CLASS, `hi_class` and Monte Python basics
IFT School on Cosmology Tools

Miguel Zumalacarregui[1,2]

[1]Nordic Institute for Theoretical Physics
[2]Berkeley Center for Cosmological Physics

March 11, 2017

Abstract

The present document summarizes the basics for the installation and execution of the CLASS, `hi_class` and Monte Python codes. It is not meant as a comprehensive guide, but rather a basic guide to get the codes up and running and a complement to the school exercises. Resources for further learning, including the official websites, documentation and links to courses are also provided.

Contents

1 Resources

Websites:

- CLASS: `https://class-code.net`
  `https://github.com/lesgourg/class_public`
- `hi_class`: `https://hiclass-code.net`
  `https://github.com/miguelzuma/hi_class_public`
- Monte Python: `http://baudren.github.io/montepython.html`
  `https://github.com/baudren/montepython_public`

Documentation

- The codes themselves: nearly as many comment as code lines.
- CLASS: `http://lesgourg.github.io/class_public/class_public-2.5.0/doc/manual/html/index.html`
- Montepython: `http://monte-python.readthedocs.io/en/latest/`

Courses:

- CLASS & MP course by Audren, Lesgourgues and Tram ($\sim 13h$)
  `https://lesgourg.github.io/class-tour-Tokyo.html`
- CLASS lecture by Julien Lesgourgues ($\sim 4h$)
  `https://lesgourg.github.io/class-tour/Narbonne.pdf`
- Montepython's brief presentation by S. Clesse ($\ll 1h$)
  `https://lesgourg.github.io/class-tour/16.06.02_Lisbon_intro.pdf`
- CLASS video tutorial ($\sim 25'$)
  `https://www.youtube.com/watch?v=R22XhKUwzX4`
  (beta version, suggestions welcome!)

Troubleshooting: forums to 1) find answers and 2) ask questions (in that order)

- CLASS: `https://github.com/lesgourg/class_public/issues`
- `hi_class`: `https://github.com/miguelzuma/hi_class_public/issues`
  `https://groups.google.com/d/forum/hi_class`
- Montepython: `https://github.com/baudren/montepython_public/issues`

Few `git` resources (version control):

- `git` online book: `https://git-scm.com/book/en/Getting-Started-Git-Basics`
- `git` - the simple guide: `rogerdudler.github.io/git-guide/`
- Interactive `git` in 15': `https://try.github.io/`

2 Download

In the terminal type

```
git clone https://github.com/lesgourg/class_public.git
git clone https://github.com/miguelzuma/hi_class_public.git
git clone https://github.com/baudren/montepython_public.git
```

This requires `git` but gives you access to all the previous versions (you should try `git`, your life won't be the same). If you don't have `git` click 'Clone or download', 'Download ZIP' in the above address (and seriously, you should try `git`).

3 Using CLASS & `hi_class`

3.1 Compile CLASS

In a terminal go to the `class_public` or `hi_class_public` directory and enter

```
make
```

to compile the executable and *classy*, the python wrapper (this should install the python wrapper locally). Enter `make class` to build *only* the executable. Remember to enter `make clean` before to recompile the python wrapper or if you have modified a header file (extension `.h`). For details on *classy* see `https://github.com/lesgourg/class_public/wiki/Python-wrapper`.

3.2 Input parameters

The code can be run on the terminal or through *classy*, the Python wrapper. The following parameters give a Planck ΛCDM model

Terminal: write `your_parameter_file.ini`

```
h =  0.6774
omega_b = 0.02230
Omega_cdm = 0.2603
Omega_fld = 0
Omega_smg =  0 #GR in hi_class
background_verbose = 1 #info
output = tCl,mPk #what to compute
write background = y
root = output/your_model_ #future files
```

Python: write a dictionary

```
params = {
    "h": 0.6774,
    "omega_b": 0.02230,
    "Omega_cdm": 0.2603,
    "Omega_fld" : 0,
    "Omega_smg" : 0, #GR in hi_class
    "background_verbose" : 1, #info
    "output" : "tCl,mPk" #observables
}
```

- For modified gravity in `hi_class` you need to set `Omega_Lambda = 0` (no CC) and `Omega_smg = -1` (determine DE density automatically) and specify a `gravity_model_smg`, `expansion_model_smg`.
- The parameter file can be as short as you need, with unspecified parameters set to default values.
- Only lines with an equal sign (=) will be interpreted. Hashtag (#) comments a line.
- Unused or misspelled parameters will be written to an `unused_parameters` file (using the option `write parameters = y`). Setting `write warnings = y` makes CLASS complain in those cases.
- The `root` directory has to exist.

⚠ All the model and output parameters are described in `explanatory.ini` (this is the first place to look for information). The `hi_class` parameters are described in `hi_class.ini`. Keep those files for reference.

3.3 Run CLASS

To run the code on terminal or *classy*:

Terminal:

From the to the base directory run

```
./class your_parameter_file.ini
```

(plus an optional `.pre` precision file)

Your output will be ready in the `root` address.

Python:

```
from classy import Class
cosmos = Class() #create universe
cosmos.set(params) #feed params to cosmos
cosmos.compute() #duh...
...        #play with the output
cosmo.struct_cleanup() #free memory
cosmo.empty() #start over
```

See Exercise 1 in the CLASS sheet to familiarize yourself with the output options available to CLASS.

4 Using Montepython

Python is an interpreted language: you don't need to compile MP, but you need to configure it.

Montepython has two execution modes (see below). For help type in the MP directory

```
python montepython/MontePython.py run --help
python montepython/MontePython.py info --help
```

A very useful example of a complete work session with Montepython is explained in

`http://monte-python.readthedocs.io/en/latest/example.html`

4.1 Configuration file (`.conf`)

You need a `.conf` file to inform MP of the CLASS/`hi_class` (mandatory) and Planck likelihood (optional). Read the `default.conf.template` file for details and instructions. Unless other file is specified MP will read from `default.conf` (but you need to create this first). This is important if you use different CLASS versions (eg. `class_public` and `hi_class_public`).

4.2 Parameter file (`.param`)

Montepython runs with a `.param` file that specifies the model to be analyzed, data to use and other specifications. This can be kept rather minimal (with unspecified parameters set to defaults).

```
#what expermiennts to include in tha analysis
data.experiments=['bao_boss','bao_boss_aniso']
# parameters: data.parameters[class name] = [mean, min, max, 1-sigma, scale, role]
#cosmological paramters to vary (role = 'cosmo' and 1-sigma not 0)
data.parameters['omega_cdm'] = [0.1120, None,None, 0.0016, 1, 'cosmo']
data.parameters['h']         = [0.703, None,None, 0.0065, 1, 'cosmo']
#fixed cosmologial arguments (also if you fix sigma=0)
data.cosmo_arguments['omega_b'] = 0.0222
#Nuisance parameters if your likelihood needs them
#derived parameters (role = 'derived')
data.parameters['z_reio']    = [0,     None, None, 0,1, 'derived']
data.parameters['Omega_Lambda'] = [0,   None, None, 0,1, 'derived']
#Montepython execution options
data.N=10
data.write_step=5
```

- Please read `base.param` for further details. Other `.param` files can be useful too.

  Do not comment in the same line: `data.parameters['...']= [...] #mycomment here` will not be read!

- Each experiment is a directory in `ls montepython/likelihoods`. More details in each subfolder.

⚠ **Parameter vectors:** For parameters that enter class as a vector (like `m_ncdm` if `N_cdm` > 1 or `parameters_smg,expansion_smg` in `hi_class`) you need the following format:

```
data.parameters['m_ncdm__1'] = [0.05, 0, None, 0., 1, 'cosmo']
data.parameters['m_ncdm__2'] = [0.01 , 0, None, 0. , 1, 'cosmo']
```

with two underscores `__` after the parameter name to specify the position in the entry (if you specify `data.cosmo_arguments['N_ncdm'] = 2`, if `N_ncdm = 1` you don't need this syntax). The above corresponds to `m_ncdm = 0.05,0.01` in an `.ini` file.

You need to specify all vector parameters, even when some are not varied.

4.3 Run chain(s)

To compute a chain for a given model, type from the terminal

```
python montepython/MontePython.py run -p model.param -o output_directory
```

This will start a new chain in the designated `output_directory` as specified in `model.param`.

- Each output directory is for a choice of model/parameters and experiments. The first run in `output_directory` will create a `log.param` with all the specifications for the run. If this file exists the code will ignore `model.param` and will instead pick the settings from the `log.param`. This ensures that all the chains in a given directory are consistent.
- This is a very minimal run, and will only produce 10 points (controlled by `-N`, good for debug). Add at least `-N 10000` (or more) if you run a chain for real. Add `--update 300` to control how often you update the covariance matrix.

  See all the options and their default values running MP with `run --help`.
- Note that MP's parallelization is optional: you can run several instances of the code (one per core) by repeating the instructions above. Pro tip: run a short sequence with `-N 10` to test the code and create the `log.param` , then type

  ```
  for n $(seq 1 4); do python montepython/Montepython run \
                      -o output_directory -N 100000 [other_options]; done
  ```

  (this will run 4 chains in `output_directory` with 10^5 points). Optional parallelization 1) allows you to run in any old computer and 2) simplifies your life when you "meet" a new cluster (and you may "meet" many in your career!).
- Each chain file name reads

  ```
  yyyy-mm-dd_N__id.txt
  ```

  where `yyyy-mm-dd` is the date in which the chain was launched, `N` is the desired number of points and `id` is the identifier (when the date and `N` are the same).
- The run will also create some additional files such as a covariance matrix. Additional files are created during the chain analysis, see below.
- To increase the convergence it is recommended to run with a covariance matrix (option `-c your_file.covmat`), especially for models with many parameters.

4.4 Analyze chains

Once you have several decent-sized chains in a file you can analyze them:

```
python montepython/MontePython.py info output_directory/
```

The above command will analyze all the chains in `output_directory`, computing the convergence of the chains over different paramters and producing statistical information on the posterior, including confidence intervals and plots.

- Several files named `output_directory.*` will be produced:
  - `.bestfit` for the best fit model, `.covmat` for a covariance matrix. These can be passed for future runs.
  - `.log` for the information on the chains.
  - `h_info`, `v_info` with horizontal/vertical tables of the parameter constraints, `.tex` for table in latex format
  - ⋆ a directory `plot/` with the 1 and 2D marginalized contours.

- This is a very minimal analysis. You can analyze a subset of the chains, produce more/less output and configure it.

  See all the options and their default values running MP with `info --help`.

- You can analyze several folders at a time

  ```
  python montepython/MontePython.py info experiment_1/ experiment_2/ ...
  ```

  This produces combined plots, which is useful to compare models/experiments.

- A very convenient flag is `--extra your_file.plot` to pack the options in a file:

  ```
  #you can operate on parameters
  info.redefine = {'parameters_smg__5': '100*parameters_smg__5-100'}
  #or redefine the names for nicer printing
  info.to_change={'parameters_smg__1':'c_K','parameters_smg__2':'c_B',
                  'parameters_smg__5':'100(M^2-1)'}
  #decide to plot just some of teh parameters
  info.to_plot=['c_K','c_B','100(M^2-1)']
  #configure the plot options
  info.decimal = 2
  info.bins = 20
  info.gaussian_smoothing = 2
  #and many other options...
  ```