CLASS

the Cosmological Linear Anisotropy Solving System[1]

Julien Lesgourgues
TTK, RWTH Aachen University

IHP, Paris, 18.11.2019

[1] code developed by Julien Lesgourgues & Thomas Tram plus many others...

class at IHP: plan

1. Goals
2. coding spirit and basic rules
3. Where to find for information and tutorials
4. Input/output options specific to Large Scale Structure
5. Developpement related to LSS calculation performance

Goals

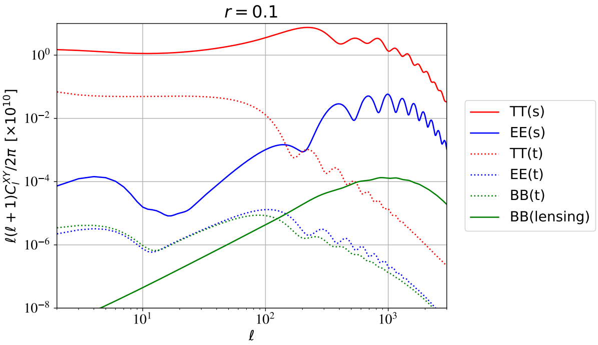
Project started on request of Planck science team, in order to have a tool independent from CAMB, and check for possible Boltzmann-code-induced bias in parameter extraction. The `class`-CAMB comparison has triggered progress in the accuracy of both codes. Agreement established at 10^{-4} (0.01%) level for CMB observables, using highest-precision settings in both codes. But the `class` projected expanded and went much further than the initial Planck purposes.

`class` aims at being:

- general (more models, more output/observables)
- modern (structured, modular, flexible, wrap-able: wrapper for python, C++, automatic precision test code)
- user friendly (documented, structured, easy to understand) and hence easier to modify (coding additional models/observables)
- accurate and fast (current master branch comparable to CAMB; ongoing developements: strong prospects for speed up, see last part of the talk)

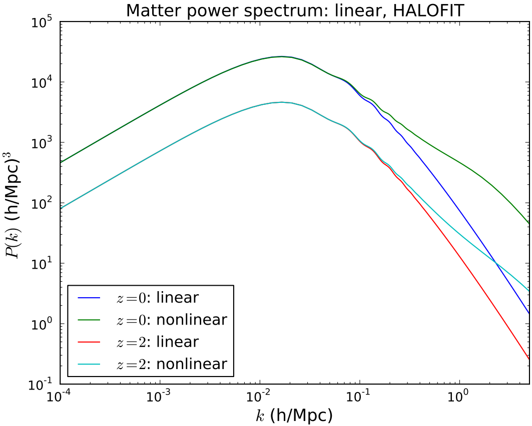
Goals: with `class` you can get:

The CMB anisotropy spectra:

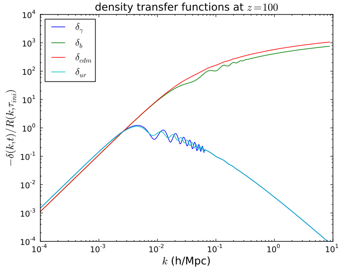


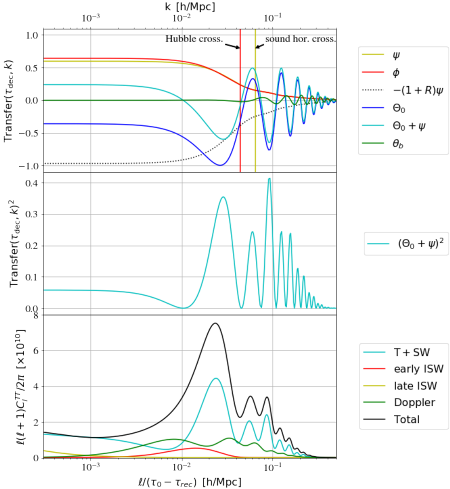
Goals: with `class` you can get:

The matter power spectrum with NL corrections from Halofit – or HMcode with ≥ 2.8:



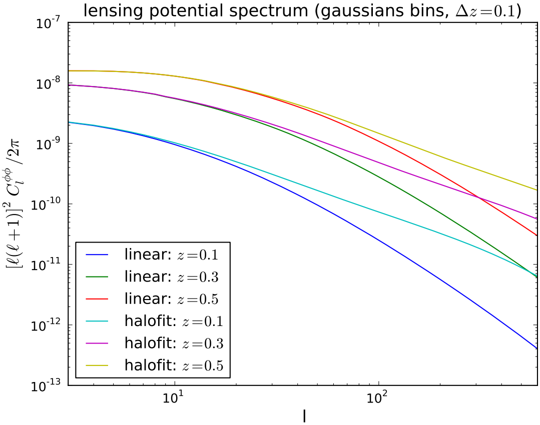
The transfer functions at a given time/redshift (e.g. initial conditions for N-body):





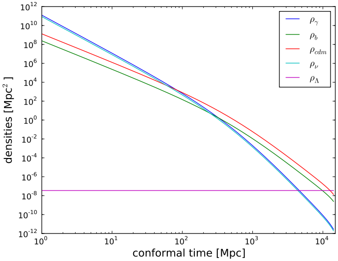
Goals: with class you can get:

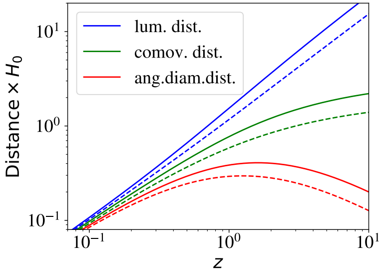
The matter density (number count) C_l's, or the lensing C_l's (with arbitrary selection/window functions):



Goals: with `class` you can get:

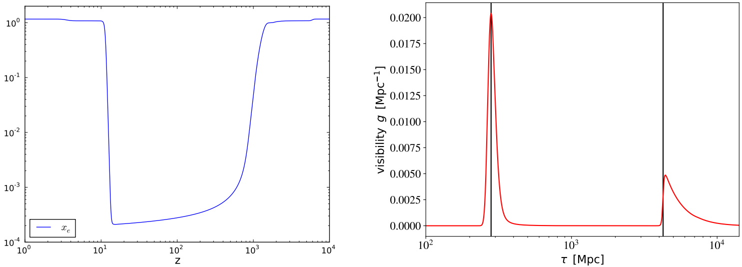
The background evolution in a given cosmological model:



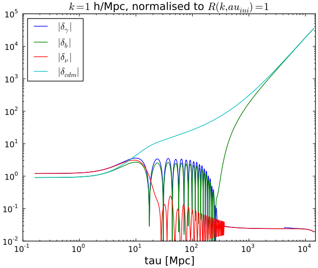


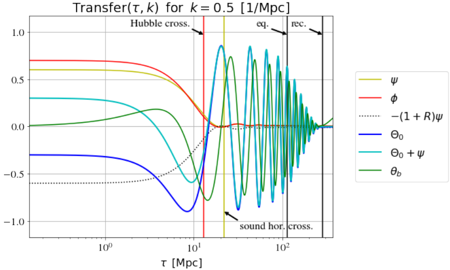
Goals: with `class` you can get:

The thermal history in a given cosmological model:



The time evolution of perturbations for individual Fourier modes:





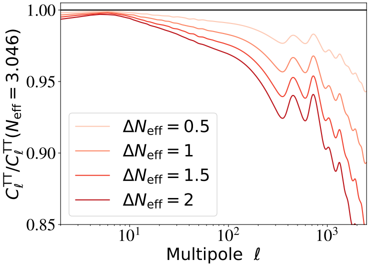
Goals: with `class` you can get:

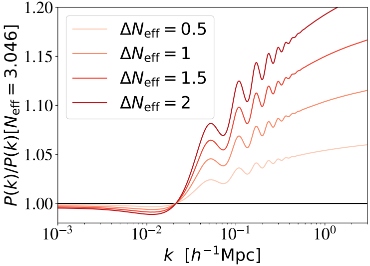
... and several other quantities, for instance:

- characteristic redshifts, physical and comoving scales, angles;
- primordial spectrum for given inflationary potential;
- decomposition of CMB C_l's in intrinsic, Sachs-Wolfe, Doppler, ISW, etc.;
- decomposition of galaxy number count C_l's in density, RSD, lensing, etc.;
- ≥ 3.0 : *CMB spectral distorsions* `[arXiv:1910.04619]`
- ...

Goals: with class you can get:

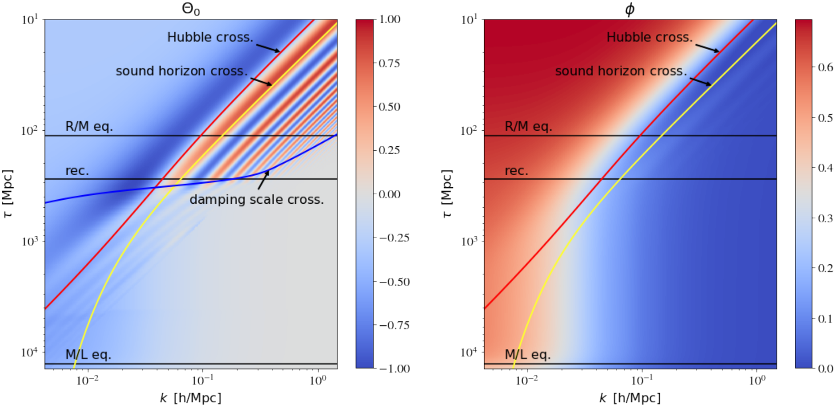
... if you use class as a python module you can extract all kind of output or intermediate quantities, manipulate them in various ways, and make all kinds of computations or nice plots:





Goals: with class you can get:

... if you use class as a python module you can extract all kind of output or intermediate quantities, manipulate them in various ways, and make all kinds of computations or nice plots:

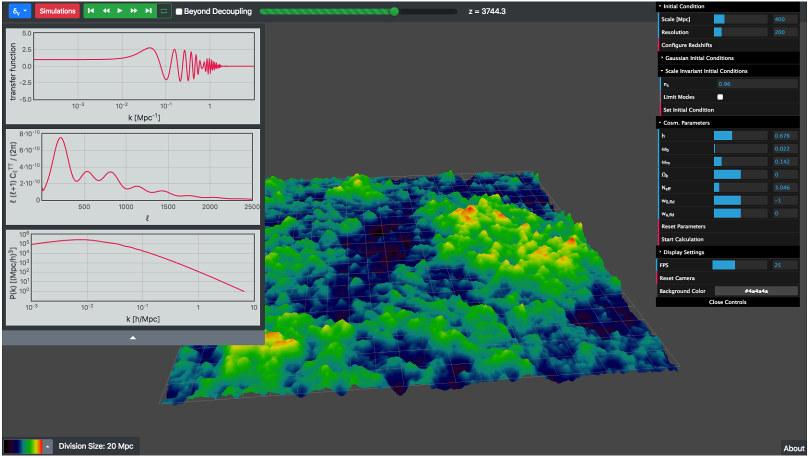


Goals: with class you can get:

... all this for a wide range of cosmological models: all those implemented in the public CAMB code, plus several other ingredients, especially in the sectors of:

- primordial perturbations (internal inflationary perturbation module with given $V(\phi)$, takes arbitrary BSI spectra, correlated isocurvature modes),
- neutrinos (chemical potentials, arbitrary phase-space distributions, flavor mixing...),
- Dark Matter (warm, annihilating, decaying, ≥ 2.9 : *interacting...*),
- Dark Energy (fluid with flexible $w(a)$ + sound speed, quintessence with given $V(\phi)$)
- also Modified Gravity if you try the recently released HiCLASS branch (Bellini, Sawicki, Zumalacarregui, `http://www.hiclass-code.net`)
- multi-gauge (synchronous, newtonian, ≥ 2.8 : *N-body...*)
- extension to second-order perturbation theory: SONG (Fidler, Pettinari, Tram, `https://github.com/coccoinomane/song`)
- interfacing with particle physics modules and codes for exotic energy injection available in ExoCLASS branch of `http://github.com/lesgourg/class_public.git` (Stöcker, Poulin)
- Class_sz (B. Bolliet), MultiClass (group of L. Verde), ...

… and movies of perturbations in 2D slices of early universe with our Real space graphical interface (≥ 2.7); here is a snapshot:



class coding spirit and basic rules

Equations follow literally notations of most famous papers

(in particular Ma & Bertschinger 1996, astro-ph/9506072).
Multi-gauge code: everything coded in newtonian and synchronous gauge, structure ready for more gauges.

`class` coding spirit and basic rules

Equations follow literally notations of most famous papers

(in particular Ma & Bertschinger 1996, `astro-ph/9506072`).
Multi-gauge code: everything coded in newtonian and synchronous gauge, structure ready for more gauges.

Input parameters interpreted and processed into final form needed by the modules

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 $\{T_{\rm cmb},\ \Omega_\gamma,\ \omega_\gamma\}$, $\{\Omega_{\rm ncdm},\ \omega_{\rm ncdm},\ m_\nu\}$, $\{\Omega_{\rm ur},\ \omega_{\rm ur},\ N_{\rm ur}\}$,...

class coding spirit and basic rules

Equations follow literally notations of most famous papers

(in particular Ma & Bertschinger 1996, `astro-ph/9506072`).
Multi-gauge code: everything coded in newtonian and synchronous gauge, structure ready for more gauges.

Input parameters interpreted and processed into final form needed by the modules

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 $\{T_{\rm cmb}, \Omega_\gamma, \omega_\gamma\}$, $\{\Omega_{\rm ncdm}, \omega_{\rm ncdm}, m_\nu\}$, $\{\Omega_{\rm ur}, \omega_{\rm ur}, N_{\rm ur}\}$,...

Homogeneous units

Inside all modules except thermodynamics: everything in Mpc^n.
Examples:
- conformal time τ in Mpc, $H = \frac{a'}{a^2}$ in Mpc^{-1}
- $\rho_{\rm class} \equiv \frac{8\pi G}{3}\rho_{\rm physical}$ in Mpc^{-2}, such that $H^2 = \sum_i \rho_i$
- k in Mpc^{-1}, $P(k)$ in Mpc^3

class coding spirit and basic rules

Accessible and self-contained

Plain C (for performance and readability) but mimicking features of C++ (see later).
No external libraries for a quick installation (but parallelisation requires OpenMP).
Lots of comments in the code.
Automatic `doxygen` documentation (Credits Deanna C. Hooper)

`class` coding spirit and basic rules

Accessible and self-contained

Plain C (for performance and readability) but mimicking features of C++ (see later).
No external libraries for a quick installation (but parallelisation requires OpenMP).
Lots of comments in the code.
Automatic `doxygen` documentation (Credits Deanna C. Hooper)

Structured and flexible

Sequence of ten modules with distinct physical tasks, no duplicate equations.

class coding spirit and basic rules

Plethoric accumulation of extended models/observables/features without making the code slower or less readable

Relies on homogeneous style and strict rules (e.g. anything related to given feature is inside an: `if (has_feature == _TRUE_){...}`)

`class` coding spirit and basic rules

Plethoric accumulation of extended models/observables/features without making the code slower or less readable

Relies on homogeneous style and strict rules (e.g. anything related to given feature is inside an: `if (has_feature == _TRUE_){...}`)

No hard-coding

- All indices allocated dynamically (according to strict and homogeneous rules for more readability)
  → see tutorials, e.g. New York lectures, "CLASS Coding", pages 8-11
- All arrays allocated dynamically
- Essentially no number found in the codes except factors in physical equations
- No hard-coded precision parameters, all precision-related numbers/flags gathered in single structure `precision`
- Not a single global variable: all variables passed as arguments of functions (for readability and parallelisation)
- Sampling steps inferred dynamically by the code for each model
- Time for switching approximations on/off inferred dynamically by the code for each model

class coding spirit and basic rules

Rigorous error management

In principle, no segmentation faults when executing public class.
When class fails, it returns an error message with a tree-like information (like e.g. python)
$\rightarrow$ see tutorials, e.g. New York lectures, "CLASS Coding", pages 15-23

class coding spirit and basic rules

Rigorous error management

In principle, no segmentation faults when executing public `class`.
When `class` fails, it returns an error message with a tree-like information (like e.g. python)
→ see tutorials, e.g. New York lectures, "CLASS Coding", pages 15-23

Version history

All previous versions can be downloaded and compared on GitHub, changes documented in `class-code.net`
Always aim at developing without breaking compatibility with older versions.
Own changes can often be merged in newer version with `git merge`.

class coding spirit and basic rules: the 10 class modules

Executing class means going once through the sequence of modules:

```
 1. input.c              # parse/make sense of input parameters
                         # (advanced logic)
 2. background.c.        # homogeneous cosmology
 3. thermodynamics.c.    # ionisation history, scattering rate
 4. perturbations.c.     # linear Fourier perturbations
 5. primordial.c.        # primordial spectrum, inflation
 6. nonlinear.c          # recipes for non-linear corrections
                         # to 2-point statistics
 7. transfer.c.          # from Fourier to multipole space
 8. spectra.c.           # 2-point statistics (power spectra)
 9. lensing.c            # CMB lensing
10. output.c             # print out (not used from python)
```

Plain C (for performances and readability purposes) mimicking C++ and object-oriented programming:

- In C++: 10 "classes", each with a constructor/destructor and a few functions callable from outside.
- In class: each module (files *.c and *.h) is associated to one structure (with all its input/output data), one initialisation function, one freeing function, and a few functions callable from outside.
- main executable only consists in calling the 10 initialisation and ten freeing functions!

class coding spirit and basic rules: class/ directory

In your class directory (e.g. class_public-2.7.2/), you should see:

```
source/   # the 10 modules of class:
          # ALL THE PHYSICS
tools/    # auxiliary pieces of code (numerical methods):
          # ALL THE MATH (no external C library)
main/     # main class function: short, just calls 10 modules
test/     # other main functions for testing part of the code
output/   # output files (when running from terminal)
include/  # header files (*.h) containing declarations
doc/      # pdf version of the manual
python/   # python wrapper of class
cpp/      # C++ wrapper of class
notebooks/ # example of jupyter notebooks
scripts/  # same as plain python scripts
RealSpaceInterface/ # graphical interface
```

plus a few other directories containing ancillary data (bbn/) or interfaced codes (hyrec/, external_Pk/)

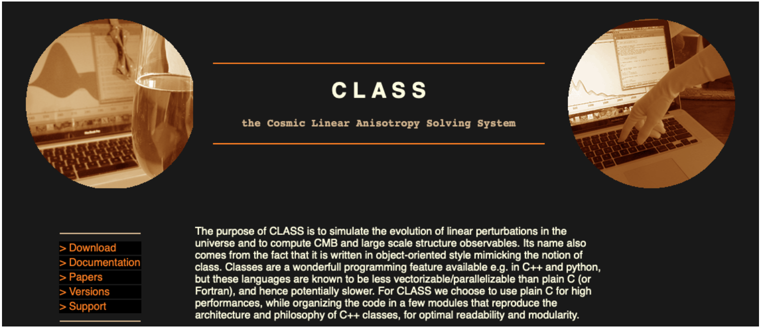
Where to find information and tutorials?

1. Basic information and links:
   - in `explanatory.ini` (but don't use this one in your runs, create your own:

```
# *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*
# *  CLASS input parameter file  *
# *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*

# This example of input file, intended for CLASS beginners, lists all
# possibilities with detailed comments.  You can use a more concise version, in
# which only the arguments in which you are interested would appear.  Only
# lines containing an equal sign not preceded by a sharp sign "#" are
# considered by the code, any other line is considered as a comment.
#
# The normal syntax is:      parameter = value(s)
# where white spaces do not matter (they are removed automatically
# by the parser unless they are part of the parameter name).
# However,                   'parameter' = value(s)
# and                        "parameter" = value(s)
# are also accepted by the parser since v2.8.0
#
# Input files must have an extension ".ini".

# ----------------------------
# ----> background parameters:
# ----------------------------

# 1) Hubble parameter : either 'H0' in km/s/Mpc or 'h' or '100*theta_s' where the
#   latter is the peak scale parameter 100(ds_dec/da_dec) close to 1.042143
#   (default: 'h' set to 0.67556)

#H0 = 67.556
h =0.67556
#100*theta_s = 1.042143

# 2) photon density: either 'T_cmb' in K or 'Omega_g' or 'omega_g' (default:
#   'T_cmb' set to 2.7255)

T_cmb = 2.7255
#Omega_g =
#omega_g =
```

Where to find information and tutorials?

1. Basic information and links:
   - in the historical `class` webpage `http://class-code.net`:
     - link to `installation wiki page`
     - summary of new features in each release
     - link to `slides from CLASS-dedicated courses` in which some are basic (see next page)
     - link to `online html documentation` in which the first subsection are basic (see in 2 pages)



1. Basic information and links:
   - in the CLASS courses slides and videos (linked from http://class-code.net or from https://lesgourg.github.io)

Courses

Academic courses:

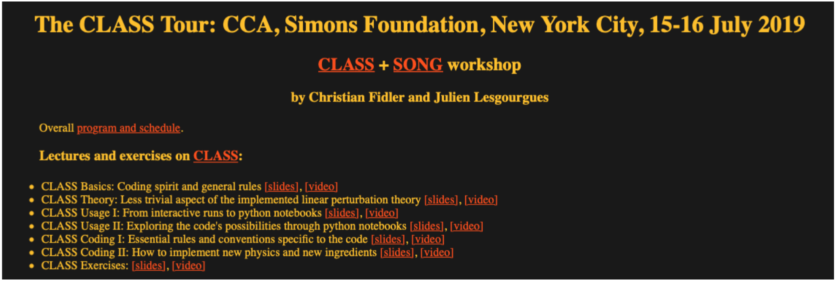
- Master course on "Relativity and Cosmology" given at RWTH Aachen University, winter 2015-2016, 2016-2017, 2017-2018 (access only with RWTH L2P account)
- Master course on "The Perturbed Universe" given at RWTH Aachen University, summer 2016, 2017, 2018 (access only with RWTH L2P account)
- Master course on "Cosmology" given at the University of Bern, Spring Semester 2015
- TASI lectures on Cosmological Perturbations given at the University of Colorado, June 2012
- doctoral level course on "Advanced Cosmology" given at EPFL in the frame of EDPY, spring 2009, 2011, 2013 (on-line notes are from 2009)
- Master course on "Cosmology" given at ENS Lyon, yearly, from winter 2007-2008 to winter 2010-2011
- undergraduate level course "Introduction to Cosmology" given at the CERN student summer school, summer 2002 to 2005
- doctoral level course on "Dark Matter and Dark Energy" given at EPFL in the frame of the *Ecole Doctorale de Suisse Romande*, autumn 2008, in collaboration with Celine Boehm (LAPTH, Annecy)
- doctoral level course on "Inflationary Cosmology" given at EPFL in the frame of the *Ecole Doctorale de Suisse Romande*, spring 2006
- beginning of a doctoral level course on "Théorie linéaire des Perturbations Cosmologiques" given at LAPTH in 2000-2001

Courses on numerical tools:

- The CLASS Tour: lecture series on CLASS and Monte Python. Given by Benjamin Audren (all until 2015), Christian Fidler (New York), Deanna Hooper (Cambridge), Julien Lesgourgues (all), Jesus Torrado (London), Thomas Tram (all but Cambridge), Miguel Zumalacarregui (Munich), at:
  - [CCA, Simons Foundation, New York, 15-16 July 2019] clik the title to get the lecture slides, videos and exercises on CLASS and SONG (no MontePython in this edition)
  - [Kavli Institute for Cosmology, Cambridge, 11-13 Septembre 2018] including CLASS lectures on basics, notebooks and coding, CLASS exercises, MontePython lecture and exercises (credits Thejs Brinckmann).
  - [Kavli IPMU, Tokyo, 27-31 Octobre 2014] *(most detailed version, especially on underlying physics; includes exercises)*
  - [Barcelona, 6-10 Octobre 2014] *(available on request)*
  - [King's & UCL, London, 11-16 May 2014]
  - [Uni. Geneva, 31 March - 03 April 2014] *(available on request)*
  - [MPA, Munich, 17-21 March 2014] *(available on request)*
  - [UNAM, Mexico City, 14-17 Octobre 2013] *(available on request)*
- lecture on CLASS from the [Darmouth-TRIUMF-U. of Washington HEP/Cosmology Tools Bootcamp, October 2017] *(general introduction to the code and presentation of eleven python scripts and notebooks illustrating most functionalities; outdated by Cambridge lectures on basics and notebooks, see above))*

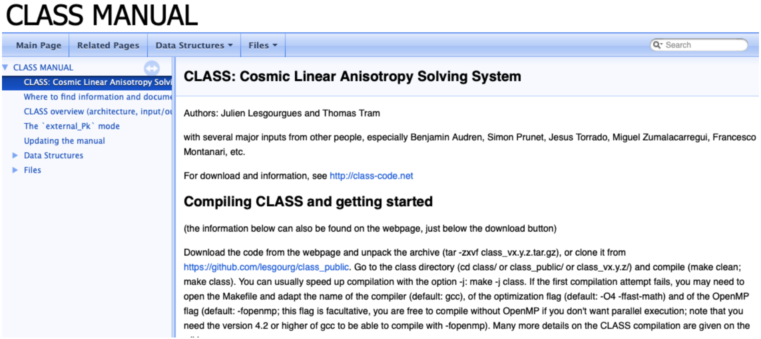
Where to find information and tutorials?

1. Basic information and links:
   - in the CLASS courses slides and videos (linked from http://class-code.net or from https://lesgourg.github.io)
     - Most up to date:
       [CCA, Simons Foundation, New York, 15-16 July 2019]
       see CLASS Basic, CLASS Usage. Includes videos.



Where to find information and tutorials?

1. Basic information and links:
   - in the online html documentation (linked from class-code.net or from github.com/lesgourg/class_public → Wiki or from the copy in your directory `class/doc/manual/html/index.html` or in the PDF `class/doc/manual/CLASS_MANUAL.pdf`), first three sections:
     - CLASS
     - Where to find information and documentation?
     - CLASS overview → updated version of the CLASS I 2011 paper



CLASS MANUAL

Main Page | Related Pages | Data Structures | Files | Search

- CLASS MANUAL
  - CLASS: Cosmic Linear Anisotropy Solvi
  - Where to find information and docume
  - CLASS overview (architecture, input/o
  - The `external_Pk` mode
  - Updating the manual
- Data Structures
- Files

CLASS: Cosmic Linear Anisotropy Solving System

Authors: Julien Lesgourgues and Thomas Tram

with several major inputs from other people, especially Benjamin Audren, Simon Prunet, Jesus Torrado, Miguel Zumalacarregui, Francesco Montanari, etc.

For download and information, see http://class-code.net

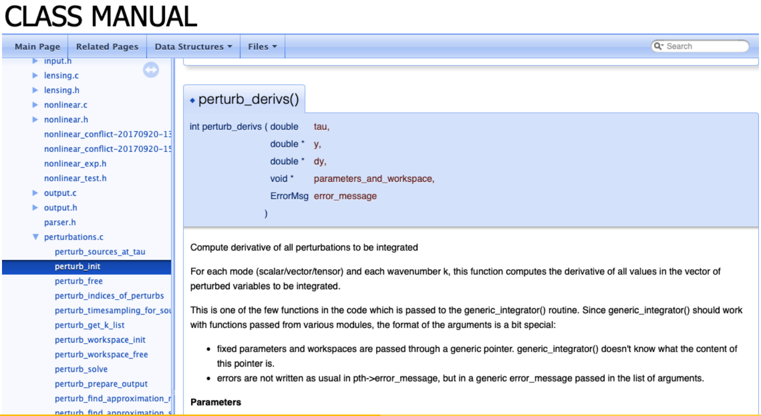
Compiling CLASS and getting started

(the information below can also be found on the webpage, just below the download button)

Download the code from the webpage and unpack the archive (tar -zxvf class_vx.y.z.tar.gz), or clone it from https://github.com/lesgourg/class_public. Go to the class directory (cd class/ or class_public/ or class_vx.y.z/) and compile (make clean; make class). You can usually speed up compilation with the option -j: make -j class. If the first compilation attempt fails, you may need to open the Makefile and adapt the name of the compiler (default: gcc), of the optimization flag (default: -O4 -ffast-math) and of the OpenMP flag (default: -fopenmp; this flag is facultative, you are free to compile without OpenMP if you don't want parallel execution; note that you need the version 4.2 or higher of gcc to be able to compile with -fopenmp). Many more details on the CLASS compilation are given on the

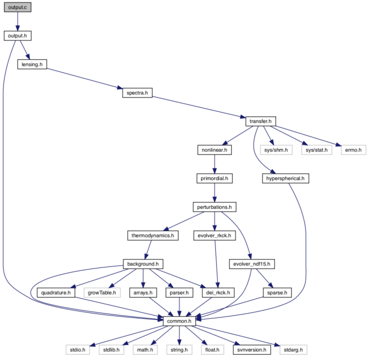
Where to find information and tutorials?

1. More advanced:
   - link to slides from CLASS-dedicated courses: New York's CLASS Coding slides, all Tokyo slides
   - full automatically-generated documentation (including dependence trees) on the online html documentation, in the last sections: Data Structures, Files.



Where to find information and tutorials?

1. More advanced:
   - link to `slides from CLASS-dedicated courses`: New York's CLASS Coding slides, all Tokyo slides
   - full automatically-generated documentation (including dependence trees) on the `online html documentation`, in the last sections: Data Structures, Files.
   - same information in your `class/doc/manual/CLASS_MANUAL.pdf`



Input/output specific to Large Scale Structure

Parameters for $P_L(k, z)$ and/or $P_{NL}(k, z)$ and/or $\sigma(R, z)$ (check details in `explanatory.ini`)

Input/output specific to Large Scale Structure

Parameters for $P_L(k,z)$ and/or $P_{NL}(k,z)$ and/or $\sigma(R,z)$ (check details in `explanatory.ini`)

```
output = ... , mPk, ...

P_k_max_h/Mpc = 1.
#P_k_max_1/Mpc = 0.7
z_pk = 0., 1.2, 3.5

non linear = Halofit # or none or HMcode (>=2.8)
# for HMcode baryonic feedback model:
feedback model = emu_dmonly # or many other options  (>=2.8)
```

If $\Omega_\nu \neq 0$ (massive neutrinos) the code will automatically compute two versions of each $P(k,z)$ and $\sigma(R,z)$: total matter (`_m`) for weak lensing and baryon+CDM (`_cb`) for galaxy correlation.

Input/output specific to Large Scale Structure

Parameters for $P_L(k,z)$ and/or $P_{NL}(k,z)$ and/or $\sigma(R,z)$ (check details in `explanatory.ini`)

```
output = ... , mPk, ...

P_k_max_h/Mpc = 1.
#P_k_max_1/Mpc = 0.7
z_pk = 0., 1.2, 3.5

non linear = Halofit # or none or HMcode (>=2.8)
# for HMcode baryonic feedback model:
feedback model = emu_dmonly # or many other options  (>=2.8)
```

If $\Omega_\nu \neq 0$ (massive neutrinos) the code will automatically compute two versions of each $P(k,z)$ and $\sigma(R,z)$: total matter (`_m`) for weak lensing and baryon+CDM (`_cb`) for galaxy correlation.
From python notebook or script: extract information with functions defined in `python/cclassy.pyx`:

```
pk(), pk_cb(), pk_lin(), pk_lin_cb(),
get_pk(), get_pk_cb(), get_pk_lin(), get_pk_lin_cb(),
get_pk_array(), get_pk_cb_array(),
get_pk_and_k_and_z(),
sigma(), sigma_cb(),
sigma8(), sigma8_cb()
```

Input/output specific to Large Scale Structure

Parameters for C_l^{XY} of number count, cosmic shear, or their cross-correlation (check details in `explanatory.ini`):

Input/output specific to Large Scale Structure

Parameters for C_l^{XY} of number count, cosmic shear, or their cross-correlation (check details in `explanatory.ini`):

```
output = ... , nCl , sCl , ...

l_max_lss = 600

selection=gaussian # or tophat , dirac
selection_mean = 0.9, 2, 1.1 # mean redshift in each bin
selection_width = 0.1 #redshift width of each bin (1 or N)
selection_bias = # see definition in CLASSgal paper
selection_magnification_bias = # see def in CLASSgal paper
non_diagonal=4 # depth of cross-correlation between bins

dNdz_selection = # window function W(z), analytic / file
dNdz_evolution = # source evolution , analytic / file

non linear = Halofit # or none or HMcode (>=2.8)

number count contributions = density , rsd , lensing , gr # see
     definition in CLASSgal paper , default: density only
```

Input/output specific to Large Scale Structure

Parameters for C_l^{XY} of number count, cosmic shear, or their cross-correlation (check details in `explanatory.ini`):

When running in a terminal, the output file header would be (for 3 bins with `non_diagonal = 3`):

```
# dimensionless total [l(l+1)/2pi] C_l's
#
# -> for galaxy lensing (lens[i]), these are C_l^phi-phi for
     the lensing potential.
#    Remember the conversion factors:
#    C_l^dd (deflection) = l(l+1) C_l^phi-phi
#    C_l^gg (shear/convergence) = 1/4 (l(l+1))^2 C_l^phi-phi
#
# 1:l      2:dens[1]-dens[1]         3:dens[1]-dens[2]
   4:dens[1]-dens[3]          5:dens[2]-dens[2]          6:dens
   [2]-dens[3]          7:dens[3]-dens[3]          8:lens[1]-
   lens[1]          9:lens[1]-lens[2]          10:lens[1]-lens
   [3]          11:lens[2]-lens[2]          12:lens[2]-lens[3]
        13:lens[3]-lens[3]          14:dens[1]-lens[1]
   15:dens[1]-lens[2]          16:dens[1]-lens[3]          17:
   dens[2]-lens[1]          18:dens[2]-lens[2]          19:dens
   [2]-lens[3]          20:dens[3]-lens[1]          21:dens[3]-
   lens[2]          22:dens[3]-lens[3]
```

Input/output specific to Large Scale Structure

Parameters for C_l^{XY} of number count, cosmic shear, or their cross-correlation (check details in `explanatory.ini`):

From python notebook or script: extract information with functions defined in `python/cclassy.pyx`:

```
density_cl()
```

$\rightarrow$ returns dictionary with keys `'ell'`, `'dd'`, `'ll'`, `'dl'`.

E.g. if ≥ 2 bins and `nondiagonal` ≥ 2; after `cls = cosmo.density_cl()`:

- the l values are in `cls['ell']`,
- the density C_l's of the first $\times$ second bin are in `cls['dd'][1]`,
- the shear C_l's of the first $\times$ first bin are in `cls['ll'][0]`,
- the cross C_l's of density (1st bin) $\times$ lensing (2nd bin) are in `cls['ld'][1]`.

Input/output specific to Large Scale Structure

Parameters for C_l^{XY} of number count, cosmic shear, or their cross-correlation (check details in `explanatory.ini`):

Important precision parameters (all such parameters exposed in `include/common.h` until 2.7, `include/precisions.h` from 2.8): $l_{\rm limber}/z_{\rm mean}$:

```
class_precision_parameter(
    l_switch_limber_for_nc_local_over_z,double,100.0) /**<
    when to use the Limber approximation for local number
    count contributions to cl's (relative to central
    redshift of each bin) */

class_precision_parameter(l_switch_limber_for_nc_los_over_z,
    double,30.0) /**< when to use the Limber approximation
    for number count contributions to cl's integrated along
    the line-of-sight (relative to central redshift of each
    bin) */
```

Set to very high numbers ($\mathcal{O}(10^4)$) for never using Limber... but then code becomes very slow.

Developpements related to LSS calculation performance

Two bottlenecks for CMB and LSS calculations:

1. `perturbations.c`: integration of system of $\mathcal{O}(20-50)$ coupled differential equations for each k. Not parallelisable beyond k loop.
2. `transfers.c`: line-of-sight integrals for harmonic transfer functions,

$$\Delta_\ell^X(k) = \int_\epsilon^{\tau_0} d\tau\ S^X(\tau,k)\ j_l(k(\tau_0-\tau))\ .$$

Argument has damped oscillations, slowly converging.
LSS $\rightarrow$ Scales like (number of redshift bins)^2

Second bottleneck: Beyond the line-of-sight integral

Nils Schöneberg, Marko Simonović, JL, Matias Zaldarriaga 1807.09540

For LSS observables:

$$C_\ell^{\alpha\beta,ij} = \sum_n \int_0^\infty dt\, I_\ell\left(\nu_n, t\right) \int_0^\infty d\chi\, W^i(\chi) W^j(\chi t) c_n^{\alpha\beta}(\chi, \chi t) \chi^{1-\nu_n} \;, \qquad (1)$$

$I_\ell\left(\nu_n, t\right)$ = precomputed cosmology-independent hypergeometric functions,
$W^i(\chi)$ = window functions for number count / lensing C_ℓ's,
$c_n^{\alpha\beta}(\chi, \chi t)$ = FFTlog expansion of $\mathcal{P}_\mathcal{R}(k) T_\alpha(k, \chi_1) T_\beta(k, \chi_2)$.

Speed up of `transfer.c` by 2-3 orders of magnitude (not for CMB: source functions not approximately separable in k and τ).

Since October: Released as separate branch `class_matter` on `github.com/lesgourg/class_public`.

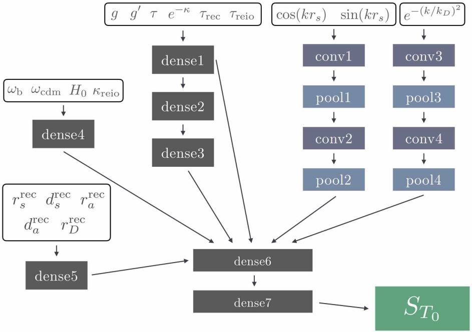
Questions: propagate this into main code, or an alternative scheme?

First bottleneck: CosmicNet

Jasper Albers, Christian Fidler, JL, Nils Schöneberg, Jesus Torrado 1907.05764

Neural networks predict source functions (instead of ODE) *with analytical approximations in the input*

$$\{\tau, \text{cosmological params., approx.}\} \xrightarrow{\mathrm{NN}^X} S^X(k_i, \tau)$$



Needs retraining when increasing number of cosmo. parameters, but not all of them!
Retraining doable on 4 cores in half a day.
Future: growth of a network repository ensured by automatic data exchange between users worldwide (CosmicNet).