CLASS and `hi_class` exercises

Miguel Zumalacarregui

March 16, 2017

The exercises have been ranked by level of difficulty (boring = 🎼, interesting = 🎼🎼, challenging = 🎼🎼🎼) and whether they require `hi_class` (`h_c`).

All exercises can be completed calling CLASS/`hi_class` from *either* the terminal or using Classy (CLASS' python wrapper). Running the code on the terminal is more direct, and should require no additional packages. However, running on python has many advantages, particularly for interactivity and visualization with matplotlib and manipulation of the output. For a premium experience try running Classy on a `jupyter` notebook![1]

Exercise 1: Producing output for a single model (🎼):

This exercise gives an overview of the different outputs that you can expect from CLASS, and is meant mainly for reference: if you get bored please move on to something more fun!

a) Run a default model with (at least) the following parameters

```
#If no cosmo parameters specified CLASS will take its defaults
#Get basically all the ouput.
lensing = yes
non linear = halofit
output = tCl,pCl,lCl,mPk,dTk,vTk # no nCl, can be very slow!
write background = yes
write thermodynamics = yes
write primordial = y
k_output_values = 0.0001, 0.01, 0.1
z_pk = 0., 1., 10
#Information on the execution: higher values -> more info!
input_verbose = 1
background_verbose = 1
thermodynamics_verbose = 1
perturbations_verbose = 1
transfer_verbose = 1
primordial_verbose = 1
spectra_verbose = 1
nonlinear_verbose = 1
lensing_verbose = 1
output_verbose = 1
write warnings = y #let the code tell you if parameters don't make sense
write parameters = yup #for future reference
#root = output/default_ #if you want a certain name for the files
alternative facts = fake #non-existent input -> produce a warning
```

These parameters ensure that you produce a lot of output to play with. In the `output/` directory you should have a bunch of files named as your `.ini` file with different subscripts.
⚠Remember to read `explanatory.ini` for details on these and other options.

[1] See `https://ipython.org/notebook.html`

b) Plot some background quantities. Try `less output/your_model_background.dat` on the terminal (or open the file) to read the output and column numbers in the file's header. Some suggestions:

- Plot the evolution of the energy density of the different components (`rho_g`, `rho_b`, `rho_ur`, `rho_cdm`) relative to the redshift.
- Plot the fractional densities of each component $\Omega_i = \rho_i/$`rho_crit`. In Python you can operate directly on the output arrays:

On Python with CLASS

```
import numpy as np
import matplotlib.pyplot as plt
b = np.loadtxt('../background.dat',
               unpack=True)
plt.plot(b[0],b[8]/[13]) #Omega_g
```

On Python with `classy`

```
import matplotlib.pyplot as plt
#after running 'cosmos'
b = cosmos.get_background()
plt.plot(b['z'],
       b['(.)rho_g']/b['(.)rho_crit'])
```

You can also check that the densities add to H^2 (in CLASS the units are so that the Friedmann equation reads $H^2 = \sum_i \rho_i$), or alternative that the fractional densities add up $\sum_i \Omega_i = 1$.

- Plot the luminosity and angular diameter distances.
- Take a look at the thermal history output: `less output/your_model_thermodynamics.dat`

c) Plot the raw (`less output/your_model_cl.dat`) and lensed (`less output/your_model_cl_lensed.dat`) CMB spectra. For instance:

- Plot the TT, TE and EE unlensed power spectra.
- Compare the lensed and unlensed TT power spectrum.
- Plot the lensing potential (column `phiphi`).

d) Plot the linear power spectrum (`less output/your_model_z*_pk.dat`)

- Compare the power spectrum at different redshifts `z1, z2, z3`
- Compare the linear and non-linear spectra (`less output/your_model_z*_pk_nl.dat`)

e) Study the evolution of perturbations:

- The transfer functions for density and velocity (`less output/your_model_z*_tk.dat`) give a snapshot of the perturbations (at the same redshifts at which you request the power spectrum). Plot the transfer function for baryons at different redshfits.
- You can also look at the time evolution of the perturbations for different scales (`less output/your_model_perturbations_k*_s.dat`). Plot the evolution of baryons for different scales. What regimes can you distinguish?

Exercise 2: Simple moidifications of Gravity (`h_c`, `)`):

`hi_class` puts gravity is in your hands! Here you will explore a simple model in which the Horndeski/EFT functions are proportional to the Dark Energy fractional density: $\alpha_i(z) = c_i \Omega_{\rm smg}(z)$, for $c_K = 1$, variable c_B, $c_M = c_T = 0$ and the same expansion as a ΛCDM universe.

The relevant parameters are

```
Omega_Lambda = 0 # no cosmological constant
Omega_fld = 0 # no perfect fluid DE
Omega_smg = -1 # use closure relation to find the density of modified gravity
gravity_model = propto_omega
              # c_K, c_B, c_M, c_T, M*_ini
parameters_smg = 1., 0.5, 0., 0., 1. #vector with the model parameters
expansion_model = lcdm #model for rho_smg(z)
expansion_smg = 0.5 #vector with expansion parameters (one varied to adjust Omega_smg)
```

Try values of the braiding $c_B = 0, 0.5, 1, 1.5$. The case $c_B = 0$ is practically indistinguishable from Λ-GR, you can take it as your fiducial model.

a) Reproduce the plots of Figure 2 of the `hi_class` paper (`https://arxiv.org/abs/1605.06102`). Check that the oscillations are due to CMB lensing.

b) Plot the time evolution of α_B (column `braiding_smg` in the `_background.dat` file). You can check that $\alpha_B/\Omega_{\rm smg}$ is a constant.

c) The effect of modified gravity on low multipoles is due to the Integrated Sachs Wolfe Effect (ISW). With CLASS you can separate the different components that contribute to the CMB to figure out which where the differences are coming from. For the $c_B = 0, 1.5$ cases compare the results of setting `temperature contributions = tsw`, `temperature contributions = lisw` and `temperature contributions = tsw, lisw`. These flags represent the temperature Sachs-Wolfe effect and the late ISW effect.

Exercise 3: Massive neutrinos (❞)

In this exercise we will compare the predictions of different models and investigate the imprint of neutrino masses on cosmological observables

a) Obtain the output for the background, CMB temperature and matter power spectra for a flat model with one massive neutrino for different values of the mass. Try 0 eV, 0.06, 0.1, and 1 eV, roughly corresponding to no mass, the minimal sum of neutrino masses for the normal and inverted hierarchies, and a large value excluded by current data. The relevant parameters are:

```
N_ur = 2.0328 #get the right number of relativistic species in the early universe
N_ncdm = 1 #number of massive neutrios (= non-cold dark matter in CLASS lingo)
m_ncdm = 0.06 #mass of the massive neutrino in eV
```

b) Plot the evolution of the energy density of the massive neutrino component `rho_ncdm[0]` (use log scale in both axes). When does it become non-relativistic?

c) Plot the CMB and matter power spectra, and the relative deviations with respect to the massless case. The CMB output should have the same ℓ values, but you might need to use interpolation for the matter power spectra (see Appendix A or an interpolation routine)

d) The case of *a single* massive neutrino is not realistic. We know from neutrino oscillation experiments that at least two neutrinos have to be massive (cf. page 50 of the PDG report `http://pdg.lbl.gov/2014/reviews/rpp2014-rev-neutrino-mixing.pdf`).
Repeat the previous exercise for the realistic cases in which the total mass is distributed in a manner compatible with neutrino data:

- Normal Hierarchy: $m_1 \ll m_2 < m_3$ with $m_2 \approx 0.0087$eV, $m_3 \approx 0.050$eV
- Inverted Hierarchy: $m_3 \ll m1 < m_2$ with $m_1 \approx m_2 \approx 0.049$eV

Note that the subindices label neutrino generations. They are important for particle physics, but irrelevant for CLASS.

e) Compare the output of each simple model (one massive neutrino, part 1) with the realistic one (2-3 massive neutrinos, part 3). Plot the relative deviations. Can we detect this differences?

f) Take a look at all the `ncdm` options in `explanatory.ini`. There are lots of things you can do to massive neutrinos, warm DM, etc...

Help: a neat iPython notebook for neutrino plotting `https://github.com/ThomasTram/iCLASS/blob/master/neutrinohierarchy.ipynb`

Exercise 4: Adding a new model in `hi_class` (`h_c`, 👣👣)

`hi_class` has been designed so that adding a new model is nearly trivial. Remember you can search for all instances of a model to see what you have to change (just type `grep propto_omega */*.*` on the terminal). There are just a handful of those.[2]

As an example you will consider a modification of the model in the previous exercise in which the α-functions are proportional to *the n-th power* of the $\Omega_{\rm smg}$:

$$\alpha_i(z) = c_i \Omega^n_{\rm smg}(z)\,. \tag{1}$$

The effective Planck mass M_* is defined by $\alpha_M = d\log(M_*^2)/d\log(a)$ and obtained integrating α_M numerically from an initial value $M_*^2(\tau_{\rm ini})$. The parameters of this model are

$$\left(c_K, c_B, c_M, c_T, M_*^2(\tau_{\rm ini}), n\right)\,, \tag{2}$$

and are specified in that order. We can call the model `propto_omega_n` internally in the code.

a) Declare the model as a case of `gravity_model` at the begining of `include/background.h`. Make sure to run `make clean` before compiling again.

b) In `source/input.c` copy-paste the block for `propto_omega` and adapt (`->` in comments)

```
if (strcmp(string1,"propto_omega") == 0) { /* model name in .ini file -> propto_omega_n */
  pba->gravity_model_smg = propto_omega; /* model name in code -> propto_omega_n */
  pba->field_evolution_smg = _FALSE_; //for non-parameterized models
  pba->M_pl_evolution_smg = _TRUE_; //M_pl from alpha_M or vice versa?
  flag2=_TRUE_; //model is read
  pba->parameters_2_size_smg = 5; /* number of parameters -> 6 */
  class_read_list_of_doubles("parameters_smg",pba->parameters_2_smg,pba->parameters_2_size_smg);
  // You can add tests for your parameters here [search for class_test() below]
}
```

The string `"propto_omega"`in the first line is the model name in the `.ini` file (you can give multiple options). `propto_omega`in `pba->gravity_model_smg` is the internal label.
Bonus: add a `class_test` to make the code stop if $n < 0$ (avoid when α's diverge at early times).

c) In `source/background.c`→`background_initial_conditions(...` copy and paste the block for `propto_omega` and change the condition in the `if (...` statement so that it refers to `propto_omega_n`. If you omit this step then $M_*^2(\tau_{\rm ini}) = 1$ and the 4^{th} parameter will be ignored.

d) In `source/background.c`→`background_gravity_functions(...` copy and paste the block for `propto_omega` and modify it. You need to 1) modify the condition in the `if (...` statement so that it refers to `propto_omega_n` 2) add the new parameter `double n = pba->parameters_2_smg[5];` (recall that the fourth entry is for the initial condition of M_*^2) and 3) modify the expression for the αs by replacing `Omega_smg` with `pow(Omega_smg,n)` to reflect the changes on your model:

```
if (pba->gravity_model_smg == propto_omega_n) { //make sure it refers to the new model
  ...
  double n = pba->parameters_2_smg[5]; // recall 4th param is IC for M_*^2

  pvecback[pba->index_bg_kineticity_smg] = c_k*pow(Omega_smg,n); // form of the alpha's
  ...
}
```

e) In `source/background.c`→`background_gravity_parameters(...` copy and paste the block for `propto_omega` and modify it. This prints the information on the model. It won't affect the results, but it's nice to have.

Now you can compile and check if the code works (it should!). Some things you can do now include:

[2] You can also find *all* the `hi_class`-specific modifications by searching for `smg` (acronym for scalar modified gravity).

– Plot the evolution of the α-functions for different values of n.

– Check that you get the same results (CMB, matter spectra) as in `propto_omega` whenever $n = 1$.

– For fun: see that setting $n \gtrsim 0$ produces very large departures relative to ΛGR, even for small c_i's (this is because modified gravity acts for a longer fraction of the cosmic history).

– Use MontePython (see corresponding notes) to run an MCMC and obtain constraints on n (together with the other parameters). Publish a paper with the results. You are welcome :)

A Some useful routines

Interpolation

In Python

```
#routine to compare relative spectra
#credits Carlos Garcia

import numpy as np
from scipy import interpolate

def rel_diff(x1, y1, x2, y2):
    """ Relative difference between data vectors
    Args:
    x1,y1,x2,y2 (np array): data to interpolate.
    Returns:
    [x,y]: arrays with x values and relative difference.
    """
    data2 = interpolate.interp1d(x2,y2)
    xmin = max(min(x1),min(x2))
    xmax = min(max(x1),max(x2))

    X = x1[x1>=xmin]
    X = x1[x1<=xmax]
    # Use the y-values of the X-array
    b1 = np.where(x1 == X[0])[0][0] #Lowest index
    b2 = np.where(x1 == X[-1])[0][0] #Highest index

    Y = y[b1:b2 + 1]

    diff = data2(X)/Y-1.

    return [X, diff]
```