

Input module

Thomas Tram

Institute of Gravitation and Cosmology, Portsmouth

October 27, 2014

How does the input module works?

Three essential functions:

```
input_init_from_arguments()
```

```
input_init()
```

```
input_read_parameters()
```

(1) `input_init_from_arguments(argc,argv,...)`

- reads the input file(s) *.ini, (*.pre)
- identifies known names associated with values, and store them in a structure called `file_content` with fields `name`, `value`, e.g.:

```
struct file_content fc;
fc.name[0] = "h";           fc.value[0] = "0.68";
fc.name[1] = "Omega_b";    fc.value[1] = "0.04";
fc.name[2] = "omega_cdm";  fc.value[2] = "0.12";
fc.name[3] = "modes";      fc.value[3] = "s,t";
...
```

- calls `input_init(&fc,...)`
or equivalently:

```
struct file_content * pfc;
pfc = &fc;
input_init(pfc,...)
```

(2) `input_init(pfc, ...)`

- eventually runs a **shooting algorithm**, as will be explained in two slides.
- calls `input_read_parameters(pfc, ...)`, the function in charge of defining **all input parameters**.

These are located in:

- precision structure (`pr`) for precision parameters
- beginning of each structure (`ba, th, pt, ...`) for cosmological parameters and parameters describing what needs to be computed

Hence the full list of arguments is

```
input_read_parameters(pfc, ppr, pba, pth, ppt, ptr, ppm, psp, pnl, ple, pop, errmsg)
```

(3) `input_read_parameters(pfc,...)`

- initialises all parameters with **default values** with `input_default_params()` and `input_default_precision()`
- overwrites** parameters existing in the file content

Exploits some simple analytical formulae, e.g. $\Omega_i = \omega_i/h^2$ to allow for different input parametrisations.

Not always sufficient: E.g. $100 \times \theta_s$ cannot be converted analytically into h . Other examples:

target parameter	unknown parameter
$100 \times \theta_s$	h
Ω_{dcdm}	$\rho_{\text{dcdm}}^{\text{ini}}$
σ_8	A_s

(3) `input_read_parameters(pfc,...)`

- initialises all parameters with **default values** with `input_default_params()` and `input_default_precision()`
- overwrites** parameters existing in the file content

Exploits some simple analytical formulae, e.g. $\Omega_i = \omega_i/h^2$ to allow for different input parametrisations.

Not always sufficient: E.g. $100 \times \theta_s$ cannot be converted analytically into h . Other examples:

target parameter	unknown parameter
$100 \times \theta_s$	h
Ω_{dcdm}	$\rho_{\text{dcdm}}^{\text{ini}}$
σ_8	A_s

Requires a **shooting method** !

Full sequence when CLASS called with **input files** or from a **wrapper**:

- 1 Call `input_init_from_argument(..)` : **read files and fill** `fc`
- 2 Call `input_init(pfc,...)` :
- 3 check if there are target/unknown parameters
- 4 if yes, **shooting**: Run CLASS iteratively using "unknown parameters" until "target parameters" are reached
- 5 all parameters are known at this point
- 6 call `input_read_parameters(..)`

Full sequence when CLASS called with **input files** or from a **wrapper**:

- 1 Call `some_function(...)` : Create and fill `fc` manually
- 2 Call `input_init(pfc,...)` :
- 3 check if there are target/unknown parameters
- 4 if yes, **shooting**: Run CLASS iteratively using "unknown parameters" until "target parameters" are reached
- 5 all parameters are known at this point
- 6 call `input_read_parameters(..)`

Let's add an input parameter!

The baryon density can be entered in CLASS by specifying either `Omega_b` or `omega_b` in the `.ini`-file. As an alternative, let us add the baryon fraction

$$\eta_b = \frac{n_N}{n_\gamma}. \quad (1)$$

The number density of photons are given by an integral over the Bose-Einstein distribution, so

$$n_\gamma = \frac{2\zeta(3)}{\pi^2} \left(\frac{k_B T_\gamma}{\hbar c} \right)^3, \quad (2)$$

while the density of nucleons is

$$n_N = \frac{\rho_b}{m_N} = \frac{3H_0^2 \Omega_b}{8\pi G m_N} \simeq 1.346 \cdot 10^{-7} \text{K}^3 \Omega_b h^2. \quad (3)$$

Exercise 1e: Adding a new input parameter

Combining these equations give us

$$\Omega_b h^2 = 1.81 \cdot 10^6 \eta_b \left(\frac{T_{\gamma,0}}{K} \right)^3 \quad (4)$$

Now open `input.c` and locate the function `input_read_parameters()`. Scroll down slowly until you find the point where the baryon density is read:

```
/* Omega_0_b (baryons) */
class_call(parser_read_double(pfc, "Omega_b", &param1, &flag1, errmsg),
            errmsg,
            errmsg);
class_call(parser_read_double(pfc, "omega_b", &param2, &flag2, errmsg),
            errmsg,
            errmsg);
class_test(((flag1 == _TRUE_) && (flag2 == _TRUE_)),
            errmsg,
            "In input file, you can only enter one of Omega_b or omega_b, choose one");
if (flag1 == _TRUE_)
    pba->Omega0_b = param1;
if (flag2 == _TRUE_)
    pba->Omega0_b = param2/pba->h/pba->h;
```

Exercise 1e: Adding a new input parameter

```
/* Omega_0_b (baryons) */  
class_call(parser_read_double(pfc, "Omega_b", &param1, &flag1, errmsg),  
           errmsg,  
           errmsg);  
class_call(parser_read_double(pfc, "omega_b", &param2, &flag2, errmsg),  
           errmsg,  
           errmsg);  
  
class_test(((flag1 == _TRUE_) && (flag2 == _TRUE_)),  
           errmsg,  
           "In input file, you can only enter one of Omega_b or omega_b, choose one");  
  
if (flag1 == _TRUE_)  
    pba->Omega0_b = param1;  
if (flag2 == _TRUE_)  
    pba->Omega0_b = param2/pba->h/pba->h;
```

Exercise 1e: Adding a new input parameter

```
/* Omega_0_b (baryons) */  
class_call(parser_read_double(pfc, "Omega_b", &param1, &flag1, errmsg),  
           errmsg,  
           errmsg);  
class_call(parser_read_double(pfc, "omega_b", &param2, &flag2, errmsg),  
           errmsg,  
           errmsg);
```

```
/* Read the baryon fraction eta_b! */  
class_call(parser_read_double(pfc, "eta_b", &param3, &flag3, errmsg),  
           errmsg,  
           errmsg);
```

```
class_test(((flag1 == _TRUE_) && (flag2 == _TRUE_)),  
           errmsg,  
           "In input file, you can only enter one of Omega_b or omega_b, choose one");
```

```
if (flag1 == _TRUE_)  
    pba->Omega0_b = param1;  
if (flag2 == _TRUE_)  
    pba->Omega0_b = param2/pba->h/pba->h;
```

Exercise 1e: Adding a new input parameter

```
/* Omega_0_b (baryons) */  
class_call(parser_read_double(pfc, "Omega_b", &param1, &flag1, errmsg),  
           errmsg,  
           errmsg);  
class_call(parser_read_double(pfc, "omega_b", &param2, &flag2, errmsg),  
           errmsg,  
           errmsg);
```

```
/* Read the baryon fraction eta_b! */  
class_call(parser_read_double(pfc, "eta_b", &param3, &flag3, errmsg),  
           errmsg,  
           errmsg);
```

```
/* Test that at most one flag has been set: */  
class_test(class_at_least_two_of_three(flag1, flag2, flag3),  
           errmsg,  
           "In input file, you can only enter one of eta_b, Omega_b or omega_b, choose  
           one");
```

```
if (flag1 == _TRUE_)  
    pba->Omega0_b = param1;  
if (flag2 == _TRUE_)  
    pba->Omega0_b = param2/pba->h/pba->h;
```

Exercise 1e: Adding a new input parameter

```
/* Omega_0_b (baryons) */  
class_call(parser_read_double(pfc, "Omega_b", &param1, &flag1, errmsg),  
           errmsg,  
           errmsg);  
class_call(parser_read_double(pfc, "omega_b", &param2, &flag2, errmsg),  
           errmsg,  
           errmsg);
```

```
/* Read the baryon fraction eta_b! */  
class_call(parser_read_double(pfc, "eta_b", &param3, &flag3, errmsg),  
           errmsg,  
           errmsg);
```

```
/* Test that at most one flag has been set: */  
class_test(class_at_least_two_of_three(flag1, flag2, flag3),  
           errmsg,  
           "In input file, you can only enter one of eta_b, Omega_b or omega_b, choose  
           one");
```

```
if (flag1 == _TRUE_)  
    pba->Omega0_b = param1;  
if (flag2 == _TRUE_)  
    pba->Omega0_b = param2/pba->h/pba->h;
```

```
/* Set Omega_b in background structure. Formula hardcoded :( */  
if (flag3 == _TRUE_)  
    pba->Omega0_b = 1.81e6*param3*pow(pba->T_cmb, 3)/pba->h/pba->h;
```

Implementation of the shooting method

Inside `input_init()` in `input.c`:

```
/** These two arrays must contain the strings of names to be searched  
    for and the corresponding new parameter */  
char * const target_namestrings[] = {"100*theta_s", "Omega_dcdm", "omega_dcdm",  
                                     "Omega_scf", "Omega_ini_dcdm", "omega_ini_dcdm"};  
char * const unknown_namestrings[] = {"h", "Omega_ini_dcdm", "Omega_ini_dcdm",  
                                       "scf_shooting_parameter", "Omega_dcdm", "  
                                       omega_dcdm"};  
enum computation_stage target_cs[] = {cs_thermodynamics, cs_background, cs_background,  
                                       cs_background, cs_background};
```

And in the include file `input.h`:

```
enum target_names {theta_s, Omega_dcdm, omega_dcdm, Omega_scf, Omega_ini_dcdm,  
                  omega_ini_dcdm};  
enum computation_stage {cs_background, cs_thermodynamics, cs_perturbations,  
                       cs_primordial, cs_nonlinear, cs_transfer, cs_spectra};  
#define _NUM_TARGETS_ 6 //Keep this number as number of target_names
```

Exercise 1f: A parameter needs shooting

Inside `input_try_unknown_parameters()` in `input.c`:

```
for (i=0; i < pfzw->target_size; i++) {
    switch (pfzw->target_name[i]) {
        case theta_s:
            output[i] = 100.*th.rs_rec/th.ra_rec-pfzw->target_value[i];
            break;
```

Inside `input_get_guess()` in `input.c`:

```
/** Here we should right reasonable guesses for the unknown parameters.
    Also estimate dx/dy, i.e. how the unknown parameter responds to the known.
    This can simply be estimated as the derivative of the guess formula.*/

for (index_guess=0; index_guess < pfzw->target_size; index_guess++) {
    switch (pfzw->target_name[index_guess]) {
        case theta_s:
            xguess[index_guess] = 3.54*pow(pfzw->target_value[index_guess],2) -5.455*pfzw->
                target_value[index_guess]+2.548;
            dx/dy[index_guess] = (7.08*pfzw->target_value[index_guess]-5.455);
            /** Update pb to reflect guess */
            ba.h = xguess[index_guess];
            ba.H0 = ba.h * 1.e5 / _c_;
            break;
```


Exercise 1f: A parameter needs shooting

Exercise

Implement σ_8 as an input parameter using the shooting method.