The Cosmic Linear Anisotropy Solving System (CLASS)

Julien Lesgourgues

February 23, 2015

1 Overall architecture of CLASS

1.1 The seven-module backbone

The purpose of CLASS consists in computing some power spectra for a given set of cosmological parameters. This task can be decomposed in few steps or modules:

1. compute the evolution of cosmological background quantitites.
2. compute the evolution of thermodynamical quantitites (ionization fractions, etc.)
3. compute the evolution of source functions $S(k, \eta)$ (by integrating over all perturbations).
4. compute Bessel functions (in order to go from Fourier to harmonic space).
5. compute transfer functions $\Delta_l(k)$ (unless one needs only Fourier spectra $P(k)$'s and no harmonic spectra C_l's).
6. compute the primordial spectrum for scalars, tensors, etc. (straightforward if the input consists in spectral parameters A_s, n_s, r, ..., but this module will incorporate the option of integrating over inflationary perturbations).
7. compute power spectra C_l's and/or $P(k)$'s.

In CLASS, each of these steps is associated with a structure:

1. `struct background` for cosmological background,
2. `struct thermo` for thermodynamics,
3. `struct perturbs` for source functions,
4. `struct bessels` for bessel functions,

5. `struct transfers` for transfer functions,

6. `struct primordial` for primordial spectra,

7. `struct spectra` for output spectra.

A given structure contains "everything concerning one step that the subsequent steps need to know" (for instance, everything about source functions that the transfer module needs to know). In particular, each structure contains one array of tabulated values (background quantitites as a function of time, thermodynamical quantitites as a function of redshift, sources as a function of (k, η), etc.). It also contains information about the size of this array and the value of the index of each physical quantity, so that the table can be easily read and interpolated. Finally, it contains any derived quantity that other modules might need to know. Hence, the comunication from one module A to another module B consists in passing a pointer to the structure filled by A, and nothing else.

Each structure is defined and filled in one of the following modules (and precisely in the order below):

1. `background.c`

2. `thermodynamics.c`

3. `perturbations.c`

4. `bessel.c`

5. `transfer.c`

6. `primordial.c`

7. `spectra.c`

Each of these modules contains at least three functions:

- *module*`_init(...)`
- *module*`_free()`
- *module_something*`_at_`*somevalue*`(...)`

The first function allocates and fills each structure. This can be done provided that the previous structures in the hierarchy have been already allocated and filled. In summary, calling one of *module*`_init(...)` amounts in solving entirely one of the steps 1 to 7.
The second function deallocates the fields of each structure. This can be done optionally at the end of the code (or, when the code is embedded in a sampler, this *must* be done between each execution of CLASS, and especially before calling *module*`_init(...)` again with different input parameters).
The third function is able to interpolate the pre-computed tables. For instance, `background_init()` fills a table of background quantitites for discrete values of

conformal time η, but `background_at_eta(eta, * values)` will return these values for any arbitrary η.
Note that functions of the type *module_something*`_at_`*somevalue*`(...)` are the only ones which are called from another module, while functions of the type *module*`_init(...)` and *module*`_free()` are the only one called by the main executable. All other functions are for internal use in each module.

1.2 Input

There are two types of input:

1. "precision parameters" (controlling the precision of the output and the execution time),

2. "input parameters" (cosmological parameters, flags telling to the code what it should compute, ...)

All "precision parameters" have been grouped in a single structure `struct precision`. The code contains *no other arbitrary numerical coefficient*. This structure is initialized in a simple module `precision.c` by the function `precision_init()`. Nothing is allocated dynamically in this function, so there is no need for a `precision_free()` function.
Each "input parameter" refers to one particular step in the computation: background, thermodynamics, perturbations, etc. Hence they are defined as part of the corresponding structure. Their values are assigned in a simple module `input.c`, by a function `input_init(...)` which has a pointer towards each structure in its list of arguments. Hence, when a given function *module*`_init(...)` is called, the corresponding structure already contains input parameters; the function fills the rest of this structure. The function `input_init(...)` does not allocate any field dynamically, so there is no need for an `input_free()` function.

1.3 Output

A simple module `output.c` writes the final results in files. The name of the files are considered as input parameters making part of a small structure `struct output`. Like for all other input parameters, these names are assigned inside the function `input_init(...)`. Again this structure contains no dynamically allocated quantitites, so there is no need for an `output_free()` function.

1.4 Summary

We hope that after this short overview, it is clear for the reader that the main executable of CLASS should consist only in the following lines (not including comments and error-management lines):

```
main() {
   struct precision pr;
   struct background ba;
   struct thermo th;
   struct perturbs pt;
   struct bessels bs;
   struct transfers tr;
   struct primordial pm;
   struct spectra sp;
   struct output op;

   precision_init(&pr)
   input_init(&ba,&th,&pt,&bs,&tr,&pm,&sp,&op)
   background_init(&pr,&ba)
   thermodynamics_init(&ba,&pr,&th)
   perturb_init(&ba,&th,&pr,&pt)
   bessel_init(&ba,&pt,&pr,&bs)
   transfer_init(&ba,&th,&pt,&bs,&pr,&tr)
   primordial_init(&pt,&pr,&pm)
   spectra_init(&pt,&tr,&pm,&sp)
   output_init(&pt,&tr,&sp,&op)

   /****** done ******/

   spectra_free()
   primordial_free()
   transfer_free()
   bessel_free()
   perturb_free()
   thermodynamics_free()
   background_free()
}
```

For a given purpose, somebody could only be interested in the intermediate steps (only background quantitites, only the thermodynamics, only the perturbations and sources, etc.) It is then straightforward to truncate the full hierarchy of modules 1, ... 7 at some arbitrary order. We provide several "reduced executables" `test_`*module* achieving precisely this.

Note also that if CLASS is embedded in a parameter sampler and only "fast" parameters are varied (i.e., parameters related to the primordial spectra), then it is only necessary to repeat the following steps after `output_init(...)`:

```
   spectra_free()
   primordial_free()
   input_init(&ba,&th,&pt,&bs,&tr,&pm,&sp,&op)
```

```
primordial_init(&pt,&pr,&pm)
spectra_init(&pt,&tr,&pm,&sp)
output_init(&pt,&tr,&sp,&op)
```

2 General principles

2.1 Flexibility

Explain allocation of indices, ...

2.2 Control of precision

Explain precision structure, ...

2.3 Control of errors