Tools for cosmology: the CLASS and Monte Python codes

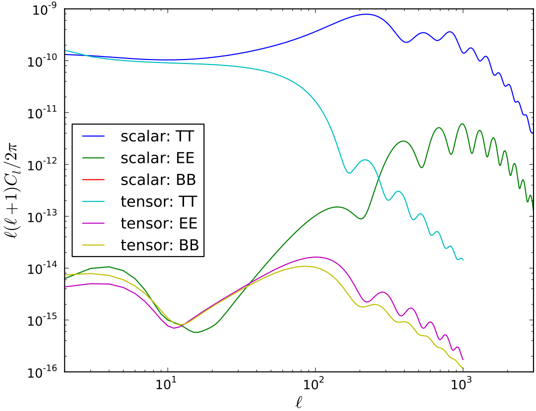
Benjamin Audren[a], Julien Lesgourgues[b,c], Thomas Tram[d]

[a]EPFL, [b]CERN, [c]LAPTh, [d]Portsmouth

Kavli IPMU, Tokyo, 27-31.10.2014

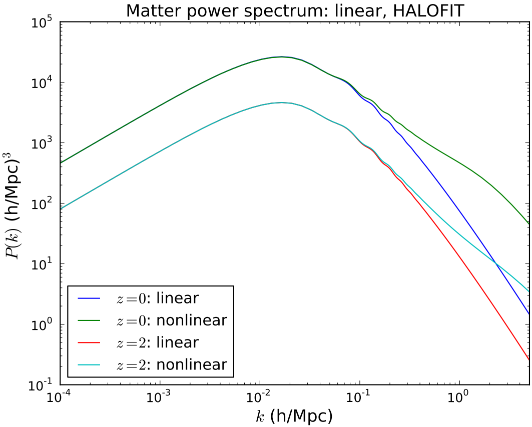
Almost any type of research activity in cosmology will use a Boltzmann code at some point

Computing CMB anisotropy spectra:



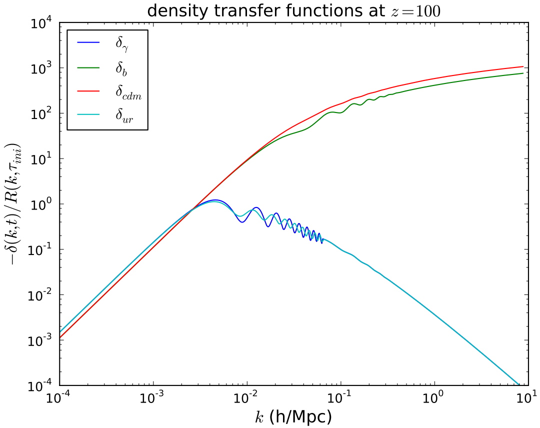
Almost any type of research activity in cosmology will use a Boltzmann code at some point

Computing matter power spectrum:



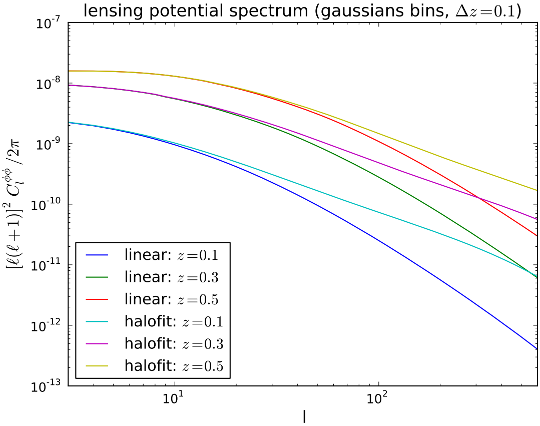
Almost any type of research activity in cosmology will use a Boltzmann code at some point

Computing transfer functions (e.g. initial conditions for N-body):



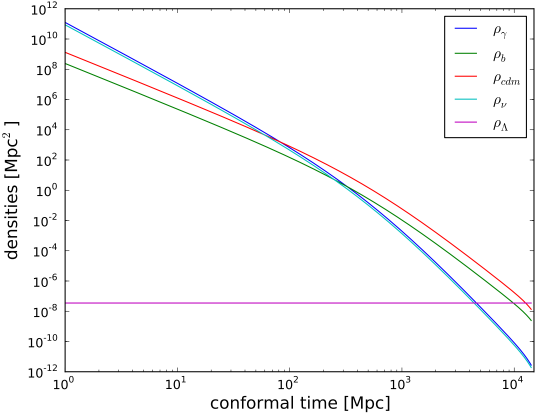
Almost any type of research activity in cosmology will use a Boltzmann code at some point

Computing matter density (number count) spectra, or lensing angular spectra:



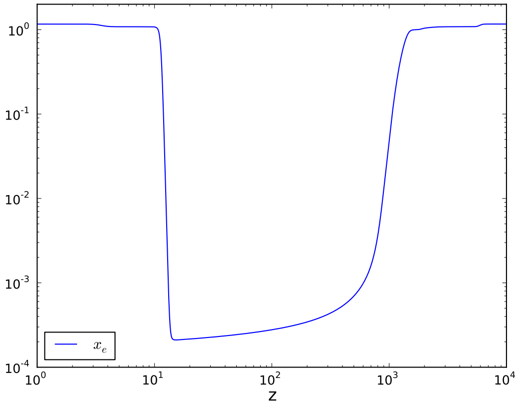
Almost any type of research activity in cosmology will use a Boltzmann code at some point

Computing background evolution in a given cosmological model:



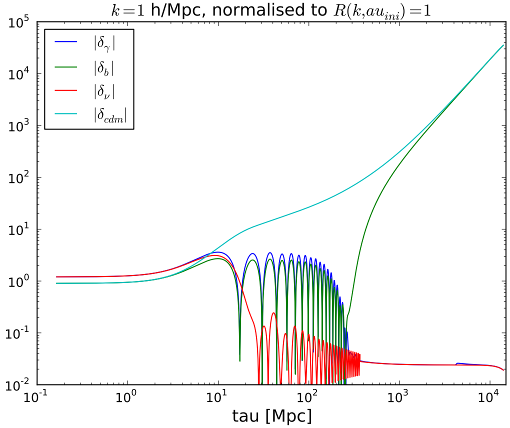
Almost any type of research activity in cosmology will use a Boltzmann code at some point

Computing thermal history in a given cosmological model:



Almost any type of research activity in cosmology will use a Boltzmann code at some point

Computing evolution of perturbations for individual Fourier modes:



Almost any type of research activity in cosmology will use a Boltzmann code at some point

As an observer...

... you want to compute these quantities easily and efficiently
... you may want to develop the code for outputting new observables

Almost any type of research activity in cosmology will use a Boltzmann code at some point

As an observer...

... you want to compute these quantities easily and efficiently
... you may want to develop the code for outputting new observables

As a theorist...

... you may want to develop the code for incorporating new physics
and compute/understand the effects of your favourite model on observables

Almost any type of research activity in cosmology will use a Boltzmann code at some point

As an observer...

... you want to compute these quantities easily and efficiently
... you may want to develop the code for outputting new observables

As a theorist...

... you may want to develop the code for incorporating new physics
and compute/understand the effects of your favourite model on observables

As both...

... you want to infer constraints on cosmological parameters from a new dataset
... you want to test your own favorite model, given existing data
... you want to predict the sensitivity of a future experiment to a given parameter

Program: Day 1

DAY I : Monday 27th October

09:30-10:15	CLASS	History, goals & philosophy.	JL
10:15-11:00	General	Bayesian parameter extraction.	BA
		Coffee	
11:30-12:15	CLASS	Overall style and structure.	JL
		Lunch	
13:30-14:15	CLASS	Input and output files. Basic running.	JL
14:15-15:00	CLASS	How to visualise the output.	TT
		Tea	
15:45-16:30	Optional	Lecturers will answer questions and provide help on exercises	

Brief history of Boltzmann codes

- 1995: Bertschinger releases the COSMICS package in f77. Contains Ma & Bertschinger (`astro-ph/9506072`) equations in synchronous gauge, Peebles recombination. Integration of Boltzmann eq. for photons/neutrinos till $\ell \sim 2500$.

Brief history of Boltzmann codes

- 1995: Bertschinger releases the COSMICS package in f77. Contains Ma & Bertschinger (`astro-ph/9506072`) equations in synchronous gauge, Peebles recombination. Integration of Boltzmann eq. for photons/neutrinos till $\ell \sim 2500$.
- 1996: Seljak & Zaldarriaga add a few functions for computing the source functions and convolve them with Bessel functions. New code much faster, released as CMBFAST.

Brief history of Boltzmann codes

- 1995: Bertschinger releases the COSMICS package in f77. Contains Ma & Bertschinger (`astro-ph/9506072`) equations in synchronous gauge, Peebles recombination. Integration of Boltzmann eq. for photons/neutrinos till $\ell \sim 2500$.
- 1996: Seljak & Zaldarriaga add a few functions for computing the source functions and convolve them with Bessel functions. New code much faster, released as CMBFAST.
- CMBFAST improved: RECFAST recombination, open/closed, lensing, but structure of the code becomes complicated.

Brief history of Boltzmann codes

- 1995: Bertschinger releases the COSMICS package in f77. Contains Ma & Bertschinger (`astro-ph/9506072`) equations in synchronous gauge, Peebles recombination. Integration of Boltzmann eq. for photons/neutrinos till $\ell \sim 2500$.
- 1996: Seljak & Zaldarriaga add a few functions for computing the source functions and convolve them with Bessel functions. New code much faster, released as CMBFAST.
- CMBFAST improved: RECFAST recombination, open/closed, lensing, but structure of the code becomes complicated.
- 1999: Lewis et al. cut CMBFAST in pieces and reorganise them differently in f90 in CAMB. Improved expressions for sources, initial conditions, lensing, etc.

Brief history of Boltzmann codes

- 1995: Bertschinger releases the COSMICS package in f77. Contains Ma & Bertschinger (`astro-ph/9506072`) equations in synchronous gauge, Peebles recombination. Integration of Boltzmann eq. for photons/neutrinos till $\ell \sim 2500$.
- 1996: Seljak & Zaldarriaga add a few functions for computing the source functions and convolve them with Bessel functions. New code much faster, released as CMBFAST.
- CMBFAST improved: RECFAST recombination, open/closed, lensing, but structure of the code becomes complicated.
- 1999: Lewis et al. cut CMBFAST in pieces and reorganise them differently in f90 in CAMB. Improved expressions for sources, initial conditions, lensing, etc.
- 2003: Doran does a similar work of reorganisation in C++: CMBEASY

Brief history of Boltzmann codes

- 1995: Bertschinger releases the COSMICS package in f77. Contains Ma & Bertschinger (`astro-ph/9506072`) equations in synchronous gauge, Peebles recombination. Integration of Boltzmann eq. for photons/neutrinos till $\ell \sim 2500$.
- 1996: Seljak & Zaldarriaga add a few functions for computing the source functions and convolve them with Bessel functions. New code much faster, released as CMBFAST.
- CMBFAST improved: RECFAST recombination, open/closed, lensing, but structure of the code becomes complicated.
- 1999: Lewis et al. cut CMBFAST in pieces and reorganise them differently in f90 in CAMB. Improved expressions for sources, initial conditions, lensing, etc.
- 2003: Doran does a similar work of reorganisation in C++: CMBEASY
- later: CAMB maintained and improved over the years; others not.

What would be expected from a new Boltzmann code?

Three goals of the Cosmic Linear Anisotropy Solving System (CLASS):

What would be expected from a new Boltzmann code?

Three goals of the Cosmic Linear Anisotropy Solving System (CLASS):

1. Friendly and flexible: should be easy to compile, to pass input parameters, to understand the code, and to modify it (extended cosmological scenarios, new observables).

What would be expected from a new Boltzmann code?

Three goals of the Cosmic Linear Anisotropy Solving System (CLASS):

1. Friendly and flexible: should be easy to compile, to pass input parameters, to understand the code, and to modify it (extended cosmological scenarios, new observables).
2. Accurate: need more and more precision. Analysing Planck and WMAP data required very different accuracy settings. Before, CAMB precision could only be calibrated w.r.t itself. CLASS played important role in pushing precision to Planck level. Similar efforts in the future (LSS, next CMB satellite, 21cm, etc.)

What would be expected from a new Boltzmann code?

Three goals of the Cosmic Linear Anisotropy Solving System (CLASS):

1. Friendly and flexible: should be easy to compile, to pass input parameters, to understand the code, and to modify it (extended cosmological scenarios, new observables).
2. Accurate: need more and more precision. Analysing Planck and WMAP data required very different accuracy settings. Before, CAMB precision could only be calibrated w.r.t itself. CLASS played important role in pushing precision to Planck level. Similar efforts in the future (LSS, next CMB satellite, 21cm, etc.)
3. Fast: for parameter extraction (Metropolis-Hastings, Multinest, Cosmo Hammer, grid-base methods). Typical project: 10'000 to 1'000'000 executions

The philosophy of `class`

Our efforts for ensuring flexibility and friendliness in CLASS, summarised in 14 key points

Friendliness and flexibility in 14 points

1. Written in plain C with no external libraries

C is free, diffuse, easy, fast (more than C++). Self-contained and ready to install, straightforward to compile.

Friendliness and flexibility in 14 points

1. Written in plain C with no external libraries

C is free, diffuse, easy, fast (more than C++). Self-contained and ready to install, straightforward to compile.

2. Input parameters are "interpreted"

Some basic logic has been incorporated in the code. Easy to elaborate further.

Examples:
- expects only one out of $\{H_0, h, 100 \times \theta_s\}$, otherwise complains;
- missing ones inferred from given one
- same with $\{T_{\rm cmb}, \Omega_\gamma, \omega_\gamma\}$, $\{\Omega_{\rm ncdm}, \omega_{\rm ncdm}, m_\nu\}$, $\{\Omega_{\rm ur}, \omega_{\rm ur}, N_{\rm ur}\}$,...

Look at beginning of explanatory.ini!!!

```
class$ more explanatory.ini
```

Friendliness and flexibility in 14 points

1. Written in plain C with no external libraries

C is free, diffuse, easy, fast (more than C++). Self-contained and ready to install, straightforward to compile.

2. Input parameters are "interpreted"

Some basic logic has been incorporated in the code. Easy to elaborate further.

Examples:
- expects only one out of $\{H_0,\ h,\ 100\times\theta_s\}$, otherwise complains;
- missing ones inferred from given one
- same with $\{T_{\rm cmb},\ \Omega_\gamma,\ \omega_\gamma\}$, $\{\Omega_{\rm ncdm},\ \omega_{\rm ncdm},\ m_\nu\}$, $\{\Omega_{\rm ur},\ \omega_{\rm ur},\ N_{\rm ur}\}$,...

3. Perturbation equations and notations taken literally from well-known Ma & Bertschinger (`astro-ph/9506072`) paper ...

... rather than specific notations of one given group, or mixed notations from various origins.

For non-flat universes we found and published the simplest possible generalisation of Ma & Bertschinger notations, (`arXiv:1305.3261`).

Friendliness and flexibility in 14 points

4. Code intensively documented

As many comment lines as C lines

Friendliness and flexibility in 14 points

4. Code intensively documented

As many comment lines as C lines

5. Easy units

All important variables are either dimensionless or in Mpc^n

Friendliness and flexibility in 14 points

4. Code intensively documented

As many comment lines as C lines

5. Easy units

All important variables are either dimensionless or in Mpc^n

6. No hard coding, for example:

• Never write a sampling step scheme in physical units; code infers sampling as given fraction of dimensionless physical quantities;
• Never write the index of an array as an integer; indexing done automatically and internally by the code; use symbolic index names;

Friendliness and flexibility in 14 points

4. Code intensively documented

As many comment lines as C lines

5. Easy units

All important variables are either dimensionless or in Mpc^n

6. No hard coding, for example:

- Never write a sampling step scheme in physical units; code infers sampling as given fraction of dimensionless physical quantities;
- Never write the index of an array as an integer; indexing done automatically and internally by the code; use symbolic index names;

7. No global variables

All variables passed as arguments of functions. Important for readability and parallelisation.

Friendliness and flexibility in 14 points

8. Clear modular structure

Dinstinct modules with separate physical tasks. No duplicate equations.

```
 1. input.c
 2. background.c
 3. thermodynamics.c
 4. perturbations.c
 5. primordial.c
 6. nonlinear.c
 7. transfer.c
 8. spectra.c
 9. lensing.c
10. output.c
```

E.g.: Friedmann equation appears in one single place. Same for linearised Einstein equations. Ideal for implementing modified gravity theories.

Search `Friedmann` in source/background.c

Friendliness and flexibility in 14 points

9. All precision variables grouped in one single place (`input.c`), and even inside a single structure '`precision`'

There are... many. True for any code, but they are usually hidden and spread!

Friendliness and flexibility in 14 points

9. All precision variables grouped in one single place (`input.c`), and even inside a single structure '`precision`'

There are... many. True for any code, but they are usually hidden and spread!

10. Given "ingredient" always implemented between brackets, in zone switched by a flag

- adding new physics does not slow down the code or compromise its readability.
- incentive to add lots of new things even if rarely used, with no drawback.
- with a search, one can localise all the parts of the code related to a given ingredient.

Examples:

```
if (has_fld == TRUE) {...}
if (has_cmb_lensing == TRUE) {...}
```

Search `has_fld` in source/background.c

Friendliness and flexibility in 14 points

9. All precision variables grouped in one single place (`input.c`), and even inside a single structure '`precision`'

There are... many. True for any code, but they are usually hidden and spread!

10. Given "ingredient" always implemented between brackets, in zone switched by a flag

- adding new physics does not slow down the code or compromise its readability.
- incentive to add lots of new things even if rarely used, with no drawback.
- with a search, one can localise all the parts of the code related to a given ingredient.

Examples:

```
if (has_fld == TRUE) {...}
if (has_cmb_lensing == TRUE) {...}
```

11. Adding new ingredient...

... can be done by searching for occurrence of another similar ingredient, copy/pasting, and adapting the new lines.

Example: if you want to add a new Dark Energy component, you may search for '_fld', duplicate all corresponding lines, change '_fld' into e.g. '_myde', and adapt the physical equations.

Friendliness and flexibility in 14 points

12. Error management

In principle, public CLASS should never crash. In case of problem, it returns an error message, with a well-documented error (line, function, what caused the crash, how to avoid it). Most of this message is generated automatically by the code.

Friendliness and flexibility in 14 points

12. Error management

In principle, public CLASS should never crash. In case of problem, it returns an error message, with a well-documented error (line, function, what caused the crash, how to avoid it). Most of this message is generated automatically by the code.

13. Version history

Old versions can always be downloaded. In most cases, new versions feature new ingredients and avoid (whenever possible) to modify or erase the old ones. Try to develop in such way that modifications to an old version can still be pasted in a new version (as much as possible).

Friendliness and flexibility in 14 points

12. Error management

In principle, public CLASS should never crash. In case of problem, it returns an error message, with a well-documented error (line, function, what caused the crash, how to avoid it). Most of this message is generated automatically by the code.

13. Version history

Old versions can always be downloaded. In most cases, new versions feature new ingredients and avoid (whenever possible) to modify or erase the old ones. Try to develop in such way that modifications to an old version can still be pasted in a new version (as much as possible).

14. Git repository and GitHub website.

The code can be downloaded as a `.tar.gz`, or as a git repository. Then, user can develop his own modification with the advantage of git (branching, memory of changes...); or merge his changes with a newer version almost automatically; or submit his modifications to the CLASS team in view of an easy merging with the public version.