CLASS

Cosmological Linear Anisotropy Solving System

Markus Mosbech
Institute for Theoretical Particle Physics and Cosmology, RWTH Aachen University

Les Karellis, France, 17-30 Aug 2025

Visit http://class-code.net/ for more info!

class in Les Karellis

What to expect in this first lecture:

•	Basics:	Why use class?
•	Usage:	Installation
•	Usage:	Python Interface
•	Basics:	Existing Species
•	Basics:	Module Overview

We will learn how to use class and which models can be run with it.

What is an Einstein-Boltzmann solver?

Often just called a *Boltzmann code* for brevity, a typical Boltzmann code will:

- Solve coupled Einstein and Boltzmann equations.
- Generally work at linear level in perturbation theory.
- Compute global (Background+Themodynamic) quantities *and* perturbations.

$$\underbrace{G_{\mu\nu} = 8\pi T_{\mu\nu}}_{\text{Einstein-equation}} \qquad \underbrace{\frac{\mathrm{d}f}{\mathrm{d}\lambda} = C[f]}_{\text{Boltzmann-equation}} \tag{1}$$

Why use a Boltzmann code?

Modern Boltzmann codes offer:

- History of the universe at the global level ($H(z)$, $\rho_i(z)$, etc.)

Why use a Boltzmann code?

Modern Boltzmann codes offer:

- History of the universe at the global level ($H(z)$, $\rho_i(z)$, etc.)
- Thermal history of the universe ($T_b(z)$, x_e, τ, etc.)

Why use a Boltzmann code?

Modern Boltzmann codes offer:

- History of the universe at the global level ($H(z)$, $\rho_i(z)$, etc.)
- Thermal history of the universe ($T_b(z)$, x_e, τ, etc.)
- Evolution of (linear) perturbations (δ_i, θ_i, ψ, ϕ, etc.)

Why use a Boltzmann code?

Modern Boltzmann codes offer:

- History of the universe at the global level ($H(z)$, $\rho_i(z)$, etc.)
- Thermal history of the universe ($T_b(z)$, x_e, τ, etc.)
- Evolution of (linear) perturbations (δ_i, θ_i, ψ, ϕ, etc.)
- Fourier space transfer functions ($T(k)$)

Why use a Boltzmann code?

Modern Boltzmann codes offer:

- History of the universe at the global level ($H(z)$, $\rho_i(z)$, etc.)
- Thermal history of the universe ($T_b(z)$, x_e, τ, etc.)
- Evolution of (linear) perturbations (δ_i, θ_i, ψ, ϕ, etc.)
- Fourier space transfer functions ($T(k)$)
- CMB spectra, both lensed and unlensed (C_ℓ^{TT},C_ℓ^{TE},C_ℓ^{EE},C_ℓ^{BB})

Why use a Boltzmann code?

Modern Boltzmann codes offer:

- History of the universe at the global level ($H(z)$, $\rho_i(z)$, etc.)
- Thermal history of the universe ($T_b(z)$, x_e, τ, etc.)
- Evolution of (linear) perturbations (δ_i, θ_i, ψ, ϕ, etc.)
- Fourier space transfer functions ($T(k)$)
- CMB spectra, both lensed and unlensed (C_ℓ^{TT},C_ℓ^{TE},C_ℓ^{EE},C_ℓ^{BB})
- Linear matter power spectrum, galaxy counts, cosmic shear ($\xi^\pm$,C_ℓ^{dd},$P_{\rm lin}(k)$)

Why use a Boltzmann code?

Modern Boltzmann codes offer:

- History of the universe at the global level ($H(z)$, $\rho_i(z)$, etc.)
- Thermal history of the universe ($T_b(z)$, x_e, τ, etc.)
- Evolution of (linear) perturbations (δ_i, θ_i, ψ, ϕ, etc.)
- Fourier space transfer functions ($T(k)$)
- CMB spectra, both lensed and unlensed (C_ℓ^{TT},C_ℓ^{TE},C_ℓ^{EE},C_ℓ^{BB})
- Linear matter power spectrum, galaxy counts, cosmic shear ($\xi^{\pm}$,C_ℓ^{dd},$P_{\rm lin}(k)$)
- Emulated non-linear power spectra

Why use a Boltzmann code?

Modern Boltzmann codes offer:

- History of the universe at the global level ($H(z)$, $\rho_i(z)$, etc.)
- Thermal history of the universe ($T_b(z)$, x_e, τ, etc.)
- Evolution of (linear) perturbations (δ_i, θ_i, ψ, ϕ, etc.)
- Fourier space transfer functions ($T(k)$)
- CMB spectra, both lensed and unlensed (C_ℓ^{TT},C_ℓ^{TE},C_ℓ^{EE},C_ℓ^{BB})
- Linear matter power spectrum, galaxy counts, cosmic shear ($\xi^\pm$,C_ℓ^{dd},$P_{\rm lin}(k)$)
- Emulated non-linear power spectra
- CMB spectral distortions

Why use a Boltzmann code?

Modern Boltzmann codes offer:

- History of the universe at the global level ($H(z)$, $\rho_i(z)$, etc.)
- Thermal history of the universe ($T_b(z)$, x_e, τ, etc.)
- Evolution of (linear) perturbations (δ_i, θ_i, ψ, ϕ, etc.)
- Fourier space transfer functions ($T(k)$)
- CMB spectra, both lensed and unlensed (C_ℓ^{TT},C_ℓ^{TE},C_ℓ^{EE},C_ℓ^{BB})
- Linear matter power spectrum, galaxy counts, cosmic shear ($\xi^{\pm}$,C_ℓ^{dd},$P_{\rm lin}(k)$)
- Emulated non-linear power spectra
- CMB spectral distortions

All computed in a matter of seconds!

Why use a Boltzmann code?

This has several use cases:

- Analysis of CMB experiments
- Analysis of LSS experiments
- Initial conditions for non-linear simulations (N-body, etc.)
- Consistent treatment of background/thermodynamic evolution

All easy to to with `class`!

Fast execution $\Rightarrow$ ideal for use in an MCMC pipeline.

Why use `class`?

`class` is:

- Accurate: `class` & `camb` cross-check each other
- Versatile: Interfaces with `MontePython`, `Cobaya`, `Cosmosis`, `Procoli`, `CosmoPower`, `OLÉ`, `CONNECT`, and others!
- Comprehensive: Computes a wide range of cosmological observables for a large selection of models beyond ΛCDM.
- Modular and well-documented: ReadTheDocs page and Doxygen documentation, thoroughly commented source code, easy to modify

All strong arguments to use `class`!

Installing class

Using class

If you have no intention of modifying source code:

```
> pip install classy
```

And the class wrapper will be ready to use in your Python environment.

This is the easiest way to install and ideal if you only plan to call class via the Python wrapper.

Modifying class

If you wish to modify source code:

```
> git clone git@github.com:
    lesgourg/class_public.
    git class
> cd class/
> make clean; make -j
```

The wrapper can be used in your Python environment, and the binary executable can be called from the terminal.

Documentation

`class` is documented in various places:

- The file `explanatory.ini` lists all input parameters.
  (downloadable from GitHub page if `classy` installed via pip)

Documentation

`class` is documented in various places:

- The file `explanatory.ini` lists all input parameters. (downloadable from GitHub page if `classy` installed via pip)
- **NEW!** Documentation of the `classy` wrapper at `https://class-code.readthedocs.io`.

Documentation

`class` is documented in various places:

- The file `explanatory.ini` lists all input parameters.
  (downloadable from GitHub page if `classy` installed via pip)
- **NEW!** Documentation of the `classy` wrapper at
  `https://class-code.readthedocs.io`.
- **Updated!** Python docstrings in the `classy` wrapper.
- `class` manual in repository

Documentation

`class` is documented in various places:

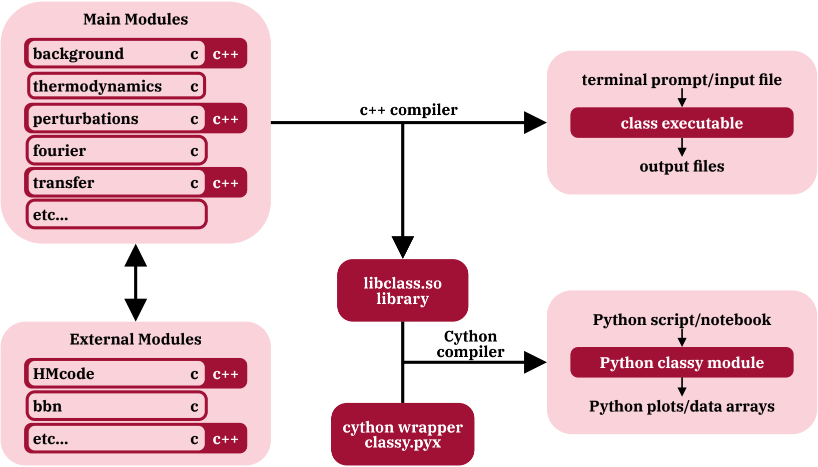
- The file `explanatory.ini` lists all input parameters. (downloadable from GitHub page if `classy` installed via pip)
- **NEW!** Documentation of the `classy` wrapper at `https://class-code.readthedocs.io`.
- **Updated!** Python docstrings in the `classy` wrapper.
- `class` manual in repository
- Online documentation at `https://github.com/lesgourg/class_public/wiki`

Documentation

class is documented in various places:

- The file explanatory.ini lists all input parameters. (downloadable from GitHub page if classy installed via pip)
- **NEW!** Documentation of the classy wrapper at https://class-code.readthedocs.io.
- **Updated!** Python docstrings in the classy wrapper.
- class manual in repository
- Online documentation at https://github.com/lesgourg/class_public/wiki
- Old course notes linked on http://class-code.net and https://schoeneberg.github.io/

The code structure



`class` vs `classy`

Let's clarify a bit of nomenclature!

- `class` is the `C` code that does all the work of solving the EBS. It is used from the command line.
- `classy` is the `Python` wrapper of `class`, and internally uses it. It is used from a python interpreter.

The two share same input/output, except different naming and format (e.g. `.ini` file vs `python dictionary` as input)

Running class in terminal

Run any input file with extension $\star$.ini:

- Simple first-usage file

```
./class default.ini
```

- Huge reference file containing all possible input parameters with comments

```
./class explanatory.ini
```

- Slim file matching Planck 2018 ”baseline model” bestfit

```
./class base_2018_plikHM_TTTEEE_lowl_lowE_lensing.ini
```

Running class in terminal

Run any input file with extension `*.ini`:

- Simple first-usage file

  ```
  ./class default.ini
  ```

- Huge reference file containing all possible input parameters with comments

  ```
  ./class explanatory.ini
  ```

- Slim file matching Planck 2018 ”baseline model” bestfit

  ```
  ./class base_2018_plikHM_TTTEEE_lowl_lowE_lensing.ini
  ```

- All input is presented in detail in `explanatory.ini` (apart from precision parameters)
- This is a *reference* file; we advise you to not modify it:
  - either start from a slim file (like `default.ini`),
  - or copy it and reduce it to a shorter and more friendly file,
  - or write your own from scratch with only needed input lines.

class input parameters

The common 'language' for input is as follows

```
parameter = value
```

with the python dictionary equivalent of

```
{'parameter':'value'}
```

where value is passed as a `python string`.

class input parameters

The common 'language' for input is as follows

```
parameter = value
```

with the python dictionary equivalent of

```
{'parameter':'value'}
```

where value is passed as a `python string`.

Special cases include

```
option = yes/no
selection = a,b,c,d
```

and comments are

```
parameter = value #comment behind parameter
#comment in its own line
```

class vs classy

Let's compare `class` vs `classy` for an execution:

`class`	`classy`
`.ini` file with `parameter = value`	`python dictionary` with `'parameter':'value'`
Solving of equations in `class`	
Output to files Options write files	Output from python functions Options enable functions (otherwise error message)

`./class myfile.ini`	`import classy`
	`cosmo = classy.Class()`
	`cosmo.set(mydictionary)`

Input parameters

If nothing given $\rightarrow$ Planck 2013 cosmology with $\sum m_\nu = 0$

Any parameter overwrites the defaults (from `.ini file` or `cosmo.set`)

Precision file `cl_ref.pre` $\rightarrow$ close to ‘optimal’ precision

Input parameters

Most common/important parameters:

- $\texttt{output} = \underbrace{\texttt{tCl}}_{\texttt{Temperature}\ C_\ell^{TT}}, \underbrace{\texttt{pCl}}_{\texttt{Polarization}\ C_\ell^{TE/EE/BB}}, \underbrace{\texttt{lCl}}_{\texttt{Lensing}\ C_\ell^{\phi\phi}},$
  $\underbrace{\texttt{mPk}}_{\texttt{Matter power}\ P(k,z)}, \underbrace{\texttt{mTk}}_{\texttt{Matter transfer}\ \delta_i(k,z)}, \underbrace{\texttt{vTk}}_{\texttt{Velocity transfer}\ \theta_i(k,z)},$
  $\underbrace{\texttt{nCl}}_{\texttt{Number count}\ C_\ell^{dd}}, \underbrace{\texttt{sCl}}_{\texttt{Galaxy Lensing count}\ C_\ell^{ss}}$

Input parameters

Most common/important parameters:

- `l_max_scalars = 2500`
- `P_k_max_1/Mpc = 1`
- `z_pk = 0, 1, 2` for `class`
- `z_max_pk = 10` for `classy`
- `format = CAMB` for initial conditions
- `write_warnings = yes` for `class`

Input parameters

Most common/important parameters:

- `input_verbose = 1`
- `background_verbose = 2`
- `thermodynamics_verbose = 1`
- `perturbations_verbose = 1`
- `fourier_verbose = 1`
- `output_verbose = 1`

Verbosity parameters for `class`

Input parameters

The ΛCDM parameters $\{H_0, A_s, n_s, \Omega_m, \Omega_b h^2, \tau_{\text{reio}}\}$:

- Hubble constant
  ```
  H_0=67
  h=0.67
  100*theta_s = 1.042
  ```

Input parameters

The ΛCDM parameters $\{H_0, A_s, n_s, \Omega_m, \Omega_b h^2, \tau_{\rm reio}\}$:

- Hubble constant
  ```
  H_0=67
  h=0.67
  100*theta_s = 1.042
  ```
- Primordial amplitude
  ```
  A_s=2.1e-9
  ln10^{10}A_s = 3.0
  ln_A_s_1e10 = 3.0
  sigma8 = 0.825
  ```
  $=\ln(10^{10}A_s)$

  $=\int \frac{k^3 P(k,z=0)}{2\pi^2} T^2(kR_8)\,\mathrm{d}\ln k$

Input parameters

The ΛCDM parameters $\{H_0, A_s, n_s, \Omega_m, \Omega_b h^2, \tau_{\rm reio}\}$:

- Hubble constant
  `H_0=67`
  `h=0.67`
  `100*theta_s = 1.042`
- Primordial amplitude
  `A_s=2.1e-9`
  `ln10^{10}A_s = 3.0` $=\ln(10^{10}A_s)$
  `ln_A_s_1e10 = 3.0`
  `sigma8 = 0.825` $=\int \frac{k^3 P(k,z=0)}{2\pi^2} T^2(kR_8)\,\mathrm{d}\ln k$
- Primordial tilt
  `n_s=0.96`

Input parameters

The ΛCDM parameters $\{H_0, A_s, n_s, \Omega_m, \Omega_b h^2, \tau_{\rm reio}\}$:

- Hubble constant
  `H_0=67`
  `h=0.67`
  `100*theta_s = 1.042`
- Primordial amplitude
  `A_s=2.1e-9`
  `ln10^{10}A_s = 3.0` $=\ln(10^{10}A_s)$
  `ln_A_s_1e10 = 3.0`
  `sigma8 = 0.825` $=\int \frac{k^3 P(k,z=0)}{2\pi^2} T^2(kR_8) \mathrm{d}\ln k$
- Primordial tilt
  `n_s=0.96`
- Matter abundance
  `Omega_m=0.3`
  `omega_m=0.14` $=\Omega_m h^2$
  `Omega_cdm=0.25`
  `Omega_cdm=0.11` $=\Omega_{\rm cdm} h^2$

Input parameters

The ΛCDM parameters $\{H_0, A_s, n_s, \Omega_m, \Omega_b h^2, \tau_{\rm reio}\}$:

- Hubble constant
  `H_0=67`
  `h=0.67`
  `100*theta_s = 1.042`
- Primordial amplitude
  `A_s=2.1e-9`
  `ln10^{10}A_s = 3.0` $=\ln(10^{10}A_s)$
  `ln_A_s_1e10 = 3.0`
  `sigma8 = 0.825` $=\int \frac{k^3 P(k,z=0)}{2\pi^2} T^2(kR_8)\mathrm{d}\ln k$
- Primordial tilt
  `n_s=0.96`
- Matter abundance
  `Omega_m=0.3`
  `omega_m=0.14` $=\Omega_m h^2$
  `Omega_cdm=0.25`
  `Omega_cdm=0.11` $=\Omega_{\rm cdm} h^2$
- Baryon abundance
  `Omega_b=0.05`
  `omega_m=0.02233` $=\Omega_b h^2$
- Reionization time
  `z_reio=7.0`
  `tau_reio=0.05` $\approx \int_0^{z_{\rm reio}} \sigma_T n_e \mathrm{d}\eta$

Input parameters

One-parameter extensions

- Curvature
  `Omega_k = 0`
- Dark radiation
  `N_ur = 3.044` $\hat{=} N_{\mathrm{eff}}$
- CMB temperature
  `T_cmb = 2.7255`
- Lensing enhancement
  `A_L = 1`
- Primordial running
  `alpha_s = 0`

Input parameters

Dark Energy options

- Cosmological constant
  `Omega_Lambda`
- Fluid
  `Omega_fld`
- Scalar field
  `Omega_scf`

Input parameters

Dark Energy options

- Cosmological constant
  `Omega_Lambda`
- Fluid
  `Omega_fld`
- Scalar field
  `Omega_scf`

Universe is always filled (to be as curved as Ω_k) by putting whatever dark energy is still available (If you want to fill with scalar field, you need `Omega_scf<0`).
Using the Budget equation

$$\sum \Omega_i = 1 + \Omega_k \tag{2}$$

Input parameters

Dark Energy options

- Cosmological constant
  `Omega_Lambda`
- Fluid
  `Omega_fld`
- Scalar field
  `Omega_scf`

Universe is always filled (to be as curved as Ω_k) by putting whatever dark energy is still available (If you want to fill with scalar field, you need `Omega_scf<0`).
Using the Budget equation

$$\sum \Omega_i = 1 + \Omega_k \tag{2}$$

Examples:

`Omega_m=0.3`	$\Rightarrow$	`Omega_Lambda -> 0.7`
`Omega_fld=0.5,Omega_m=0.3`	$\Rightarrow$	`Omega_Lambda -> 0.2`
`Omega_Lambda=0,Omega_m=0.3`	$\Rightarrow$	`Omega_fld -> 0.7`
`Omega_Lambda=0,Omega_fld=0,` `Omega_scf=-1,Omega_m=0.3`	$\Rightarrow$	`Omega_scf -> 0.7`
`Omega_k=0.2,Omega_m=0.3`	$\Rightarrow$	`Omega_Lambda -> 0.9`

Input parameters

Neutrino mass

- `N_ncdm=1` (single massive neutrino)
- `m_ncdm=0.06` (in eV)
- `N_ur = 3.044 - 1.0132 N_ncdm = 2.0308`
  (each massive neutrino contributes 1.0132 to N_{eff} [QED corrections, could be optimized in future])
  or
- `Neff = 3.044` will automatically adjust `N_ur` to achieve the desired N_{eff}

Input parameters

Neutrino mass

- `N_ncdm=3` (three massive neutrinos)
- `m_ncdm=0,0.0495,0.0582` (inverted hierarchy)
- `N_ur = 3.044 - 1.0132 N_ncdm = 0.0044`
  or
- `Neff = 3.044` will automatically adjust `N_ur` to achieve the desired N_{eff}

Input parameters

Neutrino mass

- `N_ncdm=1` (one *degenerate* massive neutrinos)
- `deg_ncdm=3` (triply degenerate)
- `m_ncdm=0.02` (total mass 0.06eV)
- `N_ur = 3.044 - 1.0132 N_ncdm * deg_ncdm = 0.0044`
  or
- `Neff = 3.044` will automatically adjust `N_ur` to achieve the desired N_{eff}

Plotting

You can get plots

1. Manually: using the output files with e.g. `gnuplot`, `IDL`, `python`, `Mathematic`, `GNU Octave`...
2. Automatically: using `python` and script `CPU.py`, or `MATLAB` and script `plot_CLASS_output.m`
3. Interactively: using `class` as a `python` module, within a `python` session or a `Jupyter Notebook`

Running class from Python

class as a Python module

- based on wrapper located in `python/classy.pyx` (developed initially by B. Audren and extended by many others)
- the compilation produces a python module `classy.py` and installs it on your computer (can be called from anywhere)
- wrapper written in Cython, encapsulates most useful class variables/functions, contains extra functions (e.g. MontePython-motivated)
- (project: get most of the wrapper generated automatically from C code at compilation - Coming soon!)
- goal: obtain, manipulate and plot the results directly within (i)python scripts or notebooks (recommended)

Existing content already in `class`

ΛCDM:

- Baryons `Omega_b`$= \Omega_b$ or `omega_b`$= \Omega_b h^2$

Existing content already in `class`

ΛCDM:

- Baryons `Omega_b`$= \Omega_b$ or `omega_b`$= \Omega_b h^2$
- Cold Dark matter `Omega_cdm`$= \Omega_{\rm cdm}$ or `omega_cdm`$= \Omega_{\rm cdm} h^2$ or `Omega_m`$= \Omega_{\rm m}$ or `omega_m`$= \Omega_{\rm m} h^2$

Existing content already in `class`

ΛCDM:

- Baryons `Omega_b`$= \Omega_b$ or `omega_b`$= \Omega_b h^2$
- Cold Dark matter `Omega_cdm`$= \Omega_{\rm cdm}$ or `omega_cdm`$= \Omega_{\rm cdm} h^2$ or `Omega_m`$= \Omega_{\rm m}$ or `omega_m`$= \Omega_{\rm m} h^2$
- Hubble `H0`$= H_0$ or `h`$= h$ or `100*theta_s`$= 100\theta_s$

Existing content already in `class`

ΛCDM:

- Baryons `Omega_b`$= \Omega_b$ or `omega_b`$= \Omega_b h^2$
- Cold Dark matter `Omega_cdm`$= \Omega_{\rm cdm}$ or `omega_cdm`$= \Omega_{\rm cdm} h^2$ or `Omega_m`$= \Omega_{\rm m}$ or `omega_m`$= \Omega_{\rm m} h^2$
- Hubble `H0`$= H_0$ or `h`$= h$ or `100*theta_s`$= 100\theta_s$
- Reionization `z_reio`$= z_{\rm reio}$ or `tau_reio`$= \tau_{\rm reio}$

Existing content already in `class`

ΛCDM:

- Baryons `Omega_b`$= \Omega_b$ or `omega_b`$= \Omega_b h^2$
- Cold Dark matter `Omega_cdm`$= \Omega_{\rm cdm}$ or `omega_cdm`$= \Omega_{\rm cdm} h^2$ or `Omega_m`$= \Omega_{\rm m}$ or `omega_m`$= \Omega_{\rm m} h^2$
- Hubble `H0`$= H_0$ or `h`$= h$ or `100*theta_s`$= 100\theta_s$
- Reionization `z_reio`$= z_{\rm reio}$ or `tau_reio`$= \tau_{\rm reio}$
- Primordial Amplitude `A_s`$= A_s$ or `ln10^_s`$= \ln(10^{10}A_s)$ or `sigma8`$= \sigma_8$

Existing content already in class

ΛCDM:

- Baryons `Omega_b`$= \Omega_b$ or `omega_b`$= \Omega_b h^2$
- Cold Dark matter `Omega_cdm`$= \Omega_{\rm cdm}$ or `omega_cdm`$= \Omega_{\rm cdm} h^2$ or `Omega_m`$= \Omega_{\rm m}$ or `omega_m`$= \Omega_{\rm m} h^2$
- Hubble `H0`$= H_0$ or `h`$= h$ or `100*theta_s`$= 100\theta_s$
- Reionization `z_reio`$= z_{\rm reio}$ or `tau_reio`$= \tau_{\rm reio}$
- Primordial Amplitude `A_s`$= A_s$ or `ln10^_s`$= \ln(10^{10} A_s)$ or `sigma8`$= \sigma_8$
- Primordial tilt `n_s`$= n_s$

Existing content already in `class`

Standard simple extensions:

Existing content already in `class`

Standard simple extensions:

- Curvature Ω_k (`Omega_k`)

Existing content already in class

Standard simple extensions:

- Curvature Ω_k (Omega_k)
- Neutrinos m_ν (m_ncdm, N_ncdm, N_ur adjustment see explanatory.ini)

Existing content already in `class`

Standard simple extensions:

- Curvature Ω_k (`Omega_k`)
- Neutrinos m_ν (`m_ncdm`, `N_ncdm`, `N_ur` adjustment see `explanatory.ini`)
- Free-streaming dark radiation N_{eff} (`N_ur`)

Existing content already in `class`

Standard simple extensions:

- Curvature Ω_k (`Omega_k`)
- Neutrinos m_ν (`m_ncdm`, `N_ncdm`, `N_ur` adjustment see `explanatory.ini`)
- Free-streaming dark radiation N_{eff} (`N_ur`)
- Helium abundance Y_p (`YHe`)

Existing content already in `class`

Standard simple extensions:

- Curvature Ω_k (`Omega_k`)
- Neutrinos m_ν (`m_ncdm`, `N_ncdm`, `N_ur` adjustment see `explanatory.ini`)
- Free-streaming dark radiation N_{eff} (`N_ur`)
- Helium abundance Y_p (`YHe`)
- Primordial running α_s (`alpha_s`)

Existing content already in class

Standard simple extensions:

- Curvature Ω_k (Omega_k)
- Neutrinos m_ν (m_ncdm, N_ncdm, N_ur adjustment see explanatory.ini)
- Free-streaming dark radiation N_{eff} (N_ur)
- Helium abundance Y_p (YHe)
- Primordial running α_s (alpha_s)
- CPL Dark Energy w_0, w_a (Omega_fld, w0_fld, wa_fld, Omega_Lambda=0?)

Existing content already in class

Standard simple extensions:

- Curvature Ω_k (Omega_k)
- Neutrinos m_ν (m_ncdm, N_ncdm, N_ur adjustment see explanatory.ini)
- Free-streaming dark radiation $N_{\rm eff}$ (N_ur)
- Helium abundance Y_p (YHe)
- Primordial running α_s (alpha_s)
- CPL Dark Energy w_0, w_a (Omega_fld, w0_fld, wa_fld, Omega_Lambda=0?)
- CMB temperature $T_{\rm cmb}$

Existing content already in `class`

Dark matter:

- Thermal Warm Dark Matter (`m_ncdm`, `T_ncdm`, `omega_ncdm`)
- Annihilating dark matter
- Decaying dark matter (`Omega_dcdmdr`, `Gamma_dcdm`)
- Non-trivial phase-space distribution (`ncdm` framework), neutrino flavor mixing, neutrino chemical potential
- Interacting with photons, baryons, neutrinos (`idm`)

Existing content already in `class`

Dark matter:

- Thermal Warm Dark Matter (`m_ncdm`, `T_ncdm`, `omega_ncdm`)
- Annihilating dark matter
- Decaying dark matter (`Omega_dcdmdr`, `Gamma_dcdm`)
- Non-trivial phase-space distribution (`ncdm` framework), neutrino flavor mixing, neutrino chemical potential
- Interacting with photons, baryons, neutrinos (`idm`)

Dark radiation:

- Fluid/Self-interacting (`idr`)
- Viscous (`ceff2_ur`, `cvis_ur`)

Existing content already in `class`

Dark matter:

- Thermal Warm Dark Matter (`m_ncdm`, `T_ncdm`, `omega_ncdm`)
- Annihilating dark matter
- Decaying dark matter (`Omega_dcdmdr`, `Gamma_dcdm`)
- Non-trivial phase-space distribution (`ncdm` framework), neutrino flavor mixing, neutrino chemical potential
- Interacting with photons, baryons, neutrinos (`idm`)

Dark radiation:

- Fluid/Self-interacting (`idr`)
- Viscous (`ceff2_ur`, `cvis_ur`)

Dark energy:

- CPL, EDE (`fld`) (this EDE is not the usual EDE → `https://github.com/PoulinV/AxiCLASS` and `https://github.com/flo1984/TriggerCLASS` to be merged)
- Other fluid-like DE (`fld`)
- Quintessence/Scalar field (`scf`)

Existing content already in `class`

Thermal modeling

- Recfast recombination
- Hyrec-2recombination
- Tanh reionization
- Multi-tanh reionization
- Reionization from file
- Energy injection (PBH Evaporation,PBH Accretion,DM Decay,DM Annihilation)

Existing content already in `class`

Thermal modeling

- Recfast recombination
- Hyrec-2recombination
- Tanh reionization
- Multi-tanh reionization
- Reionization from file
- Energy injection (PBH Evaporation,PBH Accretion,DM Decay,DM Annihilation)

Spectral distortions

- y and μ distortions
- PCA of intermediate distortions

Existing content already in `class`

Existing content already in `class`

Inflation/Primordial Powerspectrum

- Arbitrary potential $V(\phi)$
- Automatic finding of observable window
- Can impose inflation duration
- Tensor modes included automatically
- Read from file

Existing content already in `class`

Inflation/Primordial Powerspectrum

- Arbitrary potential $V(\phi)$
- Automatic finding of observable window
- Can impose inflation duration
- Tensor modes included automatically
- Read from file

Initial conditions

- Isocurvature modes (ν,b,cdm)
- Synchronous/Newtonian gauges

Existing content already in `class`

Inflation/Primordial Powerspectrum

- Arbitrary potential $V(\phi)$
- Automatic finding of observable window
- Can impose inflation duration
- Tensor modes included automatically
- Read from file

Initial conditions

- Isocurvature modes (ν,b,cdm)
- Synchronous/Newtonian gauges

Nonlinear estimation/emulation

- Halofit
- HMcode 2016 and 2022
- Perturbation theory (very soon)

Existing content already in `class`

Inflation/Primordial Powerspectrum

- Arbitrary potential $V(\phi)$
- Automatic finding of observable window
- Can impose inflation duration
- Tensor modes included automatically
- Read from file

Initial conditions

- Isocurvature modes (ν,b,cdm)
- Synchronous/Newtonian gauges

Nonlinear estimation/emulation

- Halofit
- HMcode 2016 and 2022
- Perturbation theory (very soon)

Modified Gravity
$\rightarrow$ HiCLASS branch (Bellini, Sawicki, Zumalacarregui, `http://www.hiclass-code.net`)

Existing content already in `class`

Inflation/Primordial Powerspectrum

- Arbitrary potential $V(\phi)$
- Automatic finding of observable window
- Can impose inflation duration
- Tensor modes included automatically
- Read from file

Initial conditions

- Isocurvature modes (ν,b,cdm)
- Synchronous/Newtonian gauges

Nonlinear estimation/emulation

- Halofit
- HMcode 2016 and 2022
- Perturbation theory (very soon)

Modified Gravity
$\rightarrow$ HiCLASS branch (Bellini, Sawicki, Zumalacarregui, `http://www.hiclass-code.net`)

Extension to second-order perturbation theory
SONG (Fidler, Pettinari, Tram, `https://github.com/coccoinomane/song`)

Fundamental layout of Einstein-Boltzmann solvers

Homogeneous background $H(a), \rho_i(a), D(a), \ldots$	→	Homogeneous thermodynamics $x_e(z), T_b(z), c_s(z), \ldots$

Fundamental layout of Einstein-Boltzmann solvers

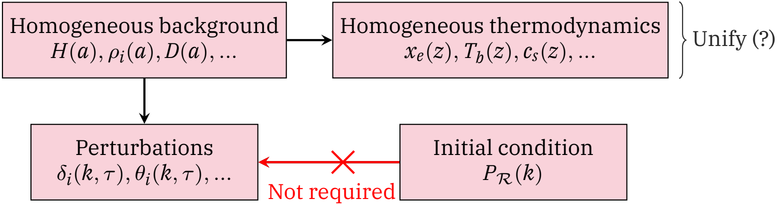
Homogeneous background
$H(a), \rho_i(a), D(a), \ldots$

→

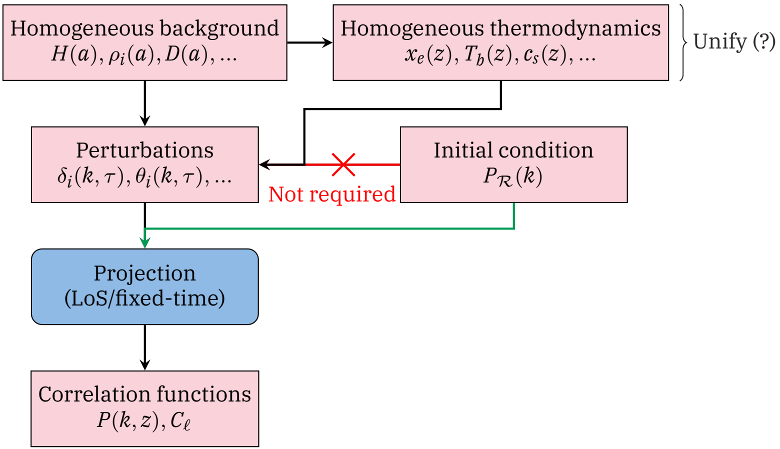
Homogeneous thermodynamics
$x_e(z), T_b(z), c_s(z), \ldots$

Unify (?)

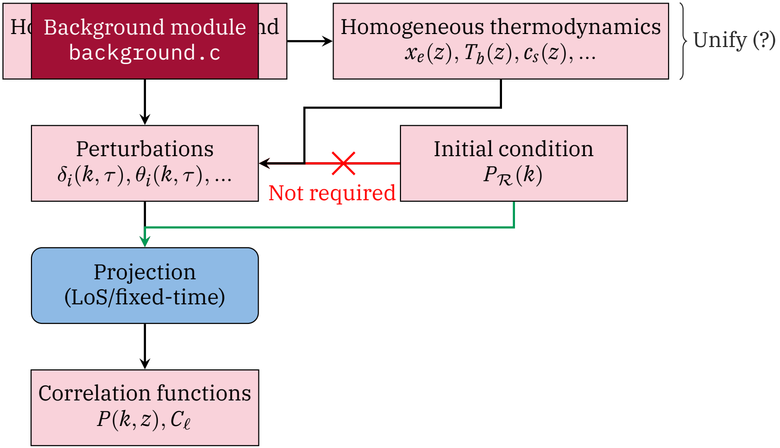
Fundamental layout of Einstein-Boltzmann solvers



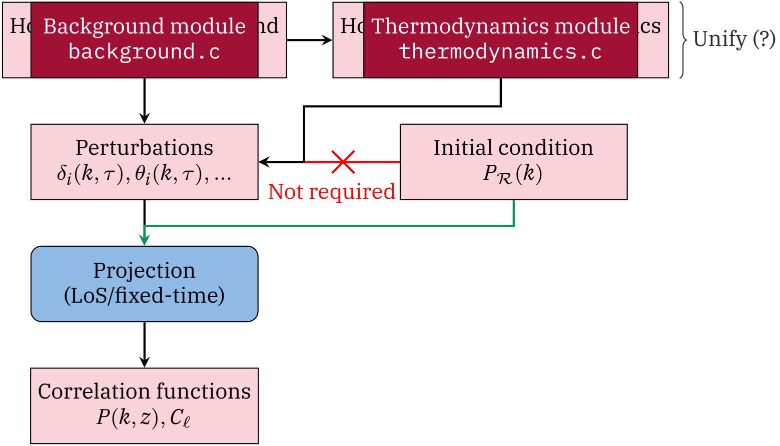
Fundamental layout of Einstein-Boltzmann solvers



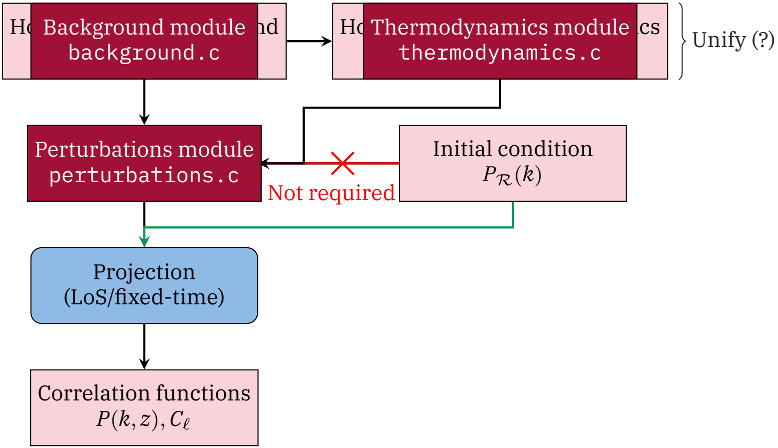
Fundamental layout of Einstein-Boltzmann solvers



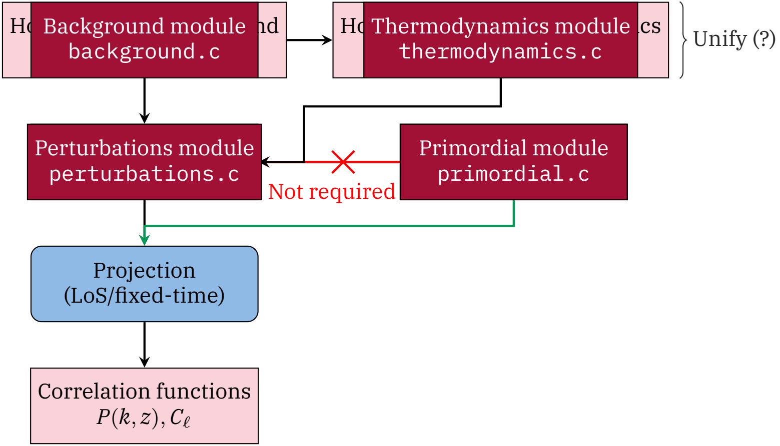
Fundamental layout of Einstein-Boltzmann solvers



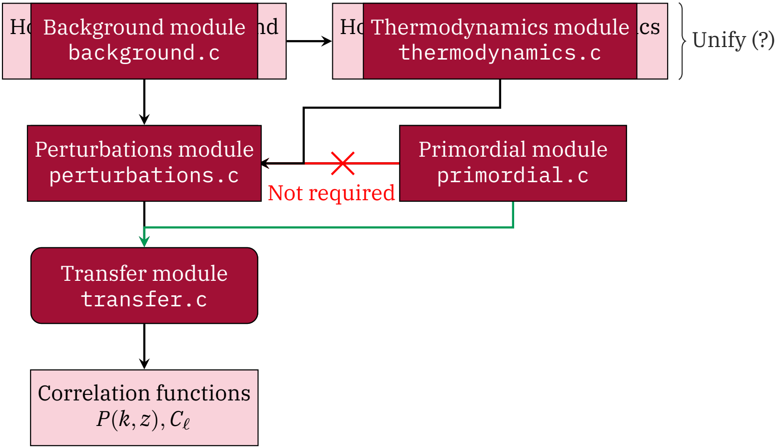
Fundamental layout of Einstein-Boltzmann solvers



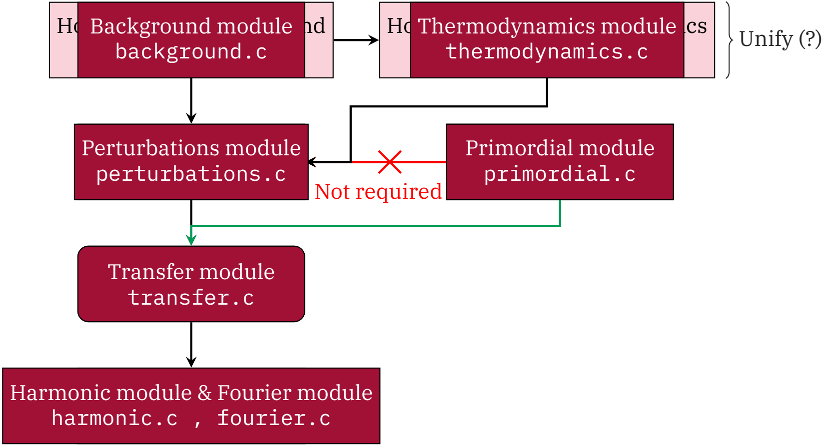
Fundamental layout of Einstein-Boltzmann solvers



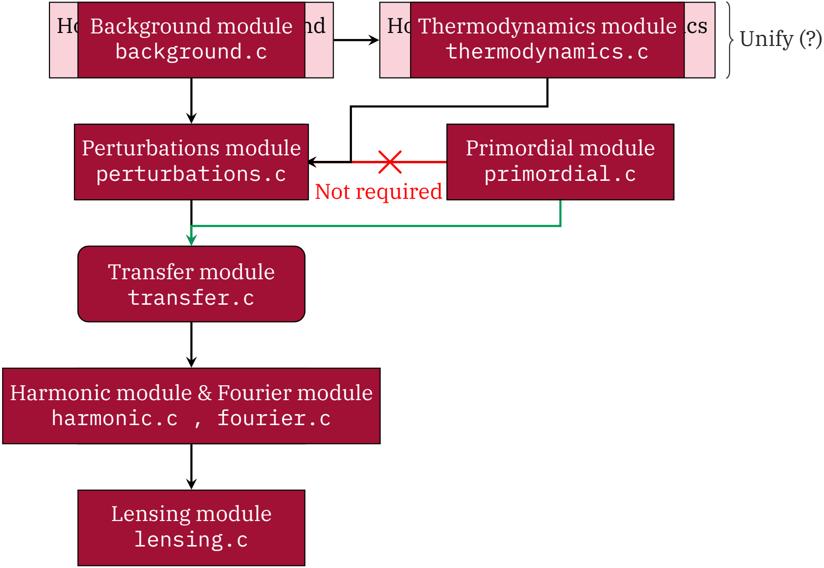
Fundamental layout of Einstein-Boltzmann solvers



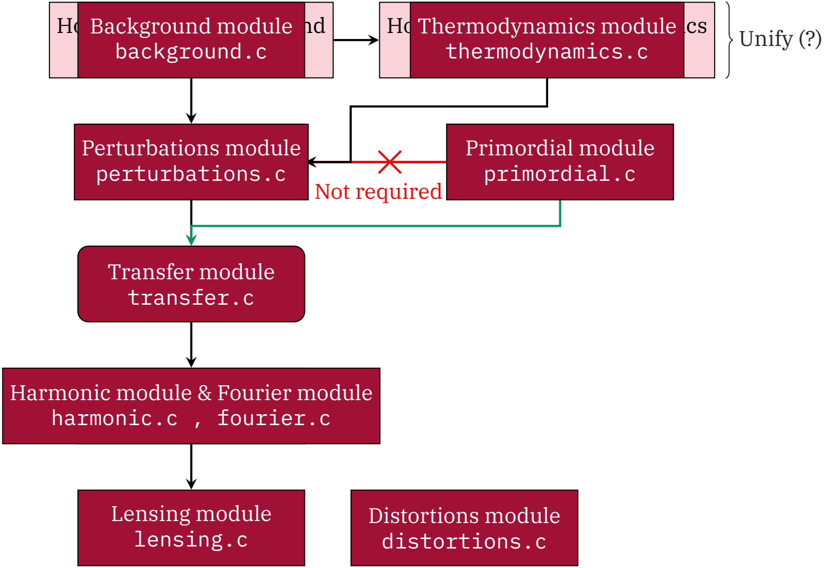
Fundamental layout of Einstein-Boltzmann solvers



Fundamental layout of Einstein-Boltzmann solvers



Fundamental layout of Einstein-Boltzmann solvers



Fundamental layout of Einstein-Boltzmann solvers