Lecture 8: The background module

Julien Lesgourgues

October 29, 2014

DAY III : Wednesday 29th October

09:30-10:15	CLASS	The background module.	JL
10:15-11:00	CLASS	Playing with the background module.	JL
		Coffee	
11:00-11:45	General	Git repositories.	BA
		Lunch	
13:30-14:15	CLASS	Introducing new physics in the background.	JL
14:15-15:00	MontePython	All running and plotting options.	BA
		Tea	
15:45-16:30	Optional	Lecturers will answer questions	

Homogeneous cosmology

- treated by the module `background.c`. So this lecture will refer mainly to the content of `include/background.h`, `source/background.c`, and to the structure referred as `ba`:

  ```
  struct background ba;
  ```

  with fields `ba.blabla`, or through the pointer `pba`:

  ```
  struct background * pba;
  ```

  with fields `pba->blabla`.

Homogeneous cosmology

- treated by the module `background.c`. So this lecture will refer mainly to the content of `include/background.h`, `source/background.c`, and to the structure referred as `ba`:

  ```
  struct background ba;
  ```

  with fields `ba.blabla`, or through the pointer `pba`:

  ```
  struct background * pba;
  ```

  with fields `pba->blabla`.

- the goal of this module is to solve the background evolution and store the results in a table. It should provide a function able to interpolate within this table at any value of time.

Homogeneous cosmology

- treated by the module `background.c`. So this lecture will refer mainly to the content of `include/background.h`, `source/background.c`, and to the structure referred as `ba`:

```
struct background ba;
```

  with fields `ba.blabla`, or through the pointer `pba`:

```
struct background * pba;
```

  with fields `pba->blabla`.

- the goal of this module is to solve the background evolution and store the results in a table. It should provide a function able to interpolate within this table at any value of time.
- other modules should be able to know all background quantities (densities, pressures, Hubble rate, angular/luminosity distances, etc.) at any given time or redshift.

Units

Units assume $c = 1$ and all quantities in Mpc^n

- times and distances are in Mpc: conformal time τ in Mpc, $H = \frac{a'}{a^2}$ in Mpc^{-1}.

Units

Units assume $c = 1$ and all quantities in Mpc^n

- times and distances are in Mpc: conformal time τ in Mpc, $H = \frac{a'}{a^2}$ in Mpc^{-1}.
- all densities and pressures appearing in the code are in fact some rescaled variables:

$$\rho = \frac{8\pi G}{3}\rho_{\mathrm{physical}}, \qquad p = \frac{8\pi G}{3}p_{\mathrm{physical}}, \qquad \text{in Mpc}^{-2}.$$

Units

Units assume $c = 1$ and all quantities in Mpc^n

- times and distances are in Mpc: conformal time τ in Mpc, $H = \frac{a'}{a^2}$ in Mpc^{-1}.
- all densities and pressures appearing in the code are in fact some rescaled variables:

$$\rho = \frac{8\pi G}{3}\rho_{\rm physical}, \qquad p = \frac{8\pi G}{3}p_{\rm physical}, \qquad \text{in Mpc}^{-2}.$$

So the Friedmann equation reads

$$H = \left(\sum_i \rho_i - \frac{K}{a^2}\right)^{1/2}$$

with the curvature K also in Mpc^{-2}.

Units

Units assume $c = 1$ and all quantities in Mpc^n

- times and distances are in Mpc: conformal time τ in Mpc, $H = \frac{a'}{a^2}$ in Mpc^{-1}.
- all densities and pressures appearing in the code are in fact some rescaled variables:
  $$\rho = \frac{8\pi G}{3}\rho_{\rm physical}, \qquad p = \frac{8\pi G}{3}p_{\rm physical}, \qquad \text{in Mpc}^{-2}.$$

So the Friedmann equation reads

$$H = \left(\sum_i \rho_i - \frac{K}{a^2} \right)^{1/2}$$

with the curvature K also in Mpc^{-2}.

[Note: in perturbation module, k also in Mpc^{-1}. That's it essentially for units!]

The function background_functions()

Most quantities can be immediately inferred from a given value of a without integrating any differential equations:

- $\rho_i = \Omega_i^0 H_0^2 \left(\frac{a}{a_0}\right)^{-3(1+w_i)}$
- $p_i = w_i \rho_i$
- $H = \left(\sum_i \rho_i - \frac{K}{a^2}\right)^{1/2}$
- $H' = \left(-\frac{3}{2}\sum_i(\rho_i + p_i) + \frac{K}{a^2}\right) a$
- $\rho_{\rm crit} = H^2$
- $\Omega_i = \rho_i / \rho_{\rm crit}$

These quantities are all returned by a function `background_functions(pba,a,...)`

What shall we integrate?

But to get them as a function of time we need to integrate one differential equation:

$$a' = a^2 H$$

Then we know $a(\tau)$, and hence all previous quantities as a function of τ.

What shall we integrate?

But to get them as a function of time we need to integrate one differential equation:

$$a' = a^2 H$$

Then we know $a(\tau)$, and hence all previous quantities as a function of τ.
Other quantities requiring an integration over time:

- proper time: $t' = a$ (since $d\tau = dt/a$)

What shall we integrate?

But to get them as a function of time we need to integrate one differential equation:

$$a' = a^2 H$$

Then we know $a(\tau)$, and hence all previous quantities as a function of τ.
Other quantities requiring an integration over time:

- proper time: $t' = a$ (since $d\tau = dt/a$)
- comoving sound horizon: $r_s' = c_s$, since $r_s = \int_{\tau_{\rm ini}}^{\tau_0} c_s d\tau$, with a squared sound speed in the photon+baryon+electron fluid

$$c_s^2 = \frac{\delta p_\gamma}{\delta\rho_\gamma + \delta\rho_b} = \frac{1}{3(1+[3\rho_b/4\rho_\gamma])}.$$

What shall we integrate?

But to get them as a function of time we need to integrate one differential equation:

$$a' = a^2 H$$

Then we know $a(\tau)$, and hence all previous quantities as a function of τ.
Other quantities requiring an integration over time:

- proper time: $t' = a$ (since $d\tau = dt/a$)
- comoving sound horizon: $r_s' = c_s$, since $r_s = \int_{\tau_{\rm ini}}^{\tau_0} c_s d\tau$, with a squared sound speed in the photon+baryon+electron fluid

$$c_s^2 = \frac{\delta p_\gamma}{\delta\rho_\gamma + \delta\rho_b} = \frac{1}{3(1+[3\rho_b/4\rho_\gamma])}.$$

- linear growth factor of density perturbations in minimal ΛCDM model (filled with dust), $D' = 1/(aH^2)$ (such that $\delta_m \propto D$).

How can we classify background variables?

What we said before is model-dependent !!! It is not true that all densities are analytical functions of $\rho_i(a)$ (e.g. with quintessence, scalar-tensor gravity, decaying dark matter, etc.)

How can we classify background variables?

What we said before is model-dependent !!! It is not true that all densities are analytical functions of $\rho_i(a)$ (e.g. with quintessence, scalar-tensor gravity, decaying dark matter, etc.)

Since v2.3, background module written in model-independent way.

How can we classify background variables?

What we said before is model-dependent !!! It is not true that all densities are analytical functions of $\rho_i(a)$ (e.g. with quintessence, scalar-tensor gravity, decaying dark matter, etc.)

Since v2.3, background module written in model-independent way.

In general, three types of parameters:

- $\{A\}$ which can be expressed directly as a function of some variables $\{B\}$.

How can we classify background variables?

What we said before is model-dependent !!! It is not true that all densities are analytical functions of $\rho_i(a)$ (e.g. with quintessence, scalar-tensor gravity, decaying dark matter, etc.)

Since v2.3, background module written in model-independent way.

In general, three types of parameters:

- $\{A\}$ which can be expressed directly as a function of some variables $\{B\}$.
- $\{B\}$, which need to be integrated over time.

How can we classify background variables?

What we said before is model-dependent !!! It is not true that all densities are analytical functions of $\rho_i(a)$ (e.g. with quintessence, scalar-tensor gravity, decaying dark matter, etc.)

Since v2.3, background module written in model-independent way.

In general, three types of parameters:

- $\{A\}$ which can be expressed directly as a function of some variables $\{B\}$.
- $\{B\}$, which need to be integrated over time.
- $\{C\}$, which also need to be integrated but are not used to compute $\{A\}$.

How can we classify background variables?

What we said before is model-dependent !!! It is not true that all densities are analytical functions of $\rho_i(a)$ (e.g. with quintessence, scalar-tensor gravity, decaying dark matter, etc.)

Since v2.3, background module written in model-independent way.

In general, three types of parameters:

- $\{A\}$ which can be expressed directly as a function of some variables $\{B\}$.
- $\{B\}$, which need to be integrated over time.
- $\{C\}$, which also need to be integrated but are not used to compute $\{A\}$.

ΛCDM and many simple extensions:

- $\{A\} = \{\rho_i, p_i, H, ...,\}$
- $\{B\} = \{a\}$
- $\{C\} = \{t, r_s, D\}$

How can we classify background variables?

What we said before is model-dependent !!! It is not true that all densities are analytical functions of $\rho_i(a)$ (e.g. with quintessence, scalar-tensor gravity, decaying dark matter, etc.)

Since v2.3, background module written in model-independent way.

In general, three types of parameters:

- $\{A\}$ which can be expressed directly as a function of some variables $\{B\}$.
- $\{B\}$, which need to be integrated over time.
- $\{C\}$, which also need to be integrated but are not used to compute $\{A\}$.

ΛCDM and many simple extensions:

- $\{A\} = \{\rho_i, p_i, H, ...,\}$
- $\{B\} = \{a\}$
- $\{C\} = \{t, r_s, D\}$

Exemple of extended cosmology with quintessence ϕ (see Thomas's lecture):

- $\{A\} = \{\rho_i, p_i, H, ..., V_\phi, \rho_\phi, p_\phi\}$
- $\{B\} = \{a, \phi, \phi'\}$

How can we classify background variables?

What we said before is model-dependent !!! It is not true that all densities are analytical functions of $\rho_i(a)$ (e.g. with quintessence, scalar-tensor gravity, decaying dark matter, etc.)

Since v2.3, background module written in model-independent way.

In general, three types of parameters:

- $\{A\}$ which can be expressed directly as a function of some variables $\{B\}$.
- $\{B\}$, which need to be integrated over time.
- $\{C\}$, which also need to be integrated but are not used to compute $\{A\}$.

ΛCDM and many simple extensions:

- $\{A\} = \{\rho_i, p_i, H, ...,\}$
- $\{B\} = \{a\}$
- $\{C\} = \{t, r_s, D\}$

Exemple of extended cosmology with quintessence ϕ (see Thomas's lecture):

- $\{A\} = \{\rho_i, p_i, H, ..., V_\phi, \rho_\phi, p_\phi\}$
- $\{B\} = \{a, \phi, \phi'\}$

Exemple of decaying dark matter with non-trivial differential equation giving $\rho_{dm}(t)$:

- $\{A\} = \{\rho_i, p_i, H, ...\}$
- $\{B\} = \{a, \rho_{dm}\}$

How can we classify background variables?

Reflected by arguments of

```
background_functions(pba, pvecback_B, format,  pvecback)
```

Have a look at comments at beginning of `source/background.c` and at `background_functions(..)`

Input background parameters

In the *.ini file, the user may pass:

- Hubble: h or H0 or 100*theta_s
- Photons: T_cmb or Omega_g or omega_g
- Baryons: Omega_b or omega_b
- Ultra-relativvistic relics: N_ur or Omega_ur or omega_ur
- CDM: Omega_cdm or omega_cdm
- Non-cold DM: ncdm : lots of possible input, see dedicated lectures
- Decaying CDM plus its relat. decay product: Omega_dcdmdr or omega_dcdmdr and Gamma_dcdm
- Curvature: Omega_k
- Cosmological constant: Omega_Lambda
- Fluid: Omega_fld, w0_fld, wa_fld, cs2_fld (assuming CLP: $w = w_0 + w_a(1 - a/a_0)$ and $\delta p = c_s^2 \delta\rho$)

[Note: quintessence models implemented in 2.4 but not advertised before 2.5 (still being cross-checked).]

Input background parameters

Just one convention to remember !!!!

One of `Omega_Lambda` or `Omega_fld` must be left unspecified, to let the code match with H_0:

$$H_0^2 = \sum_i \rho_i^0 - K/a_0^2$$

If the two are passed, there is an error message.

All details on the syntax and on these rules are explicitly written in the comments of `explanatory.ini`

Remark on the component called "fluid"

Remark on the fluid:

- $\rho_i = \Omega_i^0 H_0^2 \left(\frac{a}{a_0}\right)^{-3(1+w_i)}$ is only valid when $w_i = \mathrm{constant}$.
- for $w = w_0 + w_a(1 - a/a_0)$, an analytic integration of the energy conservation equation gives

$$\rho_i = \Omega_i^0 H_0^2 \left(\frac{a}{a_0}\right)^{-3(1+w_0+w_a)} e^{3w_a(a/a_0-1)}$$

Remark on the fluid:

- $\rho_i = \Omega_i^0 H_0^2 \left(\frac{a}{a_0}\right)^{-3(1+w_i)}$ is only valid when $w_i = \mathrm{constant}$.
- for $w = w_0 + w_a(1 - a/a_0)$, an analytic integration of the energy conservation equation gives

$$\rho_i = \Omega_i^0 H_0^2 \left(\frac{a}{a_0}\right)^{-3(1+w_0+w_a)} e^{3w_a(a/a_0-1)}$$

Finally there is a parameter `background_verbose=...` (one verbose parameter for each module). 0 gives no output at all, 1 the standard output that you see by default, etc.

External function in the background module

- `background_at_tau(pba, tau,...)` interpolates inside this table and returns $\{A(\tau)\}$, $\{B(\tau)\}$, $\{C(\tau)\}$.

External function in the background module

- `background_at_tau(pba, tau,...)` interpolates inside this table and returns $\{A(\tau)\}$, $\{B(\tau)\}$, $\{C(\tau)\}$.
- `background_tau_of_z(pba, z, &tau)` returns $\tau(z)$, can be useful just before calling `background_at_tau(pba, tau,...)`

Getting background quantities from outside

Calling background quantitites from another module is then very simple: e.g., in the perturbation module:

```
double * pvecback;
class_alloc(pvecback,
            pba->bg_size*sizeof(double),
            ppt->error_message);
...
class_call(background_at_tau(pba,tau,...,...,...,pvecback),
                                  pba->error_message,
                                  ppt->error_message);

/* We want here to compute the total background density*/
if (pba->has_cdm == _TRUE_) {
   rho_tot += pvecback[pba->index_bg_rho_cdm];
}
if (pba->has_fld == _TRUE_) {
   rho_tot += pvecback[pba->index_bg_rho_fld];
   p_tot += pvecback[pba->index_bg_p_fld];
}
...
```

Getting background quantities from outside

Calling background quantitites from another module is then very simple: e.g., in the perturbation module:

```
double * pvecback;
class_alloc(pvecback,
            pba->bg_size*sizeof(double),
            ppt->error_message);
...
class_call(background_at_tau(pba,tau,...,...,...,pvecback),
                              pba->error_message,
                              ppt->error_message);

/* We want here to compute the total background density*/
if (pba->has_cdm == _TRUE_) {
   rho_tot += pvecback[pba->index_bg_rho_cdm];
}
if (pba->has_fld == _TRUE_) {
   rho_tot += pvecback[pba->index_bg_rho_fld];
   p_tot += pvecback[pba->index_bg_p_fld];
}
...
```

3rd argument can be: `pba->long_info`, `pba->normal_info` or `pba->short_info`

Full list of coded background quantities

Currently, the list of all available background quantities is:

short_info	index_bg_a	a
"	index_bg_H	H
"	index_bg_H_prime	H'
normal_info	index_bg_rho_<i>	ρ for _b, _g,_cdm,_ur,_fld,_lambda
"	variables for ncdm	see dedicated lecture
"	index_bg_Omega_r	$\Omega_{\rm radiation}$
long_info	index_bg_rho_crit	$\rho_{\rm crit}$
"	index_bg_Omega_m	$\Omega_{\rm matter}$
"	index_bg_conf_distance	$\tau_0 - \tau = \chi$
"	index_bg_ang_distance	$d_A = a\, r$
"	index_bg_lum_distance	$d_L = (1+z)^2 d_A$
"	index_bg_time	proper time t
"	index_bg_rs	conformal sound horizon r_s
"	index_bg_D	density growth factor of ΛCDM
"	index_bg_f	velocity growth factor of ΛCDM

(with $r = \chi$, or $\sin(\sqrt{K}\chi)/\sqrt{K}$, or $\sinh(\sqrt{-K}\chi)/\sqrt{-K}$)

Dynamical indices in background module

In the background module, two lists of dynamically allocated indices:

- for all variables in the table ($\{A\}$, $\{B\}$, $\{C\}$),

```
int index_bg_a;
...
int bg_size;
```

declared in include/background.h inside the background structure, to be used in other modules.

Dynamical indices in background module

In the background module, two lists of dynamically allocated indices:

- for all variables in the table ($\{A\}$, $\{B\}$, $\{C\}$),

  ```
  int index_bg_a;
  ...
  int bg_size;
  ```

  declared in include/background.h inside the background structure, to be used in other modules.

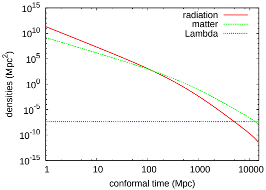
- for the ODE dy[i] = f(y[j]), i.e. for variables $\{B\}$, $\{C\}$,

  ```
  int index_bi_a;
  int index_bi_time;
  int index_bi_rs;
  int index_bi_growth;
  int bi_size;
  ```

  declared in include/background.h outside the background structure, and erased/forgotten after the execution of background_init().

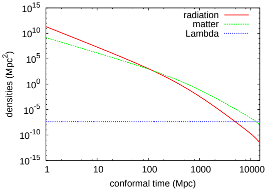
Printing the background evolution

Three ways to produce such a plot with the quantites of your choice (maybe customised to your own model):



Printing the background evolution

Three ways to produce such a plot with the quantites of your choice (maybe customised to your own model):

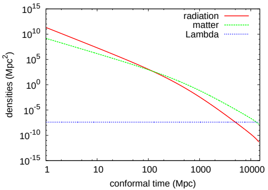


1. easiest: execute e.g. `./class myinput.ini` including in the input file:

```
write background = yes
root = output/toto_
```

Printing the background evolution

Three ways to produce such a plot with the quantites of your choice (maybe customised to your own model):



1. easiest: execute e.g. `./class myinput.ini` including in the input file:

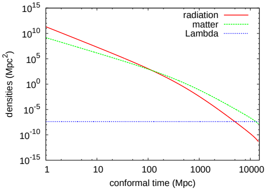
```
write background = yes
root = output/toto_
```

The output module will also write a file `output/toto_background.dat` with header:

```
# Table of selected background quantitites
# All densities are mutiplied by (8piG/3)
# z, proper time [Gyr], conformal time * c [Mpc], H/c [1/Mpc]
```
(etc.)

Printing the background evolution

Three ways to produce such a plot with the quantitates of your choice (maybe customised to your own model):

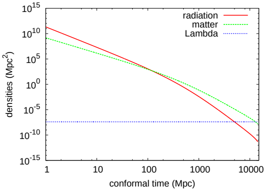


2. smartest: from python, using `classy.pyx`

```
model = Class()
model.set({...})
model.compute()
background = model.get_background()
...
```

Printing the background evolution

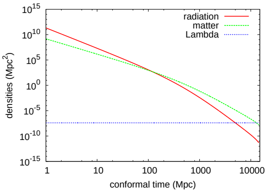
Three ways to produce such a plot with the quantitates of your choice (maybe customised to your own model):



3. good to know: directory `test/` contains several test codes executing only part of the `main(...)` function.

Printing the background evolution

Three ways to produce such a plot with the quantitates of your choice (maybe customised to your own model):



3. good to know: directory `test/` contains several test codes executing only part of the `main(...)` function.

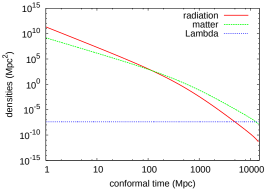
For instance: `test/test_background.c` only executes `input_init(...)`, `background_init(...)`, and then outputs a table of all background quantities. Useful also for quick debugging!

Usage:

```
> make test_background
> ./test_background myinput.ini
```

Printing the background evolution

Three ways to produce such a plot with the quantitates of your choice (maybe customised to your own model):



3. good to know: directory `test/` contains several test codes executing only part of the `main(...)` function.

For instance: `test/test_background.c` only executes `input_init(...)`, `background_init(...)`, and then outputs a table of all background quantities. Useful also for quick debugging!

Usage:

```
> make test_background
> ./test_background myinput.ini
```

Same with `test_thermodynamics.c`, `test_perturbations.c`, `test_nonlinear.c`, `test_transfer.c`...

Background exercises (see exercise sheet for more details)

Exercise IIa

Reproduce this plot from the Dodelson book *Modern Cosmology*, using the plotting software of your choice.

36 THE SMOOTH, EXPANDING UNIVERSE

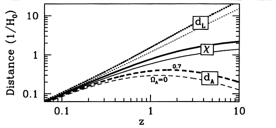


Figure 2.3. Three distance measures in a flat expanding universe. From top to bottom, the luminosity distance, the comoving distance, and the angular diameter distance. The pair of lines in each case is for a flat universe with matter only (light curves) and 70% cosmological constant Λ (heavy curves). In a Λ-dominated universe, distances out to fixed redshift are larger than in a matter-dominated universe.

Exercise IIb

Add a new matter species with an equation of state $p = w\rho$. Visualise its evolution with time.