '0' to reset the random number generator
OutputRngStateMocks                             dummy.rng
WriteMocksRngState                              0
OutputMocksRngState                             dummy.rng
NoiseModel                                      1
# 0 : Gaussian linear model
# 1 : Poisson model
NumberOfNoiseRealizations                       5
# Should be about NumberOfkBins^2 for a Poisson noise model

# Inputs: ------------------------------
InputDensityMocks                               lpt_density.h5
# Only used if ModuleLPT is switched off, uses final density produced by LPT/PM/COLA␣
↪otherwise
InputSurveyGeometry                             input_survey_geometry.h5
InputSummaryStatskGrid                          input_ss_k_grid.h5
# Only mandatory if WriteSummaryStats=1

# Output mocks: ------------------------
WriteMocks                                      1
OutputMockBase                                  output_mock_
OutputMockExt                                   .h5

# Output summaries: --------------------
WriteSummaryStats                               1
OutputSummaryStats                              output_ss.h5

## ------------------------------------------------------------------------------------
↪---------------------------------- ##
## Cosmological model ------------------------------------------------------------------
↪---------------------------------- ##
## ------------------------------------------------------------------------------------
↪---------------------------------- ##
# Planck 2015 cosmological parameters (Planck 2015 XIII, p31, table 4, last column)
# The equation of state of dark energy is parametrized by w(a) = w0_fld + (1-a)/a0 *␣
↪wa_fld
h                                               0.6774
Omega_r                                         0.0
Omega_q                                         0.6911
Omega_b                                         0.0486
Omega_m                                         0.3089
Omega_k                                         0.0
n_s                                             0.9667
```

(continues on next page)

(continued from previous page)

```
sigma8                                  0.8159
w0_fld                                  -1.0
wa_fld                                  0.0
```

3.2 Generation with python

The most convenient and recommended way to generate a Simbelmynë parameter file is using python.

The python class `param_file` defined in `pysbmy/pysbmy.py` represents a Simbelmynë parameter file. An instance can be created and written on the disk, using (for example):

```
from pysbmy import param_file
S=param_file(Particles=128, Mesh=128, BoxSize=250, corner0=-125, corner1=-125,␣
↪corner2=-125, WriteInitialConditions=1)
S.write("config.sbmy")
```

Any parameter that is not defined when creating an instance of the class `param_file` is set to its default value.

3.3 Description and default values

3.3.1 Setup

- `SnapFormat` (default `3`): Snapshot format, using standards of the Gadget simulation code. See the for a description. Accepted values: 1 (Gadget file format 1), 2 (Gadget file format 2), 3 (Gadget HDF file format).
- `NumFilesPerSnapshot` (default `1`): Number of files per snapshot.

3.3.2 Module LPT

- `ModuleLPT` (default `1`): Switch ON/OFF the module LPT. Accepted values: 0 or 1.
- `InputRngStateLPT` (default `0`): Path to the input random number generator state (before module LPT is executed). Set this to '0' to reset the random number generator
- `OutputRngStateLPT` (default `dummy.rng`): Path to the output random number generator state (after module LPT is executed).

Basic setup

- `Particles` (default `128`): Number of particles in each dimension of the initial grid
- `Mesh` (default `128`): Number of points in each dimension of the grid on which densities are computed
- `BoxSize` (default `200.0`): Box size in Mpc/h.
- `corner0` (default `-100.0`): x-position in Mpc/h of the corner of the box with respect to the observer (which is at (0,0,0)).
- `corner1` (default `-100.0`): y-position in Mpc/h of the corner of the box with respect to the observer (which is at (0,0,0)).

- `corner2` (default `-100.0`): z-position in Mpc/h of the corner of the box with respect to the observer (which is at (0,0,0)).

Initial conditions

- `ICsMode` (default `0`): Switch for the initial conditions mode. Accepted values: 0, 1, 2. 0 : the codes generates white noise, then initial conditions. 1 : external white noise specified, the code multiplies by the power spectrum. 2 : external initial conditions specified.
- `WriteICsRngState` (default `0`): Switch ON/OFF to write the random number generator state before generation of the initial conditions. Accepted values: 0 or 1.
- `OutputICsRngState` (default `dummy.rng`): Path to the random number generator state before generation of the initial conditions, if `WriteICsRngState` is `1`.
- `WriteWhiteNoise` (default `0`): Switch ON/OFF to write the white noise field.
- `OutputWhiteNoise` (default `initial_density_white.h5`): Path to the output white noise field (only used if `ICsMode` is `0` and `WriteWhiteNoise` is `1`).
- `InputWhiteNoise` (default `initial_density_white.h5`): Path to the input white noise field (only used if `ICsMode` is `1`).
- `InputInitialConditions` (default `initial_density.h5`): Path to the input initial conditions (only used if `ICsMode` is `2`).
- `WriteInitialConditions` (default `0`): Switch ON/OFF to write the initial conditons, at a scale factor of $a = 10^{-3}$.
- `OutputInitialConditions` (default `initial_density.h5`): Path to the output initial conditions (only used if `WriteInitialConditions` is `1` and `WriteInitialConditions` is `1`).

Power spectrum

- `InputPowerSpectrum` (default: `input_power.h5`): Path to the input power spectrum, in Simbelmynë standard HDF5 file format. Only used if `ICsMode` is `0` or `1`.

Final conditions

- `RedshiftLPT` (default: `0.0`): Redshift to which particles/densities are evolved after module LPT.
- `WriteLPTSnapshot` (default: `1`): Switch ON/OFF to write the snapshot after module LPT.
- `OutputLPTSnapshot` (default: `lpt_particles.gadget3`): Path to the snapshot after module LPT, if `WriteLPTSnapshot` is `1`.
- `WriteLPTDensity` (default: `1`): Switch ON/OFF to write the density field after module LPT.
- `OutputLPTDensity` (default: `lpt_density.h5`): Path to the snapshot after module LPT, if `WriteLPTDensity` is `1`.

3.3.3 Module PM/COLA

- `ModulePMCOLA` (default: `0`): Switch ON/OFF the module PM/COLA. Accepted values: 0 or 1.
- `EvolutionMode` (default: `2`): Evolution mode. Accepted values: 1 or 2. 1 : PM. 2 : COLA.

- `ParticleMesh` (default: `128`): Number of points in each dimension of the grid used in the particle-mesh code.
- `NumberOfTimeSteps` (default: `10`): Number of timesteps.
- `TimeStepDistribution` (default: `0`): Timesteps distribution. Accepted values: 0, 1, 2. 0 : linear in the scale factor. 1 : logarithmic in the scale factor. 2 : exponential in the scale factor.
- `ModifiedDiscretization` (default: `1`): Switch ON/OFF to use the modified discretization of Kick and Drift operators in COLA. For details see Appendix A in . Accepted values: 0 or 1.
- `n_LPT` (default: `-2.5`): Exponent for the Ansatz in the modified Kick and Drift operators in COLA, only used if `ModifiedDiscretization` is `1`.

Intermediate snapshots

- `WriteSnapshots` (default: `0`): Switch ON/OFF to write intermediate snapshots. Accepted values: 0 or 1.
- `OutputSnapshotsBase` (default: `particles_`): Base of the output snapshots filename, if `WriteSnapshots` is `1`.
- `OutputSnapshotsExt` (default: `.gadget3`): Extension of the output snapshots filename, if `WriteSnapshots` is `1`.
- `WriteDensities` (default: `0`): Switch ON/OFF to write intermediate density fields. Accepted values: 0 or 1.
- `OutputDensitiesBase` (default: `density_`): Base of the output density fields filename, if `WriteDensities` is `1`.
- `OutputDensitiesExt` (default: `.h5`): Extension of the output density fields snapshot filename, if `WriteDensities` is `1`.

Final snapshot

- `RedshiftFCs` (default: `0.0`): Redshift to which particles/densities are evolved after module PM/COLA.
- `WriteFinalSnapshot` (default: `0`): Switch ON/OFF to write the final snapshot. Accepted values: 0 or 1.
- `OutputFinalSnapshot` (default: `final_particles.gadget3`): Path to the output final snapshot, if `WriteFinalSnapshot` is `1`.
- `WriteFinalDensity` (default: `1`): Switch ON/OFF to write the final density field. Accepted values: 0 or 1.
- `OutputFinalDensity` (default: `final_density.h5`): Path to the output final density field, if `WriteFinalDensity` is `1`.

3.3.4 Module RSD

- `ModuleRSD` (default: `0`): Switch ON/OFF the module RSD. Accepted values: 0 or 1.
- `DoNonLinearMapping` (default: `0`): Switch ON/OFF to do the non-linear redshift-space distortions mapping. Accepted values: 0 or 1.
- `vobs0` (default: `0.0`): x-velocity of the observer with respect to the CMB/galaxies frame, in km/s.
- `vobs1` (default: `0.0`): y-velocity of the observer with respect to the CMB/galaxies frame, in km/s.
- `vobs2` (default: `0.0`): z-velocity of the observer with respect to the CMB/galaxies frame, in km/s.
- `alpha1RSD` (default: `1.0`): Value of α_1 (should be 1 in LCDM).

- `DoLPTSplit` (default: `0`): Switch ON/OFF to do split with the LPT velocity. Accepted values: 0 or 1.
- `alpha2RSD` (default: `1.0`): Value of α_2 (should be 1 in LCDM). Equivalent to `DoLPTSplit=0` if $\alpha_2 = \alpha_1$.

The RSD model is the following:

- if `DoNonLinearMapping=0` and `doLPTSplit=0`: $\mathbf{s} = \mathbf{r} + \alpha_1 \times (\mathbf{v} \cdot \mathbf{r}) \times \mathbf{r}/(H|\mathbf{r}|^2)$.
- if `DoNonLinearMapping=0` and `doLPTSplit=1`: $\mathbf{s} = \mathbf{r} + \alpha_1 \times (\mathbf{v}_{\rm LPT} \cdot \mathbf{r}) \times \mathbf{r}/(H|\mathbf{r}|^2) + \alpha_2 \times ((\mathbf{v} - \mathbf{v}_{\rm LPT}) \cdot \mathbf{r}) \times \mathbf{r}/(H|\mathbf{r}|^2)$.
- if `DoNonLinearMapping=1` and `doLPTSplit=0`: $z_{\rm obs} = z_{\rm cosmo} + (1 + z_{\rm cosmo}) \times \alpha_1 \times z_{\rm pec}$.
- if `DoNonLinearMapping=1` and `doLPTSplit=1`: $z_{\rm obs} = z_{\rm cosmo} + (1 + z_{\rm cosmo}) \times (\alpha_1 \times z_{\rm pec,LPT} + \alpha_2 \times (z_{\rm pec} - z_{\rm pec,LPT}))$.

Reshift-space snapshot

- `WriteRSSnapshot` (default: `0`): Switch ON/OFF to write the redshift-space snapshot. Accepted values: 0 or 1.
- `OutputRSSnapshot` (default: `rs_particles.gadget3`): Path to the output redshift-space snapshot, if `WriteRSSnapshot` is `1`.
- `WriteRSDensity` (default: `1`): Switch ON/OFF to write the redshift-space density field. Accepted values: 0 or 1.
- `OutputRSDensity` (default: `rs_density.h5`): Path to the output redshift-space density field, if `WriteRSDensity` is `1`.

3.3.5 Module Mock Catalogs

- `ModuleMocks` (default: `0`): Switch ON/OFF the module Mock Catalogs. Accepted values: 0 or 1.
- `InputRngStateMocks` (default: `0`): Path to the input random number generator state (before module Mock Catalogs is executed). Set this to '0' to reset the random number generator
- `OutputRngStateMocks` (default: `dummy.rng`): Path to the output random number generator state (after module Mock Catalogs is executed).
- `WriteMocksRngState` (default: `0`): Switch ON/OFF to write the random number generator state before generation of mock catalogs. Accepted values: 0 or 1.
- `OutputMocksRngState` (default `dummy.rng`): Path to the random number generator state before generation of mock catalogs, if `WriteMocksRngState` is `1`.
- `NoiseModel` (default: `1`): Noise model. Accepted values: 0, 1. 0 : Gaussian linear model. 1 : Poisson model.
- `NumberOfNoiseRealizations` (default: `5`): Number of noise realizations desired. Should be of order number of k-bins squared for a Poisson noise model.

Inputs

- `InputDensityMocks` (default: `lpt_density.h5`): Path to the input density for mock catalogs. Only used if ModuleLPT is switched off, uses final density produced by LPT/PM/COLA otherwise.
- `InputSurveyGeometry` (default: `input_survey_geometry.h5`): Path to the input survey geometry.
- `InputSummaryStatskGrid` (default: `input_ss_k_grid.h5`): Path to the input k-grid for the output summary statistics. Only mandatory if `WriteSummaryStats` is `1`.

Output mocks

- `WriteMocks` (default: `1`): Switch ON/OFF to write the mock catalogs. Accepted values: 0 or 1.
- `OutputMockBase` (default: `output_mock_`): Base of the output mock catalogs filename, if `WriteMocks` is `1`.
- `OutputMockExt` (default: `.h5`): Extension of the output mock catalogs filename, if `WriteMocks` is `1`.

Output summaries

- `WriteSummaryStats` (default: `1`): Switch ON/OFF to write summary statistics. Accepted values: 0 or 1.
- `OutputSummaryStats` (default: `output_ss.h5`): Path to the output summary statistics, if `WriteSummaryStats` is `1`.

3.3.6 Cosmological model

The cosmological model. Default values are Planck 2015 cosmological parameters (see , page 32, table 4, last column). The equation of state of dark energy is parametrized (as in CLASS) by $w(a) = w_0 + (1 - a)/a_0 \times w_a$.

- `h` (default: `0.6774`): Hubble constant
- `Omega_r` (default: `0.0`): radiation density
- `Omega_q` (default: `0.6911`): dark energy density
- `Omega_b` (default: `0.0486`): baryon density
- `Omega_m` (default: `0.3089`): matter density
- `Omega_k` (default: `0.0`): curvature density
- `n_s` (default: `0.9667`): spectral index
- `sigma8` (default: `0.8159`): σ_8
- `w0_fld` (default: `-1.0`): dark energy equation of state
- `wa_fld` (default: `0.0`): dark energy equation of state

3.4 Known issue

Sometimes when the ASCII template is used and/or the parameter file is modified "by hand" using certain text editors (among which vim), Simbelmynë will produce an error when parsing:

```
[XX:XX:XX|STATUS    ]|Reading parameter file in 'config.sbmy'...
[XX:XX:XX|ERROR     ]==|Error in utils.c:p_fgets:197: Incorrect reading in read_param.
↪c:read_parameterfile:603.
[XX:XX:XX|ERROR     ]==|Error in utils.c:p_fgets:199: Attempting to read: .
[XX:XX:XX|ERROR     ]==|Error in utils.c:p_fgets:200: I/O error (fgets)
```

This seems to be linked to end-of-line characters and will be later fixed. For the moment, the most convenient way to bypass this problem is to *use the python generator*.

CHAPTER 4

Internal structures

Simbelmynë uses internally several structures.

To each Simbelmynë structure corresponds a C-structure (defined in `libSBMY/include/structures.h`) and a python class (found in `pysbmy/*.py`). C and python reading/writing routines for HDF5 files are provided. The C reading/writing routines are defined in `libSBMY/src/io.c` and the python equivalents in `pysbmy/*.py`.

4.1 Field structure

This structure is used for cosmological fields of rank r (typically $r = 1$ for a scalar field and $r = 3$ for a vector field) defined on a 3D cartesian grid.

4.1.1 Content

The HDF5 file structure is:

- `/info/scalars/L0` (attribute, type `double`): size of the box in Mpc/h
- `/info/scalars/L1` (attribute, type `double`): size of the box in Mpc/h
- `/info/scalars/L2` (attribute, type `double`): size of the box in Mpc/h
- `/info/scalars/corner0` (attribute, type `double`): x-position in Mpc/h of the corner of the box with respect to the observer (which is at (0,0,0)).
- `/info/scalars/corner1` (attribute, type `double`): y-position in Mpc/h of the corner of the box with respect to the observer (which is at (0,0,0)).
- `/info/scalars/corner2` (attribute, type `double`): z-position in Mpc/h of the corner of the box with respect to the observer (which is at (0,0,0)).
- `/info/scalars/N0` (attribute, type `int`): number of volume elements of the mesh
- `/info/scalars/N1` (attribute, type `int`): number of volume elements of the mesh
- `/info/scalars/N2` (attribute, type `int`): number of volume elements of the mesh

- `/info/scalars/rank` (attribute, type `int`): rank of the field (usually 1 or 3)
- `/info/scalars/time` (attribute, type `double`): time at which the field is represented
- `/scalars/field` (dataset, type `4-byte float little-endian` (`<f4`), dimensions (`N0`, `N1`, `N2`)): the field

4.1.2 Use in C

In Simbelmynë C files, a Field structure can be read, allocated, written and freed using respectively:

```
Field F = read_field("field.h5")
Field F = allocate_field(rank, N0, N1, N2, corner0, corner1, corner2, L0, L1, L2,␣
↪Time);
Field F = allocate_scalar_field(N0, N1, N2, corner0, corner1, corner2, L0, L1, L2,␣
↪Time);
write_field(F, "field.h5");
free_field(F);
```

4.1.3 Use in python

With python, a field structure can be read/written using:

```
from pysbmy.field import read_field
F=read_field("field.h5")
F.write("field.h5")
```

4.2 FourierGrid and PowerSpectrum structures

These structures are used for cosmological power spectra. The entire Fourier grid of the simulation box is stored. The PowerSpectrum structure is a super-structure of FourierGrid.

4.2.1 Content

The HDF5 file structure for a FourierGrid is:

- `/info/scalars/L0` (attribute, type `double`): size of the box in Mpc/h
- `/info/scalars/L1` (attribute, type `double`): size of the box in Mpc/h
- `/info/scalars/L2` (attribute, type `double`): size of the box in Mpc/h
- `/info/scalars/N0` (attribute, type `int`): number of volume elements of the mesh
- `/info/scalars/N1` (attribute, type `int`): number of volume elements of the mesh
- `/info/scalars/N2` (attribute, type `int`): number of volume elements of the mesh
- `/info/scalars/N2_HC` (attribute, type `int`): number of modes in the half-complex Fourier space in the z-direction (N2_HC $= (N2 + 1)/2$; `N2` should be a multiple of 2)
- `/info/scalars/N_HC` (attribute, type `int`): number of modes in the half-complex Fourier space (N_HC $=$ N0 $\times$ N1 $\times$ N2_HC)
- `/info/scalars/NUM_MODES` (attribute, type `int`): number of k modes

- `/info/scalars/kmax` (attribute, type `double`): maximum wavenumber
- `/info/scalars/k_keys` (attribute, type `int`, dimensions (`N0`, `N1`, `N2_HC`)): the keys to be used as a function of the Fourier mode ($\mathrm{k\_keys}[i][j][k] = b$)
- `/info/scalars/k_modes` (attribute, type `4-byte float little-endian (<f4)` dimension `NUM_MODES`): value of the wavenumber for this key ($k[b]$)
- `/info/scalars/k_nmodes` (attribute, type `int`, dimension `NUM_MODES`): number of modes for this key ($N_k[b]$)

A PowerSpectrum structure contains additionally:

- `/scalars/powerspectrum` (dataset, type `4-byte float little-endian (<f4)` dimension `NUM_MODES`): value of the power spectrum for this key ($P[b]$)

4.2.2 Use in C

In Simbelmynë C files, a FourierGrid structure can be read, written and freed using respectively:

```
FourierGrid G = read_FourierGrid("fouriergrid.h5")
write_FourierGrid(G, "fouriergrid.h5");
free_FourierGrid(G);
```

and similarly for a PowerSpectrum structure:

```
PowerSpectrum P = read_PowerSpectrum("powerspectrum.h5")
write_PowerSpectrum(P, "powerspectrum.h5");
free_PowerSpectrum(P);
```

4.2.3 Use in python

With python, a FourierGrid structure can be read/written using:

```
from pysbmy.fft import read_FourierGrid
G=read_FourierGrid("fouriergrid.h5")
G.write("fouriergrid.h5")
```

and a PowerSpectrum structure using:

```
from pysbmy.power import read_powerspectrum
P=read_powerspectrum("powerspectrum.h5")
P.write("powerspectrum.h5")
```

4.3 SurveyGeometry and GalaxySelectionWindow structures

These structures are used for survey geometry and selection windows.

4.3.1 Content

For both SurveyGeometry and GalaxySelectionWindow structures, HDF5 files contain:

- `/info/scalars/L0` (attribute, type `double`): size of the box in Mpc/h

- /info/scalars/L1 (attribute, type double): size of the box in Mpc/h
- /info/scalars/L2 (attribute, type double): size of the box in Mpc/h
- /info/scalars/corner0 (attribute, type double): x-position in Mpc/h of the corner of the box with respect to the observer (which is at (0,0,0)).
- /info/scalars/corner1 (attribute, type double): y-position in Mpc/h of the corner of the box with respect to the observer (which is at (0,0,0)).
- /info/scalars/corner2 (attribute, type double): z-position in Mpc/h of the corner of the box with respect to the observer (which is at (0,0,0)).
- /info/scalars/N0 (attribute, type int): number of volume elements of the mesh
- /info/scalars/N1 (attribute, type int): number of volume elements of the mesh
- /info/scalars/N2 (attribute, type int): number of volume elements of the mesh
- /info/scalars/N_COSMOPAR (attribute, type int): number of cosmological parameters stored
- /info/scalars/cosmo (attribute, type double, dimension N_COSMOPAR): cosmological parameters stored in the default order of Simbelmynë (see pysbmy/cosmology.py)

SurveyGeometry structures contain additionally:

- /info/scalars/bright_cut (attribute, type double, dimension N_CAT): lower value of the cuts in magnitude in each subcatalog
- /info/scalars/faint_cut (attribute, type double, dimension N_CAT): upper value of the cuts in magnitude in each subcatalog
- /info/scalars/rmin (attribute, type double, dimension N_CAT): minimum comoving distance of objects in each subcatalog
- /info/scalars/rmax (attribute, type double, dimension N_CAT): maximum comoving distance of objects in each subcatalog
- /info/scalars/zmin (attribute, type double, dimension N_CAT): minimum redshift of objects in each subcatalog
- /info/scalars/zmax (attribute, type double, dimension N_CAT): maximum redshift of objects in each subcatalog
- /info/scalars/N_BIAS (attribute, type int): number of bias parameters
- /info/scalars/galaxy_bias_mean (attribute, type double, dimensions (N_CAT, N_BIAS)): mean values of bias parameters in each subcatalog
- /info/scalars/galaxy_bias_std (attribute, type double, dimensions (N_CAT, N_BIAS)): standard deviations of bias parameters in each subcatalog
- /info/scalars/galaxy_nmean_mean (attribute, type double, dimension N_CAT): mean value of the expected number of galaxies in each subcatalog
- /info/scalars/galaxy_nmean_std (attribute, type double, dimension N_CAT): standard deviation of the expected number of galaxies in each subcatalog
- /info/scalars/N_CAT (attribute, type int): number of subcatalogs
- /scalars/galaxy_sel_window_{ICAT} (dataset, type 4-byte float little-endian (<f4), dimensions (N0, N1, N2)) for $0 \leq \mathrm{ICAT} < \mathrm{N\_CAT}$: the 3D galaxy selection window for each subcatalog

GalaxySelectionWindow structures contain additionally:

- `/info/scalars/bright_cut` (attribute, type `double`): lower value of the cuts in magnitude in each subcatalog
- `/info/scalars/faint_cut` (attribute, type `double`): upper value of the cuts in magnitude in each subcatalog
- `/info/scalars/rmin` (attribute, type `double`): minimum comoving distance of objects in each subcatalog
- `/info/scalars/rmax` (attribute, type `double`): maximum comoving distance of objects in each subcatalog
- `/info/scalars/zmin` (attribute, type `double`): minimum redshift of objects in each subcatalog
- `/info/scalars/zmax` (attribute, type `double`): maximum redshift of objects in each subcatalog
- `/info/scalars/N_BIAS` (attribute, type `int`): number of bias parameters
- `/info/scalars/galaxy_bias_mean` (attribute, type `double`, dimension `N_BIAS`): mean values of bias parameters in each subcatalog
- `/info/scalars/galaxy_bias_std` (attribute, type `double`, dimension `N_BIAS`): standard deviations of bias parameters in each subcatalog
- `/info/scalars/galaxy_nmean_mean` (attribute, type `double`): mean value of the expected number of galaxies in each subcatalog
- `/info/scalars/galaxy_nmean_std` (attribute, type `double`): standard deviation of the expected number of galaxies in each subcatalog
- `/scalars/galaxy_sel_window_{ICAT}` if `ICAT` is defined, or `/scalars/field` (dataset, type `4-byte float little-endian` (`<f4`), dimensions (`N0`, `N1`, `N2`)): the 3D galaxy selection window

4.3.2 Use in C

C routines to read and free SurveyGeometry and GalaxySelectionWindow structures are available and can be used as follows:

```
SurveyGeometry SG = read_survey_geometry_header("survey_geometry.h5");
free_survey_geometry_header(SG);
GalaxySelectionWindow GSW = read_galaxy_selection_window("galaxy_sel_window.h5",␣
↪ICAT);
free_galaxy_selection_window(GSW);
```

`read_galaxy_selection_window` will only work if `ICAT` is set in the file to be read.

4.3.3 Use in python

With python, a SurveyGeometry structure can be read/written using:

```
from pysbmy.survey_geometry import read_survey_geometry
SG=read_survey_geometry("survey_geometry.h5");
SG.write("survey_geometry.h5");
```

and a GalaxySelectionWindow structure using:

```
from pysbmy.survey_geometry import read_galaxy_sel_window
GSW=read_galaxy_sel_window("galaxy_sel_window.h5")
GSW.write("galaxy_sel_window.h5")
```

CHAPTER 5

Reference

Simbelmynë does not have its scientific paper yet. For the time being, if you want to acknowledge use of this software, please cite , where it was first used:

Bayesian analysis of the dynamic cosmic web in the SDSS galaxy survey
F. Leclercq, J. Jasche, B. Wandelt

```
@ARTICLE{2015JCAP...06..015L,
    author = {{Leclercq}, Florent and {Jasche}, Jens and {Wandelt}, Benjamin},
    title = "{Bayesian analysis of the dynamic cosmic web in the SDSS galaxy survey}",
    journal = {Journal of Cosmology and Astro-Particle Physics},
    keywords = {Astrophysics - Cosmology and Nongalactic Astrophysics},
    year = 2015,
    month = Jun,
    volume = {2015},
    eid = {015},
    pages = {015},
    doi = {10.1088/1475-7516/2015/06/015},
    archivePrefix = {arXiv},
    eprint = {1502.02690},
    primaryClass = {astro-ph.CO},
    adsurl = {https://ui.adsabs.harvard.edu/\#abs/2015JCAP...06..015L},
    adsnote = {Provided by the SAO/NASA Astrophysics Data System}
}
```