CLASS MANUAL

Last updated September 29, 2021

Generated by Doxygen 1.9.1

Chapter 1

CLASS: Cosmic Linear Anisotropy Solving System

Authors: Julien Lesgourgues, Thomas Tram, Nils Schoeneberg

with several major inputs from other people, especially Benjamin Audren, Simon Prunet, Jesus Torrado, Miguel Zumalacarregui, Francesco Montanari, Deanna Hooper, Samuel Brieden, Daniel Meinert, Matteo Lucca, etc.

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

1.1 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 wiki page

https://github.com/lesgourg/class_public/wiki/Installation

(in particular, for compiling on Mac >= 10.9 despite of the clang incompatibility with OpenMP).

To check that the code runs, type:

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

The explanatory.ini file is THE reference input file, containing and explaining the use of all possible input parameters. We recommend to read it, to keep it unchanged (for future reference), and to create for your own purposes some shorter input files, containing only the input lines which are useful for you. Input files must have a $*$.ini extension. We provide an example of an input file containing a selection of the most used parameters, default.ini, that you may use as a starting point.

If you want to play with the precision/speed of the code, you can use one of the provided precision files (e.g. cl_permille.pre) or modify one of them, and run with two input files, for instance:

```
./class test.ini cl_permille.pre
```

The files $*$.pre are suppposed to specify the precision parameters for which you don't want to keep default values. If you find it more convenient, you can pass these precision parameter values in your $*$.ini file instead of an additional $*$.pre file.

The automatically-generated documentation is located in

```
doc/manual/html/index.html
doc/manual/CLASS_manual.pdf
```

On top of that, if you wish to modify the code, you will find lots of comments directly in the files.

1.2 Python

To use CLASS from python, or ipython notebooks, or from the Monte Python parameter extraction code, you need to compile not only the code, but also its python wrapper. This can be done by typing just 'make' instead of 'make class' (or for speeding up: 'make -j'). More details on the wrapper and its compilation are found on the wiki page

https://github.com/lesgourg/class_public/wiki

1.3 Plotting utility

Since version 2.3, the package includes an improved plotting script called CPU.py (Class Plotting Utility), written by Benjamin Audren and Jesus Torrado. It can plot the Cl's, the P(k) or any other CLASS output, for one or several models, as well as their ratio or percentage difference. The syntax and list of available options is obtained by typing 'pyhton CPU.py -h'. There is a similar script for MATLAB, written by Thomas Tram. To use it, once in MATLAB, type 'help plot_CLASS_output.m'

1.4 Developing the code

If you want to develop the code, we suggest that you download it from the github webpage

https://github.com/lesgourg/class_public

rather than from class-code.net. Then you will enjoy all the feature of git repositories. You can even develop your own branch and get it merged to the public distribution. For related instructions, check

https://github.com/lesgourg/class_public/wiki/Public-Contributing

1.5 Using the code

You can use CLASS freely, provided that in your publications, you cite at least the paper CLASS II: Approximation schemes < http://arxiv.org/abs/1104.2933>. Feel free to cite more CLASS papers!

1.6 Support

To get support, please open a new issue on the

https://github.com/lesgourg/class_public

webpage!

Chapter 2

Where to find information and documentation on CLASS?

Author: Julien Lesgourgues

- **For what the code can actually compute**: all possible input parameters, all coded cosmological models, all functionalities, all observables, etc.: read the file `explanatory.ini` in the main `CLASS` directory: it is THE reference file where we keep track of all possible input and the definition of all input parameters. For that reason we recommend to leave it always unchanged and to work with copies of it, or with short input files written from scratch.
- **For the structure, style, and concrete aspects of the code**: this documentation, especially the `CLASS overview` chapter (the extensive automatically-generated part of this documentation is more for advanced users); plus the slides of our `CLASS` lectures, for instance those from New York 2019 available at

  `https://lesgourg.github.io/class-tour-NewYork.html`

  An updated overview of available `CLASS` lecture slides is always available at

  `http://lesgourg.github.io/courses.html`

  in the section `Courses on numerical tools`.
- **For the python wrapper of `CLASS`**: at the moment, the best are the "Usage I" and "Usage II" slides of the New York 2019 course,

  `https://lesgourg.github.io/class-tour-NewYork.html`
- **For the physics and equations used in the code**: mainly, the following papers:
  - *Cosmological perturbation theory in the synchronous and conformal Newtonian gauges*
    C. P. Ma and E. Bertschinger.
    `http://arxiv.org/abs/astro-ph/9506072`
    10.1086/176550
    Astrophys. J. **455**, 7 (1995)
  - *The Cosmic Linear Anisotropy Solving System (CLASS) II: Approximation schemes*
    D. Blas, J. Lesgourgues and T. Tram.
    `http://arxiv.org/abs/1104.2933` [astro-ph.CO]
    10.1088/1475-7516/2011/07/034
    JCAP **1107**, 034 (2011)
  - *The Cosmic Linear Anisotropy Solving System (CLASS) IV: efficient implementation of non-cold relics*
    J. Lesgourgues and T. Tram.
    `http://arxiv.org/abs/1104.2935` [astro-ph.CO]
    10.1088/1475-7516/2011/09/032
    JCAP **1109**, 032 (2011)

- *Optimal polarisation equations in FLRW universes*
  T. Tram and J. Lesgourgues.
  http://arxiv.org/abs/1305.3261 [astro-ph.CO]
  10.1088/1475-7516/2013/10/002
  JCAP **1310**, 002 (2013)
- *Fast and accurate CMB computations in non-flat FLRW universes*
  J. Lesgourgues and T. Tram.
  http://arxiv.org/abs/1312.2697 [astro-ph.CO]
  10.1088/1475-7516/2014/09/032
  JCAP **1409**, no. 09, 032 (2014)
- *The CLASSgal code for Relativistic Cosmological Large Scale Structure*
  E. Di Dio, F. Montanari, J. Lesgourgues and R. Durrer.
  http://arxiv.org/abs/1307.1459 [astro-ph.CO]
  10.1088/1475-7516/2013/11/044
  JCAP **1311**, 044 (2013)
- *The synergy between CMB spectral distortions and anisotropies*
  M. Lucca, N. Schöneberg, D. C. Hooper, J. Lesgourgues, J. Chluba.
  http://arxiv.org/abs/1910.04619 [astro-ph.CO]
  JCAP 02 (2020) 026
- *Optimal Boltzmann hierarchies with nonvanishing spatial curvature*
  C. Pitrou, T. S. Pereira, J. Lesgourgues,
  http://arxiv.org/abs/2005.12119 [astro-ph.CO]
  Phys.Rev.D 102 (2020) 2, 023511

plus also some latex notes on specific sectors:

- *Equations for perturbed recombination*
  (can be turned on optionally by the user since v2.1.0)
  L. Voruz.
  http://lesgourg.github.io/class_public/perturbed_recombination.pdf
- *PPF formalism in Newtonian and synchronous gauge*
  (used by default for the fluid perturbations since v2.6.0)
  T. Tram.
  http://lesgourg.github.io/class_public/PPF_formalism.pdf

Chapter 3

CLASS overview (architecture, input/output, general principles)

Author: Julien Lesgourgues

3.1 Overall architecture of <tt>class</tt>

3.1.1 Files and directories

After downloading `CLASS`, one can see the following files in the root directory contains:

- some example of input files, the most important being `explanatory.ini`. a reference input file containing all possible flags, options and physical input parameters. While this documentation explains the structure and use of the code, `explanatory.ini` can be seen as the *physical* documentation of `CLASS`. We also provide a few other input files ending with `.ini` (a concise input file `default.ini` with the most used parameters, and two files corresponding to the best-fit baseline models of respectively Planck 2015 and 2018), and a few precision input files (ending with `.pre`)
- the `Makefile`, with which you can compile the code by typing `make clean; make -j;` this will create the executable `class` and some binary files in the directory `build/`. The `Makefile` contains other compilation options that you can view inside the file.
- `CPU.py` is a python script designed for plotting the `CLASS` output; for documentation type `python CPU.py --help`
- `plot_CLASS_output.m` is the counterpart of `CPU.py` for MatLab
- there are other input files for various applications: an example of a non-cold dark matter distribution functions (`psd_FD_single.dat`), and examples of evolution and selection functions for galaxy number count observables (`myevolution.dat, myselection.dat`).

Other files are split between the following directories:

- `source/` contains the C files for each main `CLASS` module, i.e. each block containing some part of the physical equations and logic of the Boltzmann code.

- `tools/` contains purely numerical algorithms, applicable in any context: integrators, simple manipulation of arrays (derivation, integration, interpolation), Bessel function calculation, quadrature algorithms, parser, etc.
- `main/` contains the main module `class.c` with the main routine `class(...)`, to be used in interactive runs (but not necessarily when the code is interfaced with other ones).
- `test/` contains alternative main routines which can be used to run only some part of the code, to test its accuracy, to illustrate how it can be interfaced with other codes, etc.
- `include/` contains all the include files with a `.h` suffix.
- `output/` is where the output files will be written by default (this can be changed to another directory by adjusting the input parameter `root = <...>`)
- `python/` contains the python wrapper of `CLASS`, called classy (see `python/README`)
- `cpp/` contains the C++ wrapper of `CLASS`, called ClassEngine (see `cpp/README`)
- `doc/` contains the automatic documentation (manual and input files required to build it)
- `external/` contains auxiliary or external algorithms used by `CLASS`, in particular:
- `external/hyrec/` contains the recombination code HyRec (Lee and Ali-Haimoud 2020) that solves the recombinaiton equations by default.
- `external/RecfastCLASS/` contains an modified version of the recombinaiton code RECFAST v1.5. It can be used as an alternative to solve the recombinaiton equations (with 'recombination=recfast').
- `external/heating/` contains additional peices of code and interpolation tables for the calculation of energy injection (relevant for CMB anisotropies and spectral distorsions).
- `external/distortions/` contains interpolation tables relevant for the calculation of CMB spectral distortions with J.Chluba's Green function method.
- `external/bbn/` contains interpolation tables produced by BBN codes, in order to predict e.g. $Y_{\rm He}(\omega_b, \Delta N_{\rm eff})$.
- `external/external_Pk/` contains examples of external codes that can be used to generate the primordial spectrum and be interfaced with `CLASS`, when one of the many options already built inside the code are not sufficient.
- `external/RealSpaceInterface` contains a software that uses `CLASS` to plot the eockution of linear perturabtions in reals space, with a graphical interface (credits M. Beutelspacher and G. Samaras).

3.1.2 The ten-module backbone

3.1.2.1 Ten tasks

The purpose of `class` consists in computing some background quantities, thermodynamical quantities, perturbation transfer functions, and finally 2-point statistics (power spectra) for a given set of cosmological parameters. This task can be decomposed in few steps or modules:

1. set input parameter values.
2. compute the evolution of cosmological background quantities.
3. compute the evolution of thermodynamical quantities (ionization fractions, etc.)
4. compute the evolution of source functions $S(k,\tau)$ (by integrating over all perturbations).
5. compute the primordial spectra.

6. compute the linear and non-linear 2-point statistics in Fourier space (i.e. the power spectra $P(k)$).
7. compute the transfer functions in harmonic space $\Delta_l(k)$.
8. compute the linear and non-linear 2-point statistics in harmonic space (harmonic power spectra C_l's).
9. compute the lensed CMB spectra (using second-order perturbation theory).
10. compute the CMB spectral distortions.

(11. The results can optionally be written in files when `CLASS` is used interactively. The python wrapper does not go through this step.)

3.1.2.2 Ten structures

In `class`, each of these steps is associated with a structure:

1. `struct precision` for input precision parameters (input physical parameters are dispatched among the other structures listed below)
2. `struct background` for cosmological background,
3. `struct thermodynamics` for thermodynamics,
4. `struct perturbations` for source functions,
5. `struct primordial` for primordial spectra,
6. `struct fourier` for Fourier spectra,
7. `struct transfer` for transfer functions,
8. `struct harmonic` for harmonic spectra,
9. `struct lensing` for lensed CMB spectra,
10. `struct distortions` for CMB spectral distortions,

(11. `struct output` for auxiliary variable describing the output format.)

A given structure contains "everything concerning one step that the subsequent steps need to know" (for instance, `struct perturbations` contains everything about source functions that the transfer module needs to know). In particular, each structure contains one array of tabulated values (for `struct background`, background quantities as a function of time, for `struct thermodynamics`, thermodynamical quantities as a function of redshift, for `struct perturbations`, sources $S(k,\tau)$, etc.). It also contains information about the size of this array and the value of the index of each physical quantity, so that the table can be easily read and interpolated. Finally, it contains any derived quantity that other modules might need to know. Hence, the communication from one module A to another module B consists in passing a pointer to the structure filled by A, and nothing else.

All "precision parameters" are grouped in the single structure `struct precision`. The code contains *no other arbitrary numerical coefficient*.

3.1.2.3 Ten modules

Each structure is defined and filled in one of the following modules (and precisely in the order below):

1. input.c
2. background.c
3. thermodynamics.c
4. perturbations.c
5. primordial.c
6. fourier.c
7. transfer.c
8. harmonic.c
9. lensing.c
10. distortions.c

(11. output.c)

Each of these modules contains at least three functions:

- module_init(...)
- module_free(...)
- module_something_at_somevalue

where *module* is one of input, background, thermodynamics, perturb, primordial, nonlinear, transfer, spectra, lensing, distortions, output.

The first function allocates and fills each structure. This can be done provided that the previous structures in the hierarchy have been already allocated and filled. In summary, calling one of module_init(...) amounts in solving entirely one of the steps 1 to 10.

The second function deallocates the fields of each structure. This can be done optionally at the end of the code (or, when the code is embedded in a sampler, this **must** be done between each execution of class, and especially before calling module_init(...) again with different input parameters).

The third function is able to interpolate the pre-computed tables. For instance, background_init() fills a table of background quantities for discrete values of conformal time τ, but background_at_tau(tau, * values) will return these values for any arbitrary τ.

Note that functions of the type module_something_at_somevalue are the only ones which are called from another module, while functions of the type module_init(...) and module_free(...) are the only one called by the main executable. All other functions are for internal use in each module.

When writing a C code, the ordering of the functions in the $*$.c file is in principle arbitrary. However, for the sake of clarity, we always respected the following order in each CLASS module:

1. all functions that may be called by other modules, i.e. "external functions", usually named like module_↩ something_at_somevalue(...)
2. then, module_init(...)
3. then, module_free()
4. then, all functions used only internally by the module

3.1.3 The <tt>main()</tt> function(s)

3.1.3.1 The <tt>main.c</tt> file

The main executable of `class` is the function `main()` located in the file `main/main.c`. This function consist only in the following lines (not including comments and error-management lines explained later):

```
main() {
 struct precision pr;

 struct background ba;

 struct thermodynamics th;

 struct perturbations pt;

 struct primordial pm;

 struct fourier fo;

 struct transfer tr;

 struct harmonic hr;

 struct lensing le;

 struct distortions sd;

 struct output op;


 input_init(argc, argv,&pr,&ba,&th,&pt,&tr,&pm,&hr,&fo,&le,&sd,&op,errmsg);

 background_init(&pr,&ba);

 thermodynamics_init(&pr,&ba,&th);

 perturbations_init(&pr,&ba,&th,&pt);

 primordial_init(&pr,&pt,&pm);

 fourier_init(&pr,&ba,&th,&pt,&pm,&fo);

 transfer_init(&pr,&ba,&th,&pt,&fo,&tr);

 harmonic_init(&pr,&ba,&pt,&pm,&fo,&tr,&hr);

 lensing_init(&pr,&pt,&hr,&fo,&le);

 distortions_init(&pr,&ba,&th,&pt,&pm,&sd);

 output_init(&ba,&th,&pt,&pm,&tr,&hr,&fo,&le,&op)


 /****** done ******/


 distortions_free(&sd);

 lensing_free(&le);

 harmonic_free(&hr);

 transfer_free(&tr);

 fourier_free(&fo);

 primordial_free(&pm);

 perturbations_free(&pt);
```

```
thermodynamics_free(&th);

background_free(&ba);
```

We can come back on the role of each argument. The arguments above are all pointers to the 10 structures of the code, excepted `argc, argv` which contains the input files passed by the user, and `errmsg` which contains the output error message of the input module (error management will be described below).

`input_init_from_arguments` needs all structures, because it will set the precision parameters inside the `precision` structure, and the physical parameters in some fields of the respective other structures. For instance, an input parameter relevant for the primordial spectrum calculation (like the tilt n_s) will be stored in the `primordial` structure. Hence, in `input_init_from_arguments`, all structures can be seen as output arguments.

Other `module_init()` functions typically need all previous structures, which contain the result of the previous modules, plus its own structures, which contain some relevant input parameters before the function is called, as well as all the result form the module when the function has been executed. Hence all passed structures can be seen as input argument, excepted the last one which is both input and output. An example is `perturbations_init(&pr,&ba,&th,&pt)`.

Each function `module_init()` does not need **all** previous structures, it happens that a module does not depend on a **all** previous one. For instance, the primordial module does not need information on the background and thermodynamics evolution in order to compute the primordial spectra, so the dependency is reduced: `primordial_init(&pr,&pt,&pm)`.

Each function `module_init()` only deallocates arrays defined in the structure of their own module, so they need only their own structure as argument. (This is possible because all structures are self-contained, in the sense that when the structure contains an allocated array, it also contains the size of this array). The first and last module, `input` and `output`, have no `input_free()` or `output_free()` functions, because the structures `precision` and `output` do not contain arrays that would need to be de-allocated after the execution of the module.

3.1.3.2 The <tt>test_<...>.c</tt> files

For a given purpose, somebody could only be interested in the intermediate steps (only background quantities, only the thermodynamics, only the perturbations and sources, etc.) It is then straightforward to truncate the full hierarchy of modules 1, ... 10 at some arbitrary order. We provide several "reduced executables" achieving precisely this. They are located in `test/test_module_.c` (like, for instance, `test/test_perturbations.c`) and they can be complied using the Makefile, which contains the appropriate commands and definitions (for instance, you can type `make test_perturbations`).

The `test/` directory contains other useful example of alternative main functions, like for instance `test_loops.c` which shows how to call `CLASS` within a loop over different parameter values. There is also a version `test/test_loops_omp.c` using a double level of openMP parallelisation: one for running several `CLASS` instances in parallel, one for running each `CLASS` instance on several cores. The comments in these files are self-explanatory.

3.2 Input/output

3.2.1 Input

There are two types of input:

- "precision parameters" (controlling the precision of the output and the execution time),
- "input parameters" (cosmological parameters, flags telling to the code what it should compute, ...)

The code can be executed with a maximum of two input files, e.g.

```
./class explanatory.ini cl_permille.pre
```

The file with a `.ini` extension is the cosmological parameter input file, and the one with a `.pre` extension is the precision file. Both files are optional: all parameters are set to default values corresponding to the "most usual choices", and are eventually replaced by the parameters passed in the two input files. For instance, if one is happy with default accuracy settings, it is enough to run with

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

Input files do not necessarily contain a line for each parameter, since many of them can be left to default value. The example file `explanatory.ini` is very long and somewhat indigestible, since it contains all possible parameters, together with lengthy explanations. We recommend to keep this file unchanged for reference, and to copy it in e.g. `test.ini`. In the latter file, the user can erase all sections in which he/she is absolutely not interested (e.g., all the part on isocurvature modes, or on tensors, or on non-cold species, etc.). Another option is to create an input file from scratch, copying just the relevant lines from `explanatory.ini`. For the simplest applications, the user will just need a few lines for basic cosmological parameters, one line for the `output` entry (where one can specifying which power spectra must be computed), and one line for the `root` entry (specifying the prefix of all output files).

The syntax of the input files is explained at the beginning of `explanatory.ini`. Typically, lines in those files look like:

```
parameter1 = value1

free comments

parameter2 = value2 # further comments

# commented_parameter = commented_value
```

and parameters can be entered in arbitrary order. This is rather intuitive. The user should just be careful not to put an "=" sign not preceded by a "#" sign inside a comment: the code would then think that one is trying to pass some unidentified input parameter.

The syntax for the cosmological and precision parameters is the same. It is clearer to split these parameters in the two files `.ini` and `.pre`, but there is no strict rule about which parameter goes into which file: in principle, precision parameters could be passed in the `.ini`, and vice-versa. The only important thing is not to pass the same parameter twice: the code would then complain and not run.

The `CLASS` input files are also user-friendly in the sense that many different cosmological parameter bases can be used. This is made possible by the fact that the code does not only read parameters, it "interprets them" with the level of logic which has been coded in the `input.c` module. For instance, the Hubble parameter, the photon density, the baryon density and the ultra-relativistic neutrino density can be entered as:

```
h = 0.7

T_cmb = 2.726      # Kelvin units

omega_b = 0.02

N_eff = 3.04
```

(in arbitrary order), or as

```
H0 = 70

omega_g = 2.5e-5     # g is the label for photons

Omega_b = 0.04

omega_ur = 1.7e-5    # ur is the label for ultra-relativistic species
```

or any combination of the two. The code knows that for the photon density, one should pass one (but not more than one) parameter out of `T_cmb`, `omega_g`, `Omega_g` (where small omega's refer to $\omega_i \equiv \Omega_i h^2$). It searches for one of these values, and if needed, it converts it into one of the other two parameters, using also other input parameters. For instance, `omega_g` will be converted into `Omega_g` even if h is written later in the file than `omega_g`: the order makes no difference. Lots of alternatives have been defined. If the code finds that not enough parameters have been passed for making consistent deductions, it will complete the missing information with in-built default values. On the contrary, if it finds that there is too much information and no unique solution, it will complain and return an error.

In summary, the input syntax has been defined in such way that the user does not need to think too much, and can pass his preferred set of parameters in a nearly informal way.

Let us mention a two useful parameters defined at the end of `explanatory.ini`, that we recommend setting to `yes` in order to run the code in a safe way:

`write parameters = [yes or no]` (*default*: `no`)

When set to yes, all input/precision parameters which have been read are written in a file `<root>parameters.ini`, to keep track all the details of this execution; this file can also be re-used as a new input file. Also, with this option, all parameters that have been passed and that the code did not read (because the syntax was wrong, or because the parameter was not relevant in the context of the run) are written in a file `<root>unused_parameters`. When you have doubts about your input or your results, you can check what is in there.

`write warnings = [yes or no]` (*default*: `no`)

When set to yes, the parameters that have been passed and that the code did not read (because the syntax was wrong, or because the parameter was not relevant in the context of the run) are written in the standard output as `[Warning:]....`

There is also a list of "verbose" parameters at the end of `explanatory.ini`. They can be used to control the level of information passed to the standard output (0 means silent; 1 means normal, e.g. information on age of the universe, etc.; 2 is useful for instance when you want to check on how many cores the run is parallelised; 3 and more are intended for debugging).

`CLASS` comes with a list of precision parameter files ending by `.pre`. Honestly we have not been updating all these files recently, and we need to do a bit of cleaning there. However you can trust `cl_ref.pre`. We have derived this file by studying both the convergence of the CMB output with respect to all `CLASS` precision parameters, and the agreement with `CAMB`. We consider that this file generates good reference CMB spectra, accurate up to the hundredth of per cent level, as explained in the CLASS IV paper and re-checked since then. You can try it with e.g.

```
./class explanatory.ini cl_ref.pre
```

but the run will be extremely long. This is an occasion to run a many-core machine with a lot of RAM. It may work also on your laptop, but in half an hour or so.

If you want a reference matter power spectrum P(k), also accurate up to the hundredth of percent level, we recommend using the file `pk_ref.pre`, identical to `cl_ref.pre` excepted that the truncation of the neutrino hierarchy has been pushed to `l_max_ur=150`.

In order to increase moderately the precision to a tenth of percent, without prohibitive computing time, we recommend using `cl_permille.pre`.

3.2.2 Output

The input file may contain a line

```
root = <root>
```

where <root> is a path of your choice, e.g. output/test_. Then all output files will start like this, e.g. output/test_cl.dat, output/test_cl_lensed.dat, etc. Of course the number of output files depends on your settings in the input file. There can be input files for CMB, LSS, background, thermodynamics, transfer functions, primordial spectra, etc. All this is documented in explanatory.ini.

If you do not pass explicitly a root = <root>, the code will name the output in its own way, by concatenating output/, the name of the input parameter file, and the first available integer number, e.g.

```
output/explanatory03_cl.dat, etc.
```

3.3 General principles

3.3.1 Error management

Error management is based on the fact that all functions are defined as integers returning either _SUCCESS_ or _FAILURE_. Before returning _FAILURE_, they write an error message in the structure of the module to which they belong. The calling function will read this message, append it to its own error message, and return a _FAILURE_; and so on and so forth, until the main routine is reached. This error management allows the user to see the whole nested structure of error messages when an error has been met. The structure associated to each module contains a field for writing error messages, called structure_i.error_message, where structure_i could be one of background, thermo, perturbs, etc. So, when a function from a module i is called within module j and returns an error, the goal is to write in structure_j.error_message a local error message, and to append to it the error message in structure_i.error_message. These steps are implemented in a macro class_call(), used for calling whatever function:

```
class_call(module_i_function(...,structure_i),
           structure_i.error_message,
           structure_j.error_message);
```

So, the first argument of call_call() is the function we want to call; the second argument is the location of the error message returned by this function; and the third one is the location of the error message which should be returned to the higher level. Usually, in the bulk of the code, we use pointer to structures rather than structure themselves; then the syntax is

```
class_call(module_i_function(...,pi),
       pi->error_message,
       pj->error_message);`
```

where in this generic example, pi and pj are assumed to be pointers towards the structures structure_i and structure_j.

The user will find in include/common.h a list of additional macros, all starting by class_...(), which are all based on this logic. For instance, the macro class_test() offers a generic way to return an error in a standard format if a condition is not fulfilled. A typical error message from CLASS looks like:

```
Error in module_j_function1
```

```
module_j_function1 (L:340) :  error in module_i_function2(...)

module_i_function2 (L:275) :  error in module_k_function3(...)

...

=> module_x_functionN (L:735) :  your choice of input parameter blabla=30
is not consistent with the constraint blabla<1
```

where the `L`'s refer to line numbers in each file. These error messages are very informative, and are built almost entirely automatically by the macros. For instance, in the above example, it was only necessary to write inside the function `module_x_functionN()` a test like:

```
class_test(blabla >= 1,
           px->error_message,
         "your choice of input parameter blabla=%e
is not consistent with the constraint blabla<%e",
           blabla,blablamax);
```

All the rest was added step by step by the various `class_call()` macros.

3.3.2 Dynamical allocation of indices

On might be tempted to decide that in a given array, matrix or vector, a given quantity is associated with an explicit index value. However, when modifying the code, extra entries will be needed and will mess up the initial scheme; the user will need to study which index is associated to which quantity, and possibly make an error. All this can be avoided by using systematically a dynamical index allocation. This means that all indices remain under a symbolic form, and in each, run the code attributes automatically a value to each index. The user never needs to know this value.

Dynamical indexing is implemented in a very generic way in CLASS, the same rules apply everywhere. They are explained in these lecture slides:

https://www.dropbox.com/sh/ma5muh76sggwk8k/AABl_DDUBEzAjjdywMjeTya2a?dl=0

in the folder `CLASS_Lecture_slides/lecture5_index_and_error.pdf`.

3.3.3 No hard coding

Any feature or equation which could be true in one cosmology and not in another one should not be written explicitly in the code, and should not be taken as granted in several other places. Discretization and integration steps are usually defined automatically by the code for each cosmology, instead of being set to something which might be optimal for minimal models, and not sufficient for other ones. You will find many example of this in the code. As a consequence, in the list of precision parameter, you rarely find actual stepsize. You find rather parameters representing the ratio between a stepsize and a physical quantity computed for each cosmology.

3.3.4 Modifying the code

Implementing a new idea completly from scratch would be rather intimidating, even for the main developpers of `CLASS`. Fortunately, we never have to work from scratch. Usually we want to code a new species, a new observable, a new approximation scheme, etc. The trick is to think of another species, observable, approximation scheme, etc., looking as close as possible to the new one.

Then, playing with the `grep` command and the `search` command of your editor, search for all occurences of this nearest-as-possible other feature. This is usually easy thanks to our naming scheme. For each species, observable, approximation scheme, etc., we usually use the same sequence of few letters everywhere (fo instance, `fld` for the fluid usually representing Dark Energy). Grep for `fld` and you'll get all the lines related to the fluid. There is another way: we use everywhere some conditional jumps related to a given feature. For instance, the lines related to the fluid are always in between `if (pba->has_fld == _TRUE_) { ... }` and the lines related to the cosmic shear observables are always in between `if (ppt->has_lensing_potential == _TRUE_) { ... }`. Locating these flags and conditional jumps shows you all the parts related to a given feature/ingredient.

Once you have localised your nearest-as-possible other feature, you can copy/paste these lines and adapt them to the case of your new feature! You are then sure that you didn't miss any step, even the smallest technical steps (definition of indices, etc.)

3.4 Units

Internally, the code uses almost everywhere units of Mpc to some power, excepted in the inflation module, where many quantities are in natural units (wrt the true Planck mass).

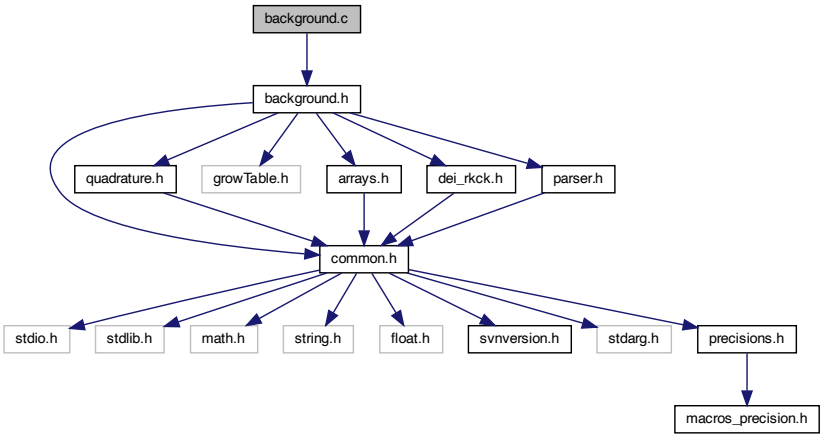
Chapter 4

File Documentation

4.1 background.c File Reference

```
#include "background.h"
```

Include dependency graph for background.c:



Functions

- int background_at_z (struct background *pba, double z, enum vecback_format return_format, enum interpolation_method inter_mode, int *last_index, double *pvecback)
- int background_at_tau (struct background *pba, double tau, enum vecback_format return_format, enum interpolation_method inter_mode, int *last_index, double *pvecback)
- int background_tau_of_z (struct background *pba, double z, double *tau)
- int background_z_of_tau (struct background *pba, double tau, double *z)
- int background_functions (struct background *pba, double a, double *pvecback_B, enum vecback_format return_format, double *pvecback)
- int background_w_fld (struct background *pba, double a, double *w_fld, double *dw_over_da_fld, double *integral_fld)
- int background_varconst_of_z (struct background *pba, double z, double *alpha, double *me)

- int background_init (struct precision *ppr, struct background *pba)
- int background_free (struct background *pba)
- int background_free_noinput (struct background *pba)
- int background_free_input (struct background *pba)
- int background_indices (struct background *pba)
- int background_ncdm_distribution (void *pbadist, double q, double *f0)
- int background_ncdm_test_function (void *pbadist, double q, double *test)
- int background_ncdm_init (struct precision *ppr, struct background *pba)
- int background_ncdm_momenta (double *qvec, double *wvec, int qsize, double M, double factor, double z, double *n, double *rho, double *p, double *drho_dM, double *pseudo_p)
- int background_ncdm_M_from_Omega (struct precision *ppr, struct background *pba, int n_ncdm)
- int background_checks (struct precision *ppr, struct background *pba)
- int background_solve (struct precision *ppr, struct background *pba)
- int background_initial_conditions (struct precision *ppr, struct background *pba, double *pvecback, double *pvecback_integration, double *loga_ini)
- int background_find_equality (struct precision *ppr, struct background *pba)
- int background_output_titles (struct background *pba, char titles[_MAXTITLESTRINGLENGTH_])
- int background_output_data (struct background *pba, int number_of_titles, double *data)
- int background_derivs (double loga, double *y, double *dy, void *parameters_and_workspace, ErrorMsg error_message)
- int background_sources (double loga, double *y, double *dy, int index_loga, void *parameters_and_↩ workspace, ErrorMsg error_message)
- int background_timescale (double loga, void *parameters_and_workspace, double *timescale, ErrorMsg error_message)
- int background_output_budget (struct background *pba)
- double V_e_scf (struct background *pba, double phi)
- double V_p_scf (struct background *pba, double phi)
- double V_scf (struct background *pba, double phi)

4.1.1 Detailed Description

Documented background module

- Julien Lesgourgues, 17.04.2011
- routines related to ncdm written by T. Tram in 2011
- new integration scheme written by N. Schoeneberg in 2020

Deals with the cosmological background evolution. This module has two purposes:

- at the beginning, to initialize the background, i.e. to integrate the background equations, and store all background quantities as a function of conformal time inside an interpolation table.
- to provide routines which allow other modules to evaluate any background quantity for a given value of the conformal time (by interpolating within the interpolation table), or to find the correspondence between redshift and conformal time.

The overall logic in this module is the following:

1. most background parameters that we will call {A} (e.g. rho_gamma, ..) can be expressed as simple analytical functions of the scale factor 'a' plus a few variables that we will call {B} (e.g. (phi, phidot) for quintessence, or some temperature for exotic particles, etc...). [Side note: for simplicity, all variables {B} are declared redundently inside {A}.]

2. in turn, quantities {B} can be found as a function of the the scale factor [or rather (a/a_0)] by integrating the background equations. Thus {B} also includes the density of species which energy conservation equation must be integrated explicitely, like the density of fluids or of decaying dark matter.
3. some other quantities that we will call {C} (like e.g. proper and conformal time, the sound horizon, the analytic scale-invariant growth factor) also require an explicit integration with respect to (a/a_0) [or rather log(a/a_p)], since they cannot be inferred analytically from (a/a_0) and parameters {B}. The difference between {B} and {C} parameters is that {C} parameters do not need to be known in order to get {A}.

So, we define the following routines:

- background_functions() returns all background quantities {A} as a function of (a/a_0) and of quantities {B}.
- background_solve() integrates the quantities {B} and {C} with respect to log(a/a_0); this integration requires many calls to background_functions().
- the result is stored in the form of a big table in the background structure. There is one column for the scale factor, and one for each quantity {A} or {C} [Side note: we don;t include {B} here because the {B} variables are already decalred redundently also as {A} quantitites.]

Later in the code:

- If we know the variables (a/a_0) + {B} and need some quantity {A} (but not {C}), the quickest and most precise way is to call directly background_functions() (for instance, in simple models, if we want H at a given value of the scale factor).
- If we know 'tau' and want any other quantity, we can call background_at_tau(), which interpolates in the table and returns all values.
- If we know 'z' but not the {B} variables, or if we know 'z' and we want {C} variables, we need to call background_at_z(), which interpolates in the table and returns all values.
- Finally, it can be useful to get 'tau' for a given redshift 'z' or vice-versa: this can be done with background_tau_of_z() or background_z_of_tau().

In order to save time, background_at_tau() ans background_at_z() can be called in three modes: short_info, normal_info, long_info (returning only essential quantities, or useful quantities, or rarely useful quantities). Each line in the interpolation table is a vector whose first few elements correspond to the short_info format; a larger fraction contribute to the normal format; and the full vector corresponds to the long format. The guideline is that short_info returns only geometric quantities like a, H, H'; normal format returns quantities strictly needed at each step in the integration of perturbations; long_info returns quantities needed only occasionally.

In summary, the following functions can be called from other modules:

1. background_init() at the beginning background_at_tau(),
2. background_at_z(), background_tau_of_z(), background_z_of_tau() at any later time
3. background_free() at the end, when no more calls to the previous functions are needed

For units and normalisation conventions, there are two guiding principles:

1) All quantities are expressed in natural units in which everything is in powers of Mpc, e.g.:

- t stands for (cosmological or proper time)$*$c in Mpc

- tau stands for (conformal time)$*$c in Mpc
- H stands for (Hubble parameter)/c in Mpc^{-1}
- etc.

2) New since v3.0: all quantities that should normally scale with some power of a_0^n are renormalised by a_0^{-n}, in order to be independent of a_0, e.g.

- a in the code stands for a/a_0 in reality
- tau in the code stands for $a_0 \tau c$ in Mpc
- any prime in the code stands for $(1/a_0)d/d\tau$
- r stands for any comoving radius times a_0
- etc.

4.1.2 Function Documentation

4.1.2.1 background_at_z()

```
int background_at_z (
              struct background * pba,
              double z,
              enum vecback_format return_format,
              enum interpolation_method inter_mode,
              int * last_index,
              double * pvecback )
```

Background quantities at given redshift z.

Evaluates all background quantities at a given value of redshift by reading the pre-computed table and interpolating.

Parameters

pba	Input: pointer to background structure (containing pre-computed table)
z	Input: redshift
return_format	Input: format of output vector (short_info, normal_info, long_info)
inter_mode	Input: interpolation mode (normal or closeby)
last_index	Input/Output: index of the previous/current point in the interpolation array (input only for closeby mode, output for both)
pvecback	Output: vector (assumed to be already allocated)

Returns

the error status

Summary:

- define local variables
- check that log(a) = log(1/(1+z)) = -log(1+z) is in the pre-computed range
- deduce length of returned vector from format mode
- interpolate from pre-computed table with array_interpolate() or array_interpolate_growing_closeby() (depending on interpolation mode)

4.1.2.2 background_at_tau()

```
int background_at_tau (
              struct background * pba,
              double tau,
              enum vecback_format return_format,
              enum interpolation_method inter_mode,
              int * last_index,
              double * pvecback )
```

Background quantities at given conformal time tau.

Evaluates all background quantities at a given value of conformal time by reading the pre-computed table and interpolating.

Parameters

pba	Input: pointer to background structure (containing pre-computed table)
tau	Input: value of conformal time
return_format	Input: format of output vector (short_info, normal_info, long_info)
inter_mode	Input: interpolation mode (normal or closeby)
last_index	Input/Output: index of the previous/current point in the interpolation array (input only for closeby mode, output for both)
pvecback	Output: vector (assumed to be already allocated)

Returns

the error status

Summary:

- define local variables
- Get current redshift
- Get background at corresponding redshift

4.1.2.3 background_tau_of_z()

```
int background_tau_of_z (
             struct background * pba,
             double z,
             double * tau )
```

Conformal time at given redshift.

Returns tau(z) by interpolation from pre-computed table.

Parameters

pba	Input: pointer to background structure
z	Input: redshift
tau	Output: conformal time

Returns

the error status

Summary:

- define local variables
- check that z is in the pre-computed range
- interpolate from pre-computed table with array_interpolate()

4.1.2.4 background_z_of_tau()

```
int background_z_of_tau (
             struct background * pba,
             double tau,
             double * z )
```

Redshift at given conformal time.

Returns z(tau) by interpolation from pre-computed table.

Parameters

pba	Input: pointer to background structure
tau	Input: conformal time
z	Output: redshift

Returns

the error status

Summary:

- define local variables
- check that tau is in the pre-computed range
- interpolate from pre-computed table with array_interpolate()

4.1.2.5 background_functions()

```
int background_functions (
             struct background * pba,
             double a,
             double * pvecback_B,
             enum vecback_format return_format,
             double * pvecback )
```

Function evaluating all background quantities which can be computed analytically as a function of a and of {B} quantities (see discussion at the beginning of this file).

Parameters

pba	Input: pointer to background structure
a	Input: scale factor (in fact, with our normalisation conventions, this is (a/a_0))
pvecback_B	Input: vector containing all {B} quantities
return_format	Input: format of output vector
pvecback	Output: vector of background quantities (assumed to be already allocated)

Returns

the error status

Summary:

- define local variables
- initialize local variables
- pass value of a to output
- compute each component's density and pressure

<– This depends on a_prime_over_a, so we cannot add it now!

See e.g. Eq. A6 in 1811.00904.

- compute expansion rate H from Friedmann equation: this is the only place where the Friedmann equation is assumed. Remember that densities are all expressed in units of $[3c^2/8\pi G]$, ie $\rho_{class} = [8\pi G\rho_{physical}/3c^2]$
- compute derivative of H with respect to conformal time

The contribution of scf was not added to dp_dloga, add p_scf_prime here:

- compute critical density
- compute relativistic density to total density ratio
- compute other quantities in the exhaustive, redundant format
- store critical density
- compute Omega_m
- cosmological time
- comoving sound horizon
- growth factor
- velocity growth factor
- Varying fundamental constants

4.1.2.6 background_w_fld()

```
int background_w_fld (
              struct background * pba,
              double a,
              double * w_fld,
              double * dw_over_da_fld,
              double * integral_fld )
```

Single place where the fluid equation of state is defined. Parameters of the function are passed through the background structure. Generalisation to arbitrary functions should be simple.

Parameters

pba	Input: pointer to background structure
a	Input: current value of scale factor (in fact, with our conventions, of (a/a_0))
w_fld	Output: equation of state parameter w_fld(a)
dw_over_da_fld	Output: function dw_fld/da
integral_fld	Output: function $\int_a^{a_0} da 3(1+w_{fld})/a$

Returns

the error status

- first, define the function w(a)

- then, give the corresponding analytic derivative dw/da (used by perturbation equations; we could compute it numerically, but with a loss of precision; as long as there is a simple analytic expression of the derivative of the previous function, let's use it!
- finally, give the analytic solution of the following integral: $\int_a^{a0} da 3(1+w_{fld})/a$. This is used in only one place, in the initial conditions for the background, and with a=a_ini. If your w(a) does not lead to a simple analytic solution of this integral, no worry: instead of writing something here, the best would then be to leave it equal to zero, and then in background_initial_conditions() you should implement a numerical calculation of this integral only for a=a_ini, using for instance Romberg integration. It should be fast, simple, and accurate enough.

note: of course you can generalise these formulas to anything, defining new parameters pba->w..._fld. Just remember that so far, HyRec explicitely assumes that w(a)= w0 + wa (1-a/a0); but Recfast does not assume anything

4.1.2.7 background_varconst_of_z()

```
int background_varconst_of_z (
              struct background * pba,
              double z,
              double * alpha,
              double * me )
```

Single place where the variation of fundamental constants is defined. Parameters of the function are passed through the background structure. Generalisation to arbitrary functions should be simple.

Parameters

pba	Input: pointer to background structure
z	Input: current value of redhsift
alpha	Output: fine structure constant relative to its current value
me	Output: effective electron mass relative to its current value

Returns

the error status

4.1.2.8 background_init()

```
int background_init (
              struct precision * ppr,
              struct background * pba )
```

Initialize the background structure, and in particular the background interpolation table.

Parameters

ppr	Input: pointer to precision structure
pba	Input/Output: pointer to initialized background structure

Returns

the error status

Summary:

- write class version
- if shooting failed during input, catch the error here
- assign values to all indices in vectors of background quantities
- check that input parameters make sense and write additional information about them
- integrate the background over log(a), allocate and fill the background table
- find and store a few derived parameters at radiation-matter equality

4.1.2.9 background_free()

```
int background_free (
             struct background * pba )
```

Free all memory space allocated by background_init() and by input_read_parameters().

Parameters

pba	Input: pointer to background structure (to be freed)

Returns

the error status

4.1.2.10 background_free_noinput()

```
int background_free_noinput (
             struct background * pba )
```

Free only the memory space NOT allocated through input_read_parameters(), but through background_init()

Parameters

pba	Input: pointer to background structure (to be freed)

Returns

the error status

4.1.2.11 background_free_input()

```
int background_free_input (
             struct background * pba )
```

Free pointers inside background structure which were allocated in input_read_parameters()

Parameters

pba	Input: pointer to background structure

Returns

the error status

4.1.2.12 background_indices()

```
int background_indices (
             struct background * pba )
```

Assign value to each relevant index in vectors of background quantities.

Parameters

pba	Input: pointer to background structure

Returns

the error status

Summary:

- define local variables
- initialize all flags: which species are present?
- initialize all indices

4.1.2.13 background_ncdm_distribution()

```
int background_ncdm_distribution (
             void * pbadist,
             double q,
             double * f0 )
```

This is the routine where the distribution function f0(q) of each ncdm species is specified (it is the only place to modify if you need a partlar f0(q))

Parameters

pbadist	Input: structure containing all parameters defining f0(q)
q	Input: momentum
f0	Output: phase-space distribution

- extract from the input structure pbadist all the relevant information
- shall we interpolate in file, or shall we use analytical formula below?
- a) deal first with the case of interpolating in files
- b) deal now with case of reading analytical function

Next enter your analytic expression(s) for the p.s.d.'s. If you need different p.s.d.'s for different species, put each p.s.d inside a condition, like for instance: if (n_ncdm==2) {∗f0=...}. Remember that n_ncdm = 0 refers to the first species.

This form is only appropriate for approximate studies, since in reality the chemical potentials are associated with flavor eigenstates, not mass eigenstates. It is easy to take this into account by introducing the mixing angles. In the later part (not read by the code) we illustrate how to do this.

4.1.2.14 background_ncdm_test_function()

```
int background_ncdm_test_function (
             void * pbadist,
             double q,
             double * test )
```

This function is only used for the purpose of finding optimal quadrature weights. The logic is: if we can accurately convolve f0(q) with this function, then we can convolve it accurately with any other relevant function.

Parameters

pbadist	Input: structure containing all background parameters
q	Input: momentum
test	Output: value of the test function test(q)

Using a + bq creates problems for otherwise acceptable distributions which diverges as $1/r$ or $1/r^2$ for $r \to 0$

4.1.2.15 background_ncdm_init()

```
int background_ncdm_init (
             struct precision * ppr,
             struct background * pba )
```

This function finds optimal quadrature weights for each ncdm species

Parameters

ppr	Input: precision structure
pba	Input/Output: background structure

Automatic q-sampling for this species

- in verbose mode, inform user of number of sampled momenta for background quantities

Manual q-sampling for this species. Same sampling used for both perturbation and background sampling, since this will usually be a high precision setting anyway

- in verbose mode, inform user of number of sampled momenta for background quantities

4.1.2.16 background_ncdm_momenta()

```
int background_ncdm_momenta (
             double * qvec,
             double * wvec,
             int qsize,
             double M,
             double factor,
             double z,
             double * n,
             double * rho,
             double * p,
             double * drho_dM,
             double * pseudo_p )
```

For a given ncdm species: given the quadrature weights, the mass and the redshift, find background quantities by a quick weighted sum over. Input parameters passed as NULL pointers are not evaluated for speed-up

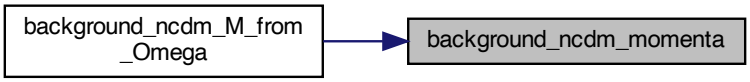
Parameters

qvec	Input: sampled momenta
wvec	Input: quadrature weights
qsize	Input: number of momenta/weights
M	Input: mass
factor	Input: normalization factor for the p.s.d.
z	Input: redshift
n	Output: number density
rho	Output: energy density
p	Output: pressure
drho_dM	Output: derivative used in next function
pseudo↩	Output: pseudo-pressure used in perturbation module for fluid approx

Summary:

- rescale normalization at given redshift
- initialize quantities
- loop over momenta
- adjust normalization

Here is the caller graph for this function:



4.1.2.17 background_ncdm_M_from_Omega()

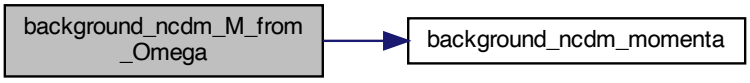
```
int background_ncdm_M_from_Omega (
             struct precision * ppr,
             struct background * pba,
             int n_ncdm )
```

When the user passed the density fraction Omega_ncdm or omega_ncdm in input but not the mass, infer the mass with Newton iteration method.

Parameters

ppr	Input: precision structure
pba	Input/Output: background structure
n_ncdm	Input: index of ncdm species

Here is the call graph for this function:



4.1.2.18 background_checks()

```
int background_checks (
              struct precision * ppr,
              struct background * pba )
```

Perform some check on the input background quantities, and send to standard output some information about them

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to initialized background structure

Returns

the error status

- define local variables
- control that cosmological parameter values make sense, otherwise inform user
- in verbose mode, send to standard output some additional information on non-obvious background parameters

4.1.2.19 background_solve()

```
int background_solve (
              struct precision * ppr,
              struct background * pba )
```

This function integrates the background over time, allocates and fills the background table

Parameters

ppr	Input: precision structure
pba	Input/Output: background structure

Summary:

- define local variables
- setup background workspace
- allocate vector of quantities to be integrated
- impose initial conditions with background_initial_conditions()
- Determine output vector
- allocate background tables

- define values of loga at which results will be stored
- choose the right evolver
- perform the integration
- recover some quantities today
- In a loop over lines, fill rest of background table for quantities that depend on numbers like "conformal_age" or "D_today" that were calculated just before
- fill tables of second derivatives (in view of spline interpolation)
- compute remaining "related parameters"
- so-called "effective neutrino number", computed at earliest time in interpolation table. This should be seen as a definition: Neff is the equivalent number of instantaneously-decoupled neutrinos accounting for the radiation density, beyond photons
- send information to standard output
- store information in the background structure

4.1.2.20 background_initial_conditions()

```
int background_initial_conditions (
             struct precision * ppr,
             struct background * pba,
             double * pvecback,
             double * pvecback_integration,
             double * loga_ini )
```

Assign initial values to background integrated variables.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pvecback	Input: vector of background quantities used as workspace
pvecback_integration	Output: vector of background quantities to be integrated, returned with proper initial values
loga_ini	Output: value of loga (in fact with our conventions log(a/a_0)) at initial time

Returns

the error status

Summary:

- define local variables
- fix initial value of a

If we have ncdm species, perhaps we need to start earlier than the standard value for the species to be relativistic. This could happen for some WDM models.

- We must add the relativistic contribution from NCDM species
- f is the critical density fraction of DR. The exact solution is:

```
f = -Omega_rad+pow(pow(Omega_rad,3./2.)+0.5*pow(a,6)*pvecback_integration[pba->index↩
_bi_rho_dcdm]*pba->Gamma_dcdm/pow(pba->H0,3),2./3.);
```

but it is not numerically stable for very small f which is always the case. Instead we use the Taylor expansion of this equation, which is equivalent to ignoring f(a) in the Hubble rate.

There is also a space reserved for a future case where dr is not sourced by dcdm

- Fix initial value of ϕ, ϕ' set directly in the radiation attractor => fixes the units in terms of rho_ur

TODO:

- There seems to be some small oscillation when it starts.
- Check equations and signs. Sign of phi_prime?
- is rho_ur all there is early on?
- --> If there is no attractor solution for scf_lambda, assign some value. Otherwise would give a nan.
- --> If no attractor initial conditions are assigned, gets the provided ones.
- compute initial proper time, assuming radiation-dominated universe since Big Bang and therefore $t = 1/(2H)$ (good approximation for most purposes)
- compute initial conformal time, assuming radiation-dominated universe since Big Bang and therefore $\tau = 1/(aH)$ (good approximation for most purposes)
- compute initial sound horizon, assuming $c_s = 1/\sqrt{3}$ initially
- set initial value of D and D' in RD. D and D' need only be set up to an overall constant, since they will later be re-normalized. From Ma&Bertschinger, one can derive D $\sim$ (ktau)^2 at early times, from which one finds D'/D = 2 aH (assuming aH=1/tau during RD)
- return the value finally chosen for the initial log(a)

4.1.2.21 background_find_equality()

```
int background_find_equality (
             struct precision * ppr,
             struct background * pba )
```

Find the time of radiation/matter equality and store characteristic quantitites at that time in the background structure..

Parameters

ppr	Input: pointer to precision structure
pba	Input/Output: pointer to background structure

Returns

the error status

4.1.2.22 background_output_titles()

```
int background_output_titles (
            struct background * pba,
            char titles[_MAXTITLESTRINGLENGTH_] )
```

Subroutine for formatting background output

Parameters

pba	Input: pointer to background structure
titles	Ouput: name of columns when printing the background table

Returns

the error status

- Length of the column title should be less than *OUTPUTPRECISION*+6 to be indented correctly, but it can be as long as .

4.1.2.23 background_output_data()

```
int background_output_data (
            struct background * pba,
            int number_of_titles,
            double * data )
```

Subroutine for writing the background output

Parameters

pba	Input: pointer to background structure
number_of_titles	Input: number of background quantities to print at each time step
data	Ouput: 1d array storing all the background table

Returns

the error status

Stores quantities

4.1.2.24 background_derivs()

```
int background_derivs (
              double loga,
              double * y,
              double * dy,
              void * parameters_and_workspace,
              ErrorMsg error_message )
```

Subroutine evaluating the derivative with respect to loga of quantities which are integrated (tau, t, etc).

This is one of the few functions in the code which is passed to the generic_integrator() routine. Since generic_integrator() should work with functions passed from various modules, the format of the arguments is a bit special:

- fixed input parameters and workspaces are passed through a generic pointer. Here, this is just a pointer to the background structure and to a background vector, but generic_integrator() doesn't know its fine structure.
- the error management is a bit special: errors are not written as usual to pba->error_message, but to a generic error_message passed in the list of arguments.

Parameters

loga	Input: current value of log(a)
y	Input: vector of variable
dy	Output: its derivative (already allocated)
parameters_and_workspace	Input: pointer to fixed parameters (e.g. indices)
error_message	Output: error message

Summary:

- define local variables
- scale factor a (in fact, given our normalisation conventions, this stands for a/a_0)
- calculate functions of a with background_functions()
- Short hand notation for Hubble
- calculate derivative of cosmological time $dt/dloga = 1/H$
- calculate derivative of conformal time $d\tau/dloga = 1/aH$
- calculate detivative of sound horizon $drs/dloga = drs/dtau * dtau/dloga = c_s/aH$
- solve second order growth equation $[D"(\tau) = -aHD'(\tau) + 3/2a^2\rho_M D(\tau)$ written as $dD/dloga = D'/(aH)$ and $dD'/dloga = -D' + (3/2)(a/H)\rho_M D$
- compute dcdm density $d\rho/dloga = -3\rho - \Gamma/H\rho$
- Compute dr density $d\rho/dloga = -4\rho - \Gamma/H\rho$
- Compute fld density $d\rho/dloga = -3(1 + w_{fld}(a))\rho$
- Scalar field equation: $\phi" + 2aH\phi' + a^2dV = 0$ (note H is wrt cosmological time) written as $d\phi/dlna = phi'/(aH)$ and $d\phi'/dlna = -2 * phi' - (a/H)dV$

4.1.2.25 background_sources()

```
int background_sources (
             double loga,
             double * y,
             double * dy,
             int index_loga,
             void * parameters_and_workspace,
             ErrorMsg error_message )
```

At some step during the integraton of the background equations, this function extracts the qantities that we want to keep memory of, and stores them in a row of the background table (as well as extra tables: z_table, tau_table).

This is one of the few functions in the code which is passed to the generic_integrator() routine. Since generic_↩ integrator() should work with functions passed from various modules, the format of the arguments is a bit special:

- fixed parameters and workspaces are passed through a generic pointer. generic_integrator() doesn't know the content of this pointer.
- the error management is a bit special: errors are not written as usual to pba->error_message, but to a generic error_message passed in the list of arguments.

Parameters

loga	Input: current value of log(a)
y	Input: current vector of integrated quantities (with index_bi)
dy	Input: current derivative of y w.r.t log(a)
index_loga	Input: index of the log(a) value within the background_table
parameters_and_workspace	Input/output: fixed parameters (e.g. indices), workspace, background structure where the output is written...
error_message	Output: error message

- localize the row inside background_table where the current values must be stored
- scale factor a (in fact, given our normalisation conventions, this stands for a/a_0)
- corresponding redhsift 1/a-1
- corresponding conformal time

-> compute all other quantities depending only on a + {B} variables and get them stored in one row of background↩ _table The value of {B} variables in pData are also copied to pvecback.

4.1.2.26 background_timescale()

```
int background_timescale (
             double loga,
             void * parameters_and_workspace,
             double * timescale,
             ErrorMsg error_message )
```

Evalute the typical timescale for the integration of he background over loga=log(a/a_0). This is only required for rkck, but not for the ndf15 evolver.

The evolver will take steps equal to this value times ppr->background_integration_stepsize. Since our variable of integration is loga, and the time steps are (delta a)/a, the reference timescale is precisely one, i.e., the code will take some steps such that (delta a)/a = ppr->background_integration_stepsize.

The argument list is predetermined by the format of generic_evolver; however in this particular case, they are never used.

This is one of the few functions in the code which is passed to the generic_integrator() routine. Since generic_integrator() should work with functions passed from various modules, the format of the arguments is a bit special:

- fixed parameters and workspaces are passed through a generic pointer (void *). generic_integrator() doesn't know the content of this pointer.
- the error management is a bit special: errors are not written as usual to pba->error_message, but to a generic error_message passed in the list of arguments.

Parameters

loga	Input: current value of log(a/a_0)
parameters_and_workspace	Input: fixed parameters (e.g. indices), workspace, approximation used, etc.
timescale	Output: perturbation variation timescale
error_message	Output: error message

4.1.2.27 background_output_budget()

```
int background_output_budget (
              struct background * pba )
```

Function outputting the fractions Omega of the total critical density today, and also the reduced fractions omega=Omega*h*h

It also prints the total budgets of non-relativistic, relativistic, and other contents, and of the total

Parameters

pba	Input: Pointer to background structure

Returns

the error status

4.1.2.28 V_e_scf()

```
double V_e_scf (
```

```
            struct background * pba,
            double phi )
```

Scalar field potential and its derivatives with respect to the field _scf For Albrecht & Skordis model: 9908085

- $V = V_{p_{scf}} * V_{e_{scf}}$
- $V_e = \exp(-\lambda\phi)$ (exponential)
- $V_p = (\phi - B)^\alpha + A$ (polynomial bump)

TODO:

- Add some functionality to include different models/potentials (tuning would be difficult, though)
- Generalize to Kessence/Horndeski/PPF and/or couplings
- A default module to numerically compute the derivatives when no analytic functions are given should be added.
- Numerical derivatives may further serve as a consistency check.

  The units of phi, tau in the derivatives and the potential V are the following:
  - phi is given in units of the reduced Planck mass $m_{pl} = (8\pi G)^{(-1/2)}$
  - tau in the derivative is given in units of Mpc.
  - the potential $V(\phi)$ is given in units of m_{pl}^2/Mpc^2. With this convention, we have $\rho^{class} = (8\pi G)/3\rho^{physical} = 1/(3m_{pl}^2)\rho^{physical} = 1/3 * [1/(2a^2)(\phi')^2 + V(\phi)]$ and ρ^{class} has the proper dimension Mpc^-2.

Here is the caller graph for this function:

4.1.2.29 V_p_scf()

```
double V_p_scf (
            struct background * pba,
            double phi )
```

parameters and functions for the polynomial coefficient $V_p = (\phi - B)^\alpha + A$(polynomial bump)

double scf_alpha = 2;

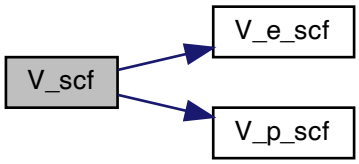
double scf_B = 34.8;

double scf_A = 0.01; (values for their Figure 2) Here is the caller graph for this function:

4.1.2.30 V_scf()

```
double V_scf (
            struct background * pba,
            double phi )
```

Fianlly we can obtain the overall potential $V = V_p * V_e$ Here is the call graph for this function:

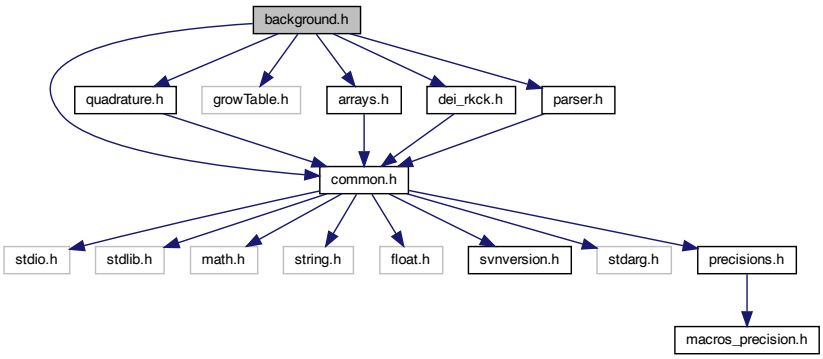


4.2 background.h File Reference

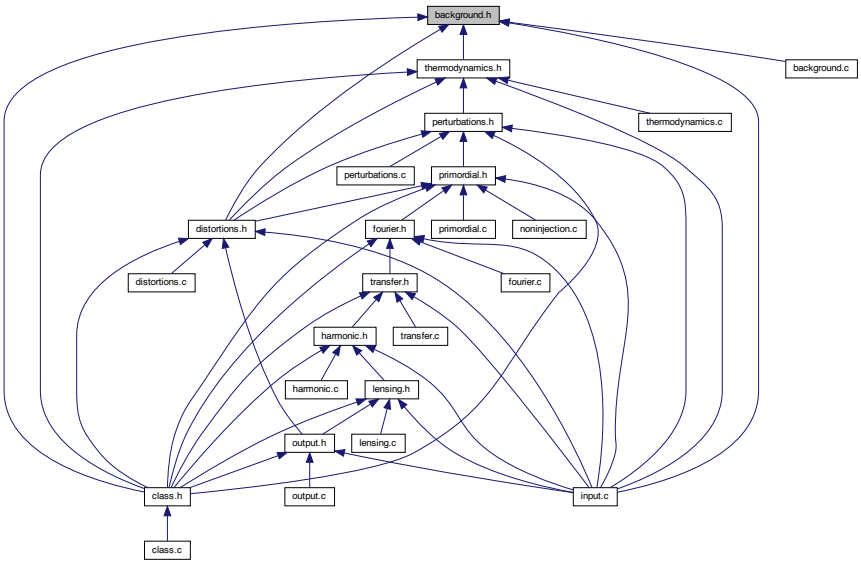
```
#include "common.h"
#include "quadrature.h"
#include "growTable.h"
#include "arrays.h"
#include "dei_rkck.h"
```

```
#include "parser.h"
```

Include dependency graph for background.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct background
- struct background_parameters_and_workspace
- struct background_parameters_for_distributions

Enumerations

- enum spatial_curvature
- enum equation_of_state
- enum varconst_dependence
- enum vecback_format
- enum interpolation_method

4.2.1 Detailed Description

Documented includes for background module

4.2.2 Data Structure Documentation

4.2.2.1 struct background

background structure containing all the background information that other modules need to know.

Once initialized by the backgound_init(), contains all necessary information on the background evolution (except thermodynamics), and in particular, a table of all background quantities as a function of time and scale factor, used for interpolation in other modules.

Data Fields

double	H0	H_0: Hubble parameter (in fact, [H_0/c]) in Mpc^{-1}
double	h	reduced Hubble parameter
double	Omega0_g	$\Omega_{0\gamma}$: photons
double	T_cmb	T_{cmb}: current CMB temperature in Kelvins
double	Omega0_b	Ω_{0b}: baryons
double	Omega0_ur	$\Omega_{0\nu r}$: ultra-relativistic neutrinos
double	Omega0_cdm	Ω_{0cdm}: cold dark matter
double	Omega0_idr	Ω_{0idr}: interacting dark radiation
double	T_idr	T_{idr}: current temperature of interacting dark radiation in Kelvins
double	Omega0_idm_dr	Ω_{0idm_dr}: dark matter interacting with dark radiation
double	Omega0_dcdmdr	$\Omega_{0dcdm} + \Omega_{0dr}$: decaying cold dark matter (dcdm) decaying to dark radiation (dr)
double	Omega_ini_dcdm	$\Omega_{ini,dcdm}$: rescaled initial value for dcdm density (see 1407.2418 for definitions)
double	Gamma_dcdm	Γ_{dcdm}: decay constant for decaying cold dark matter
double	tau_dcdm	
int	N_ncdm	Number of distinguishable ncdm species
char *	ncdm_psd_files	list of filenames for tabulated p-s-d
int *	got_files	list of flags for each species, set to true if p-s-d is passed through file
double *	ncdm_psd_parameters	list of parameters for specifying/modifying ncdm p.s.d.'s, to be customized for given model (could be e.g. mixing angles)
double *	M_ncdm	vector of masses of non-cold relic: dimensionless ratios m_ncdm/T_ncdm
double *	m_ncdm_in_eV	list of ncdm masses in eV (inferred from M_ncdm and other parameters above)
double *	Omega0_ncdm	
double	Omega0_ncdm_tot	Omega0_ncdm for each species and for the total Omega0_ncdm
double *	T_ncdm	

Data Fields

double	T_ncdm_default	list of 1st parameters in p-s-d of non-cold relics: relative temperature T_ncdm1/T_gamma; and its default value
double *	ksi_ncdm	
double	ksi_ncdm_default	list of 2nd parameters in p-s-d of non-cold relics: relative chemical potential ksi_ncdm1/T_ncdm1; and its default value
double *	deg_ncdm	
double	deg_ncdm_default	vector of degeneracy parameters in factor of p-s-d: 1 for one family of neutrinos (= one neutrino plus its anti-neutrino, total g$*$=1+1=2, so deg = 0.5 g$*$); and its default value
int *	ncdm_input_q_size	Vector of numbers of q bins
double *	ncdm_qmax	Vector of maximum value of q
double	Omega0_k	Ω_{0_k}: curvature contribution
double	Omega0_lambda	Ω_{0_Λ}: cosmological constant
double	Omega0_fld	Ω_{0de}: fluid
double	Omega0_scf	Ω_{0scf}: scalar field
short	use_ppf	flag switching on PPF perturbation equations instead of true fluid equations for perturbations. It could have been defined inside perturbation structure, but we leave it here in such way to have all fld parameters grouped.
double	c_gamma_over_c_fld	ppf parameter defined in eq. (16) of 0808.3125 [astro-ph]
enum equation_of_state	fluid_equation_of_state	parametrisation scheme for fluid equation of state
double	w0_fld	$w0_{DE}$: current fluid equation of state parameter
double	wa_fld	wa_{DE}: fluid equation of state parameter derivative
double	cs2_fld	$c_{s\ DE}^2$: sound speed of the fluid in the frame comoving with the fluid (so, this is not [delta p/delta rho] in the synchronous or newtonian gauge!)
double	Omega_EDE	wa_{DE}: Early Dark Energy density parameter
double *	scf_parameters	list of parameters describing the scalar field potential
short	attractor_ic_scf	whether the scalar field has attractor initial conditions
int	scf_tuning_index	index in scf_parameters used for tuning
double	phi_ini_scf	$\phi(t_0)$: scalar field initial value
double	phi_prime_ini_scf	$d\phi(t_0)/d\tau$: scalar field initial derivative wrt conformal time
int	scf_parameters_size	size of scf_parameters
double	varconst_alpha	finestructure constant for varying fundamental constants
double	varconst_me	electron mass for varying fundamental constants
enum varconst_dependence	varconst_dep	dependence of the varying fundamental constants as a function of time

Data Fields

double	varconst_transition_redshift	redshift of transition between varied fundamental constants and normal fundamental constants in the 'varconst_instant' case
double	age	age in Gyears
double	conformal_age	conformal age in Mpc
double	K	K: Curvature parameter $K = -\Omega 0_k * a^2_{today} * H^2_0$;
int	sgnK	K/|K|: -1, 0 or 1
double	Neff	so-called "effective neutrino number", computed at earliest time in interpolation table
double	Omega0_dcdm	Ω_{0dcdm}: decaying cold dark matter
double	Omega0_dr	Ω_{0dr}: decay radiation
double	Omega0_m	total non-relativistic matter today
double	Omega0_r	total ultra-relativistic radiation today
double	Omega0_de	total dark energy density today, currently defined as 1 - Omega0_m - Omega0_r - Omega0_k
double	Omega0_nfsm	total non-free-streaming matter, that is, cdm, baryons and wdm
double	a_eq	scale factor at radiation/matter equality
double	H_eq	Hubble rate at radiation/matter equality [Mpc^-1]
double	z_eq	redshift at radiation/matter equality
double	tau_eq	conformal time at radiation/matter equality [Mpc]
int	index_bg_a	scale factor (in fact (a/a_0), see normalisation conventions explained at beginning of background.c)
int	index_bg_H	Hubble parameter in Mpc^{-1}
int	index_bg_H_prime	its derivative w.r.t. conformal time
int	index_bg_rho_g	photon density
int	index_bg_rho_b	baryon density
int	index_bg_rho_cdm	cdm density
int	index_bg_rho_idm_dr	density of dark matter interacting with dark radiation
int	index_bg_rho_lambda	cosmological constant density
int	index_bg_rho_fld	fluid density
int	index_bg_w_fld	fluid equation of state
int	index_bg_rho_idr	density of interacting dark radiation
int	index_bg_rho_ur	relativistic neutrinos/relics density
int	index_bg_rho_dcdm	dcdm density
int	index_bg_rho_dr	dr density
int	index_bg_phi_scf	scalar field value
int	index_bg_phi_prime_scf	scalar field derivative wrt conformal time
int	index_bg_V_scf	scalar field potential V
int	index_bg_dV_scf	scalar field potential derivative V'
int	index_bg_ddV_scf	scalar field potential second derivative V''

Data Fields

int	index_bg_rho_scf	scalar field energy density
int	index_bg_p_scf	scalar field pressure
int	index_bg_p_prime_scf	scalar field pressure
int	index_bg_rho_ncdm1	density of first ncdm species (others contiguous)
int	index_bg_p_ncdm1	pressure of first ncdm species (others contiguous)
int	index_bg_pseudo_p_ncdm1	another statistical momentum useful in ncdma approximation
int	index_bg_rho_tot	Total density
int	index_bg_p_tot	Total pressure
int	index_bg_p_tot_prime	Conf. time derivative of total pressure
int	index_bg_Omega_r	relativistic density fraction ($\Omega_\gamma + \Omega_{\nu r}$)
int	index_bg_rho_crit	critical density
int	index_bg_Omega_m	non-relativistic density fraction ($\Omega_b + \Omega_c dm + \Omega_{\nu nr}$)
int	index_bg_conf_distance	conformal distance (from us) in Mpc
int	index_bg_ang_distance	angular diameter distance in Mpc
int	index_bg_lum_distance	luminosity distance in Mpc
int	index_bg_time	proper (cosmological) time in Mpc
int	index_bg_rs	comoving sound horizon in Mpc
int	index_bg_D	scale independent growth factor D(a) for CDM perturbations
int	index_bg_f	corresponding velocity growth factor [dlnD]/[dln a]
int	index_bg_varc_alpha	value of fine structure constant in varying fundamental constants
int	index_bg_varc_me	value of effective electron mass in varying fundamental constants
int	bg_size_short	size of background vector in the "short format"
int	bg_size_normal	size of background vector in the "normal format"
int	bg_size	size of background vector in the "long format"
int	bt_size	number of lines (i.e. time-steps) in the four following array
double *	loga_table	vector loga_table[index_loga] with values of log(a) (in fact $log(a/a0)$, logarithm of relative scale factor compared to today)
double *	tau_table	vector tau_table[index_loga] with values of conformal time τ (in fact $a_0 ctau$, see normalisation conventions explained at beginning of background.c)
double *	z_table	vector z_table[index_loga] with values of z (redshift)
double *	background_table	table background_table[index_tau*pba->bg↩ _size+pba->index_bg] with all other quantities (array of size bg_size*bt_size)
double *	d2tau_dz2_table	vector d2tau_dz2_table[index_loga] with values of $d^2\tau/dz^2$ (conformal time)

Data Fields

double *	d2z_dtau2_table	vector d2z_dtau2_table[index_loga] with values of $d^2z/d\tau^2$ (conformal time)
double *	d2background_dloga2_table	table d2background_dtau2_table[index_↩ loga*pba->bg_size+pba->index_bg] with values of $d^2b_i/d\log(a)^2$
int	index_bi_rho_dcdm	{B} dcdm density
int	index_bi_rho_dr	{B} dr density
int	index_bi_rho_fld	{B} fluid density
int	index_bi_phi_scf	{B} scalar field value
int	index_bi_phi_prime_scf	{B} scalar field derivative wrt conformal time
int	index_bi_time	{C} proper (cosmological) time in Mpc
int	index_bi_rs	{C} sound horizon
int	index_bi_tau	{C} conformal time in Mpc
int	index_bi_D	{C} scale independent growth factor D(a) for CDM perturbations.
int	index_bi_D_prime	{C} D satisfies $[D"(\tau) = -aHD'(\tau) + 3/2a^2\rho_M D(\tau)$
int	bi_B_size	Number of {B} parameters
int	bi_size	Number of {B}+{C} parameters
short	has_cdm	presence of cold dark matter?
short	has_idm_dr	presence of dark matter interacting with dark radiation?
short	has_dcdm	presence of decaying cold dark matter?
short	has_dr	presence of relativistic decay radiation?
short	has_scf	presence of a scalar field?
short	has_ncdm	presence of non-cold dark matter?
short	has_lambda	presence of cosmological constant?
short	has_fld	presence of fluid with constant w and cs2?
short	has_ur	presence of ultra-relativistic neutrinos/relics?
short	has_idr	presence of interacting dark radiation?
short	has_curvature	presence of global spatial curvature?
short	has_varconst	presence of varying fundamental constants?
int *	ncdm_quadrature_strategy	Vector of integers according to quadrature strategy.
double **	q_ncdm_bg	Pointers to vectors of background sampling in q
double **	w_ncdm_bg	Pointers to vectors of corresponding quadrature weights w
double **	q_ncdm	Pointers to vectors of perturbation sampling in q
double **	w_ncdm	Pointers to vectors of corresponding quadrature weights w
double **	dlnf0_dlnq_ncdm	Pointers to vectors of logarithmic derivatives of p-s-d
int *	q_size_ncdm_bg	Size of the q_ncdm_bg arrays
int *	q_size_ncdm	Size of the q_ncdm arrays
double *	factor_ncdm	List of normalization factors for calculating energy density etc.
short	shooting_failed	flag is set to true if shooting failed.

Data Fields

ErrorMsg	shooting_error	Error message from shooting failed.
short	background_verbose	flag regulating the amount of information sent to standard output (none if set to zero)
ErrorMsg	error_message	zone for writing error messages

4.2.2.2 struct background_parameters_and_workspace

temporary parameters and workspace passed to the background_derivs function

4.2.2.3 struct background_parameters_for_distributions

temporary parameters and workspace passed to phase space distribution function

4.2.3 Enumeration Type Documentation

4.2.3.1 spatial_curvature

enum spatial_curvature

list of possible types of spatial curvature

4.2.3.2 equation_of_state

enum equation_of_state

list of possible parametrisations of the DE equation of state

4.2.3.3 varconst_dependence

enum varconst_dependence

list of possible parametrizations of the varying fundamental constants

4.2.3.4 vecback_format

enum vecback_format

list of formats for the vector of background quantities

4.2.3.5 interpolation_method

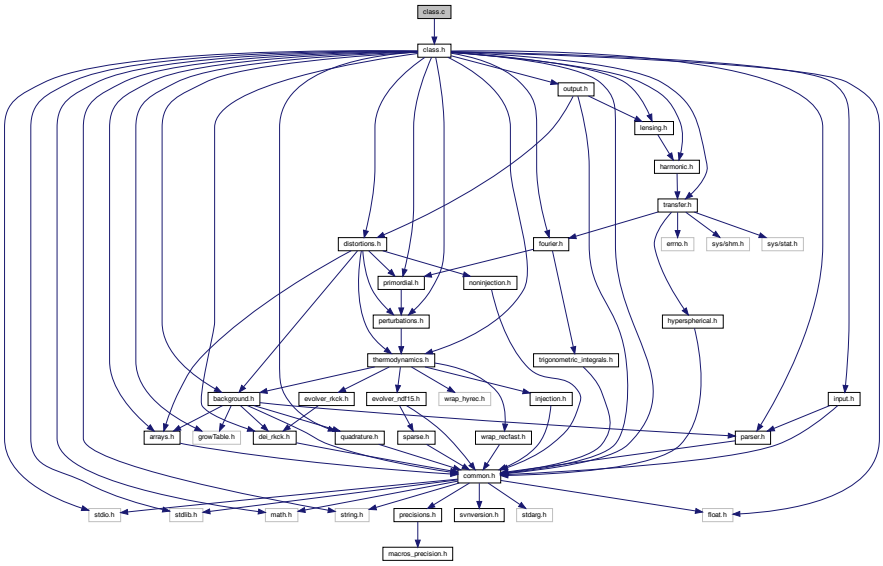
```
enum interpolation_method
```

list of interpolation methods: search location in table either by bisection (inter_normal), or step by step starting from given index (inter_closeby)

4.3 class.c File Reference

```
#include "class.h"
```

Include dependency graph for class.c:



4.3.1 Detailed Description

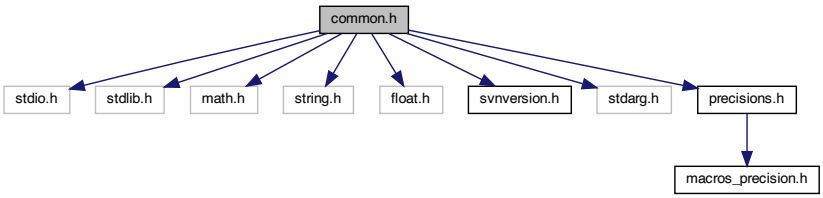
Julien Lesgourgues, 17.04.2011

4.4 common.h File Reference

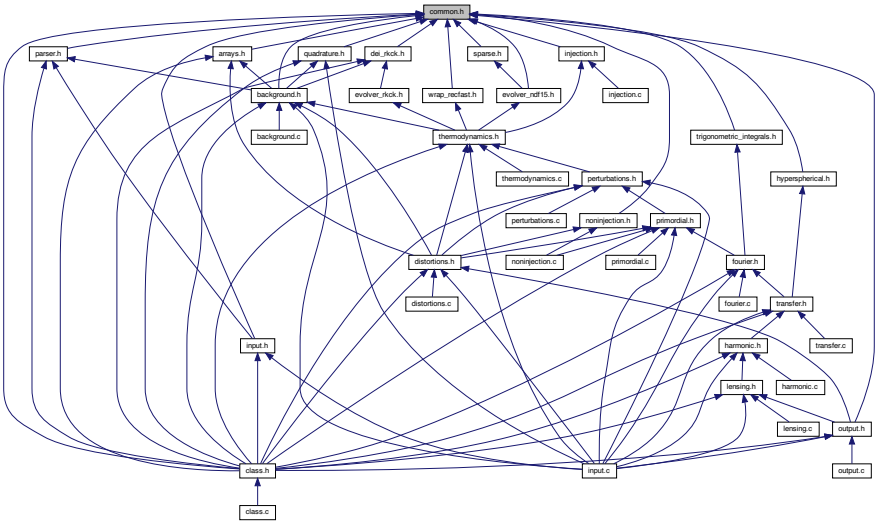
```
#include "stdio.h"
#include "stdlib.h"
#include "math.h"
#include "string.h"
#include "float.h"
#include "svnversion.h"
#include <stdarg.h>
```

```
#include "precisions.h"
```

Include dependency graph for common.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct precision

Enumerations

- enum evolver_type
- enum pk_def { delta_m_squared , delta_tot_squared , delta_bc_squared , delta_tot_from_poisson_squared }
- enum file_format

4.4.1 Detailed Description

Generic libraries, parameters and functions used in the whole code.

4.4.2 Data Structure Documentation

4.4.2.1 struct precision

All precision parameters.

Includes integrations steps, flags telling how the computation is to be performed, etc.

Data Fields

double	smallest_allowed_variation	machine-dependent, assigned automatically by the code
ErrorMsg	error_message	zone for writing error messages

4.4.3 Enumeration Type Documentation

4.4.3.1 evolver_type

`enum evolver_type`

parameters related to the precision of the code and to the method of calculation list of evolver types for integrating perturbations over time

4.4.3.2 pk_def

`enum pk_def`

List of ways in which matter power spectrum P(k) can be defined. The standard definition is the first one (delta_↩ m_squared) but alternative definitions can be useful in some projects.

Enumerator

delta_m_squared	normal definition (delta_m includes all non-relativistic species at late times)
delta_tot_squared	delta_tot includes all species contributions to (delta rho), and only non-relativistic contributions to rho
delta_bc_squared	delta_bc includes contribution of baryons and cdm only to (delta rho) and to rho
delta_tot_from_poisson_squared	use delta_tot inferred from gravitational potential through Poisson equation

4.4.3.3 file_format

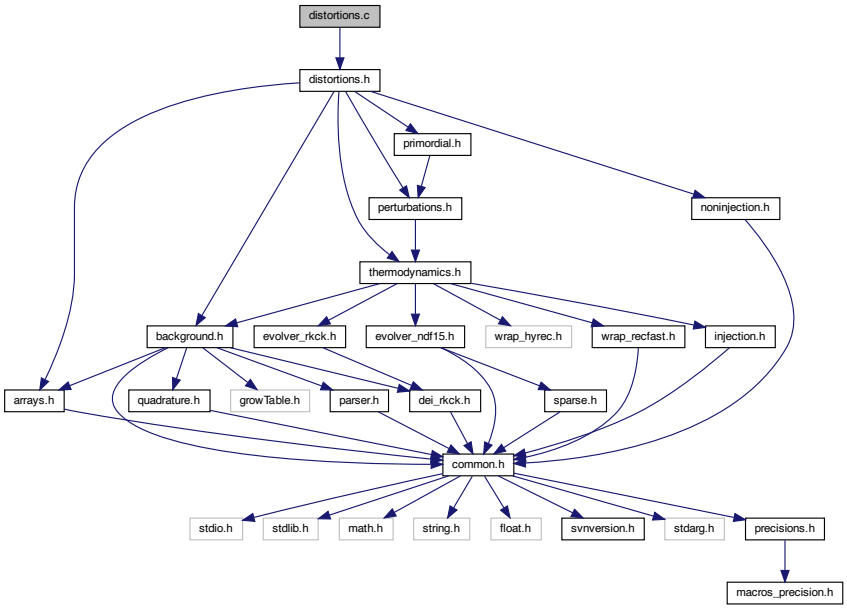
`enum file_format`

Different ways to present output files

4.5 distortions.c File Reference

```
#include "distortions.h"
```

Include dependency graph for distortions.c:



Functions

- int distortions_init (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, struct primordial *ppm, struct distortions *psd)
- int distortions_free (struct distortions *psd)
- int distortions_constants (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct distortions *psd)
- int distortions_set_detector (struct precision *ppr, struct distortions *psd)
- int distortions_generate_detector (struct precision *ppr, struct distortions *psd)
- int distortions_read_detector_noisefile (struct precision *ppr, struct distortions *psd)
- int distortions_indices (struct distortions *psd)
- int distortions_get_xz_lists (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct distortions *psd)
- int distortions_compute_branching_ratios (struct precision *ppr, struct distortions *psd)
- int distortions_compute_heating_rate (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, struct primordial *ppm, struct distortions *psd)
- int distortions_compute_spectral_shapes (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct distortions *psd)
- int distortions_add_effects_reio (struct background *pba, struct thermodynamics *pth, struct distortions *psd, double T_e, double Dtau, double beta, double beta_z, double x, double *y_reio, double *DI_reio)
- int distortions_read_br_data (struct precision *ppr, struct distortions *psd)
- int distortions_spline_br_data (struct distortions *psd)
- int distortions_interpolate_br_data (struct distortions *psd, double z, double *f_g, double *f_y, double *f_mu, double *f_E, int *last_index)
- int distortions_free_br_data (struct distortions *psd)
- int distortions_read_sd_data (struct precision *ppr, struct distortions *psd)
- int distortions_spline_sd_data (struct distortions *psd)
- int distortions_interpolate_sd_data (struct distortions *psd, double nu, double *G_T, double *Y_SZ, double *M_mu, double *S, int *index)
- int distortions_free_sd_data (struct distortions *psd)

- int distortions_output_heat_titles (struct distortions *psd, char titles[_MAXTITLESTRINGLENGTH_])
- int distortions_output_heat_data (struct distortions *psd, int number_of_titles, double *data)
- int distortions_output_sd_titles (struct distortions *psd, char titles[_MAXTITLESTRINGLENGTH_])
- int distortions_output_sd_data (struct distortions *psd, int number_of_titles, double *data)

4.5.1 Detailed Description

Documented module on spectral distortions Matteo Lucca, 31.10.2018 Nils Schoeneberg, 18.02.2019

When using this module please consider citing: Lucca et al. 2019 (JCAP02(2020)026, arXiv:1910.04619) as well as related pioneering works such as: Chluba & Sunyaev 2012 (MNRAS419(2012)1294-1314, arXiv:1109.6552) Chluba 2013 (MNRAS434(2013)352, arXiv:1304.6120) Clube & Jeong 2014 (MNRAS438(2014)2065–2082, arXiv:1306.5751)

4.5.2 Function Documentation

4.5.2.1 distortions_init()

```
int distortions_init (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             struct primordial * ppm,
             struct distortions * psd )
```

Initialize the distortions structure.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to the thermodynamics structure
ppt	Input: pointer to the perturbations structure
ppm	Input: pointer to the primordial structure
psd	Input/Output: pointer to initialized distortions structure

Returns

the error status

Set physical constants

Set/Check the distortions detector

Assign values to all indices in the distortions structure

Define z and x arrays

Define branching ratios

Define heating function

Define final spectral distortions

4.5.2.2 distortions_free()

```
int distortions_free (
             struct distortions * psd )
```

Free all memory space allocated by distortions_init()

Parameters

psd	Input: pointer to distortions structure (to be freed)

Returns

the error status

Define local variables

Delete lists

Delete noise file

Delete branching ratios

Delete heating functions

Delete distortion shapes

Delete distortion amplitudes

Delete total distortion

4.5.2.3 distortions_constants()

```
int distortions_constants (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct distortions * psd )
```

Calculate physical constant.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
psd	Input: pointer to the distortions structure

Returns

the error status

Define unit conventions

Define transition redshifts z_muy and z_th

4.5.2.4 distortions_set_detector()

```
int distortions_set_detector (
             struct precision * ppr,
             struct distortions * psd )
```

Check wether the detector name and the detector properties are a valid combination.

There are four options for the user

defined_name = true, defined_detector = true Meaning: The user requests a specific detector with specific settings --> Check that the detector exists and has the same settings

defined_name = true, defined_detector = false Meaning: The user requests a specific detector --> Check that the detector exists and use the given settings

defined_name = false, defined_detector = true Meaning: The user requests specific settings, but does not name their detector --> Check that the settings exists, or create them

defined_name = false, defined_detector = false Meaning: The user just wants the default detector and settings --> Just use the default settings and skip this function

Parameters

ppr	Input: pointer to precision structure
psd	Input/Output: pointer to initialized distortions structure

Returns

the error status

Local variables

Open file

4.5.2.5 distortions_generate_detector()

```
int distortions_generate_detector (
             struct precision * ppr,
             struct distortions * psd )
```

Evaluate branching ratios, spectral shapes, E and S vectors for a given detector as described in external/distortions/↩ README using generate_PCA_files.py.

Parameters

ppr	Input: pointer to precision structure
psd	Input: pointer to the distortions structure

Returns

the error status

Define local variables

4.5.2.6 distortions_read_detector_noisefile()

```
int distortions_read_detector_noisefile (
            struct precision * ppr,
            struct distortions * psd )
```

Reads the external detector noise file containing the array of frequencies and the detector accuracies Assumed to be in units of [GHz] and [10^-26 W/m^2/Hz/sr] respectively

Parameters

ppr	Input: pointer to the precisions structure
psd	Input: pointer to the distortions structure

Returns

the error status

Define local variables

Open file

Read header

Read number of lines, infer size of arrays and allocate them

Read parameters

4.5.2.7 distortions_indices()

```
int distortions_indices (
            struct distortions * psd )
```

Assign value to each relevant index in vectors of distortions quantities.

Parameters

psd	Input: pointer to distortions structure

Returns

the error status

Define local variables

Define indeces for tables - br_table defined in distortions_compute_branching_ratios,

- sd_parameter_table and
- sd_table defined in distortions_compute_spectral_shapes

4.5.2.8 distortions_get_xz_lists()

```
int distortions_get_xz_lists (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct distortions * psd )
```

Calculate redshift and frequency vectors and weights for redshift integral.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to the thermodynamics structure
psd	Input/Output: pointer to initialized distortions structure

Returns

the error status

Define local variables

Define and allocate z array

Define and allocate integrating weights for z array

Define and allocate x array

Define and allocate integrating weights for x array

4.5.2.9 distortions_compute_branching_ratios()

```
int distortions_compute_branching_ratios (
             struct precision * ppr,
             struct distortions * psd )
```

Calculate branching ratios.

Computing the full evolution of the thermal history of the universe is rather time consuming and mathematically challenging. It is therefore not implemented here. However, there are (at least) 5 levels of possible approximatin to evaluate the SD branching ratios (see also Chluba 2016 for useful discussion) 1) Use a sharp transition at z↩ _mu-y and no distortions before z_th ('branching approx'=sharp_sharp) 2) Use a sharp transition at z_mu-y and a soft transition at z_th ('branching approx'=sharp_soft) 3) Use a soft transition at a_mu-y and z_th as described in Chluba 2013 ('branching approx'=soft_soft) In this case, the user must be aware that energy conservation is violated and no residuals are taken into consideration. 4) Use a soft transition at a_mu-y and z_th imposing conservation of energy ('branching approx'=soft_soft_cons) 5) Use a PCA method as described in Chluba & Jeong 2014 ('branching approx'=exact) In this case, the definition of the BRs is detector dependent and the user has therefore to specify the detector type and corresponding characteristics.

All quantities are stored in the table br_table.

Parameters

ppr	Input: pointer to the precision structure
psd	Input: pointer to the distortions structure

Returns

the error status

Define local variables

Allocate space for branching ratios in br_table

Calulate branching ratios

4.5.2.10 distortions_compute_heating_rate()

```
int distortions_compute_heating_rate (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             struct primordial * ppm,
             struct distortions * psd )
```

Import heating rates from heating structure.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to the thermodynamics structure
ppt	Input: pointer to the perturbations structure
ppm	Input: pointer to the primordial structure
psd	Input: pointer to the distortions structure

Returns

the error status

Define local variables

Update heating table with second order contributions

Allocate space for background vector

Allocate space for total heating function

Import quantities from background structure

Import heat from non-injection structure

Add heat from injection structure

Calculate total heating rate

Update heating table with second order contributions

4.5.2.11 distortions_compute_spectral_shapes()

```
int distortions_compute_spectral_shapes (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct distortions * psd )
```

Calculate spectral amplitudes and corresponding distortions.

The calculation has been done according to Chluba & Jeong 2014 (arxiv:1306.5751). All quantities are stored in the tables sd_parameter_table and sd_table.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
psd	Input: pointer to the distortions structure

Returns

the error status

Define local variables

Allocate space for spectral distortion amplitude in table sd_parameter_table

Compute distortion amplitudes corresponding to each branching ratio (g, y and mu)

Allocate space for distortions shapes in distortions_table

Calculate spectral shapes

Compute distortion amplitude for residual parameter epsilon

Allocate space for final spectral distortion

Calculate spectral distortions according to Chluba & Jeong 2014 (arxiv:1306.5751, Eq. (11))

Include additional sources of distortions

Compute total heating

Print found parameters

4.5.2.12 distortions_add_effects_reio()

```
int distortions_add_effects_reio (
              struct background * pba,
              struct thermodynamics * pth,
              struct distortions * psd,
              double T_e,
              double Dtau,
              double beta,
              double beta_z,
              double x,
              double * y_reio,
              double * DI_reio )
```

Compute relativistic contribution from reionization and structure formation according to 1) Nozawa et al. 2005 (up to order 3 in theta_e) or 2) Chluba et al. 2012 (up to order ? in ?). Note that, for the moment, this appoximation is only valid for cluster temperatures lower than few KeV.

Parameters

pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
psd	Input: pointer to the distortions structure
T_e	Input: electron temperature in keV
Dtau	Input: optical depth
beta	Input: peculiar velocity of the cluster
beta_z	Input: peculiar velocity of the cluster with respect to the line-of-sight
x	Input: dimensionless frequency
y_reio	Output: y-parameter
DI_reio	Output: spectral distortion

Returns

the error status

Define local variables

Thermal SZ effect (TSZ)

Non-relativistic TSZ

Relativistic TSZ

Kinematic SZ effect (kSZ)

Total distortion

4.5.2.13 distortions_read_br_data()

```
int distortions_read_br_data (
             struct precision * ppr,
             struct distortions * psd )
```

Reads the external file branching_ratios calculated according to Chluba & Jeong 2014

Parameters

ppr	Input: pointer to precision structure
psd	Input: pointer to the distortions structure

Returns

the error status

Define local variables

Open file

Read header

Read number of lines, infer size of arrays and allocate them

Read parameters

4.5.2.14 distortions_spline_br_data()

```
int distortions_spline_br_data (
             struct distortions * psd )
```

Spline the quantitites read in distortions_read_br_data

Parameters

psd	Input: pointer to the distortions structure

Returns

the error status

Allocate second derivatives

Spline branching ratios

4.5.2.15 distortions_interpolate_br_data()

```
int distortions_interpolate_br_data (
             struct distortions * psd,
             double z,
             double * f_g,
             double * f_y,
             double * f_mu,
             double * f_E,
             int * last_index )
```

Interpolate the quantitites splined in distortions_spline_br_data

Parameters

psd	Input: pointer to the distortions structure
z	Input: redshift
f_g	Output: branching ratio for temperature shift
f_y	Output: branching ratio for y distortions
f_mu	Output: branching ratio for mu-distortions
f_E	Output: branching ratio for residuals (multipole expansion)
last_index	Output: multipole of PCA expansion for f_E

Returns

the error status

Define local variables

Find z position

Evaluate corresponding values for the branching ratios

4.5.2.16 distortions_free_br_data()

```
int distortions_free_br_data (
             struct distortions * psd )
```

Free from distortions_read_br_data and distortions_spline_br_data

Parameters

psd	Input: pointer to distortions structure (to be freed)

Returns

the error status

4.5.2.17 distortions_read_sd_data()

```
int distortions_read_sd_data (
            struct precision * ppr,
            struct distortions * psd )
```

Reads the external file distortions_shapes calculated according to Chluba & Jeong 2014

Parameters

ppr	Input: pointer to precision structure
psd	Input: pointer to the distortions structure

Returns

the error status

Define local variables

Open file

Read header

Read number of lines, infer size of arrays and allocate them

Read parameters

4.5.2.18 distortions_spline_sd_data()

```
int distortions_spline_sd_data (
            struct distortions * psd )
```

Spline the quantitites read in distortions_read_sd_data

Parameters

psd	Input: pointer to the distortions structure

Returns

the error status

Allocate second derivatievs

Spline branching ratios

4.5.2.19 distortions_interpolate_sd_data()

```
int distortions_interpolate_sd_data (
            struct distortions * psd,
            double nu,
            double * G_T,
            double * Y_SZ,
            double * M_mu,
            double * S,
            int * index )
```

Interpolate the quantitites splined in distortions_spline_sd_data

Parameters

psd	Input: pointer to the distortions structure
nu	Input: dimnetionless frequency
G_T	Output: shape of temperature shift
Y_SZ	Output: shape of y distortions
M_mu	Output: shape of mu-distortions
S	Output: shape of residuals (multipole expansion)
index	Output: multipole of PCA expansion for S

Returns

the error status

Define local variables

Find z position

Evaluate corresponding values for the branching ratios

4.5.2.20 distortions_free_sd_data()

```
int distortions_free_sd_data (
            struct distortions * psd )
```

Free from distortions_read_sd_data and distortions_spline_sd_data

Parameters

psd	Input: pointer to distortions structure (in which some fields should be freed)

Returns

the error status

4.5.2.21 distortions_output_heat_titles()

```
int distortions_output_heat_titles (
            struct distortions * psd,
            char titles[_MAXTITLESTRINGLENGTH_] )
```

Define title of columns in the heat output

Parameters

psd	Input: pointer to distortions structure
titles	Output: title of each column in the output

4.5.2.22 distortions_output_heat_data()

```
int distortions_output_heat_data (
            struct distortions * psd,
            int number_of_titles,
            double * data )
```

Store data in the heat output

Parameters

psd	Input/Output: pointer to distortions structure
number_of_titles	Input: numbert of column in the output
data	Input: data to be stored

4.5.2.23 distortions_output_sd_titles()

```
int distortions_output_sd_titles (
            struct distortions * psd,
            char titles[_MAXTITLESTRINGLENGTH_] )
```

Define title of columns in the spectral distortion output

Parameters

psd	Input: pointer to distortions structure
titles	Output: title of each column in the output

4.5.2.24 distortions_output_sd_data()

```
int distortions_output_sd_data (
             struct distortions * psd,
             int number_of_titles,
             double * data )
```

Store data in the distortion output

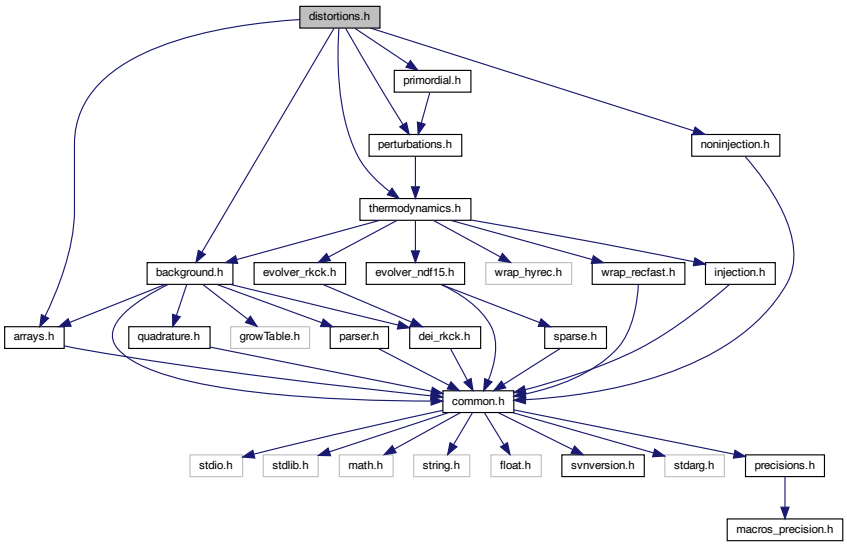
Parameters

psd	Input/Output: pointer to distortions structure
number_of_titles	Input: numbert of column in the output
data	Input: data to be stored

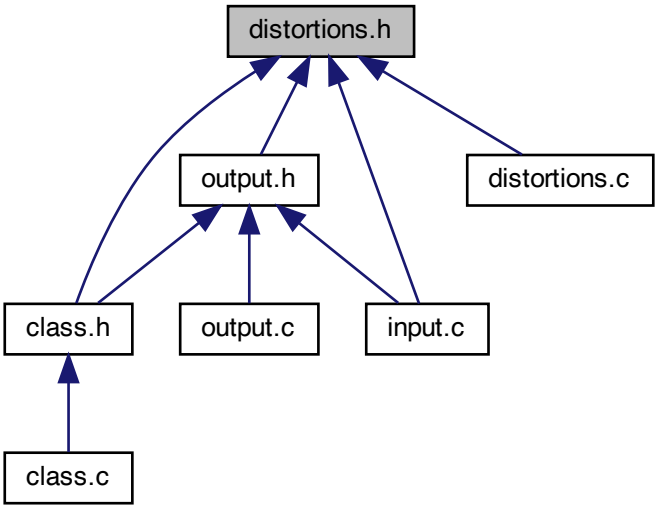
4.6 distortions.h File Reference

```
#include "arrays.h"
#include "background.h"
#include "thermodynamics.h"
#include "perturbations.h"
#include "primordial.h"
#include "noninjection.h"
```

Include dependency graph for distortions.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct distortions

Enumerations

- enum br_approx
- enum reio_approx

4.6.1 Detailed Description

Documented module on spectral distortions Matteo Lucca, 31.10.2018 Nils Schoeneberg, 18.02.2019

4.6.2 Data Structure Documentation

4.6.2.1 struct distortions

distorsions structure, containing all the distortion-related parameters and evolution that other modules need to know.

Data Fields

int	sd_branching_approx	Which approximation to use for the branching ratios?
int	sd_PCA_size	Number of PCA components for the calculation of residual distortions

Data Fields

DetectorFileName	sd_detector_file_name	Name of detector list file
DetectorName	sd_detector_name	Name of detector
double	sd_detector_nu_min	Minimum frequency of chosen detector
double	sd_detector_nu_max	Maximum frequency of chosen detector
double	sd_detector_nu_delta	Bin size of chosen detector
int	sd_detector_bin_number	Number of frequency bins of chosen detector
double	sd_detector_delta_Ic	Sensitivity of the chosen detector
enum reio_approx	sd_reio_type	Calculation method for Sunyaev Zeldovich contributions from re-ionization
double	sd_add_y	Possible additional y contribution (manually) to the SD signal
double	sd_add_mu	Possible additional mu contribution (manually) to the SD signal
double	z_muy	Redshift of the transition of mu to y era
double	z_th	Redshift of the transition from thermal shift to mu era
double	z_min	Minimum redshift
double	z_max	Maximum redshift
int	z_size	Lenght of redshift array
double	z_delta	Redshift intervals
double *	z	Redshift list z[index_z] = list of values
double *	z_weights	Weights for integration over z
double	x_min	Minimum dimentionless frequency
double	x_max	Maximum dimentionless frequency
double	x_delta	dimentionless frequency intervals
int	x_size	Lenght of dimentionless frequency array
double *	x	Dimensionless frequency x[index_x] = list of values
double *	x_weights	Weights for integration over x
double	x_to_nu	Conversion factor nu[GHz] = x_to_nu * x
double	DI_units	Conversion from unitless DI to DI[10^26 W m^-2 Hz^-1 sr^-1]
char	sd_detector_noise_file[2 *_FILENAMESIZE_+_MAX_DETECTOR_NAME_LENGTH_+256]	Full path of detector noise file
DetectorFileName	sd_PCA_file_generator	Full path of PCA generator file
DetectorFileName	sd_detector_list_file	Full path of detector list file
double **	br_table	Branching ratios br_table[index_type][index_z]
double *	sd_parameter_table	Spectral Distortion parameters (g,mu,y,r) sd_parameter_table[index_type]
double **	sd_shape_table	Spectral Distortion shapes (G,M,Y,R) sd_shape_table[index_type][index_x]
double **	sd_table	Spectral Distortion Intensities (final deltaI seperated by component) sd_table[index_type][index_x]
int	index_type_g	temperature shift/g type distortion
int	index_type_mu	mu type distortion
int	index_type_y	y type distortion
int	index_type_PCA	PCA type distortion (first index)

Data Fields

int	type_size	Number of total components for the type array
double	epsilon	
double *	dQrho_dz_tot	
double	Drho_over_rho	
double *	DI	DI[index_x] = list of values
double *	br_exact_z	Redshift array for reading from file br_exact_z[index_z]
int	br_exact_Nz	Number of redshift values for reading from file
double *	f_g_exact	temperature shift/g distortion branching ratio f_g_exact[index_z]
double *	ddf_g_exact	second derivative of the above ddf_g_exact[index_z]
double *	f_y_exact	y distortion branching ratio f_y_exact[index_z]
double *	ddf_y_exact	second derivative of the above ddf_y_exact[index_z]
double *	f_mu_exact	mu distortion shape branching ratio f_mu_exact[index_z]
double *	ddf_mu_exact	second derivative of the above ddf_mu_exact[index_z]
double *	E_vec	PCA component E branching ratio for reading from file E_vec[index_e*br_exact_Nz+index_z] with index_e=[1..8]
double *	ddE_vec	second derivative of the above ddE_vec[index_e*br_exact_Nz+index_z]
int	E_vec_size	number of PCA component E branching ratios
double *	PCA_nu	Frquency array for reading from file PCA_nu[index_nu]
int	PCA_Nnu	Number of frequency values for reading from file
double *	PCA_G_T	temperature shift/g distortion shape PCA_G_T[index_nu]
double *	ddPCA_G_T	second derivative of the above ddPCA_G_T[index_nu]
double *	PCA_Y_SZ	y distortion shape PCA_Y_SZ[index_nu]
double *	ddPCA_Y_SZ	second derivative of the above ddPCA_Y_SZ[index_nu]
double *	PCA_M_mu	mu distortion shape PCA_M_mu[index_nu]
double *	ddPCA_M_mu	second derivative of the above ddPCA_M_mu[index_nu]
double *	S_vec	PCA component S shape for reading from file S_vec[index_s*S_vec_size+index_x] with index_s=[1..8]
double *	ddS_vec	second derivative of the above ddS_vec[index_s*S_vec_size+index_x]
int	S_vec_size	number of PCA component S spectral shapes

Data Fields

double *	delta_lc_array	delta_lc[index_x] for detectors with given sensitivity in each bin
int	has_distortions	do we need to compute spectral distortions?
int	has_user_defined_detector	does the user specify their own detector?
int	has_user_defined_name	does the user specify the name of their detector?
int	has_detector_file	do we have a file for the detector specification?
int	has_SZ_effect	do we include the SZ effect?
int	include_only_exotic	shall we only take exotic injection contributions?
int	include_g_distortion	shall we include the g distortion in the total distortion ?
int	has_noninjected	do we have terms that are not injected (like dissipation of acoustic waves)?
struct noninjection	ni	noninjection file structure
short	distortions_verbose	flag regulating the amount of information sent to standard output (none if set to zero)
ErrorMsg	error_message	zone for writing error messages

4.6.3 Enumeration Type Documentation

4.6.3.1 br_approx

enum br_approx

List of possible branching ratio approximations

4.6.3.2 reio_approx

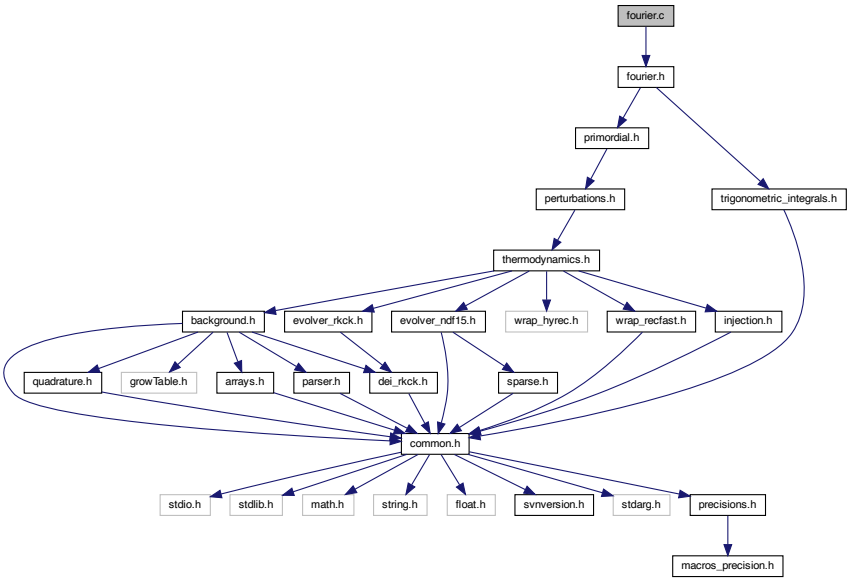
enum reio_approx

List of possible schemes to compute relativistic contribution from reionization and structure formatio

4.7 fourier.c File Reference

```
#include "fourier.h"
```

Include dependency graph for fourier.c:



Functions

- int fourier_pk_at_z (struct background *pba, struct fourier *pfo, enum linear_or_logarithmic mode, enum pk↩ _outputs pk_output, double z, int index_pk, double *out_pk, double *out_pk_ic)
- int fourier_pk_at_k_and_z (struct background *pba, struct primordial *ppm, struct fourier *pfo, enum pk_↩ outputs pk_output, double k, double z, int index_pk, double *out_pk, double *out_pk_ic)
- int fourier_pks_at_kvec_and_zvec (struct background *pba, struct fourier *pfo, enum pk_outputs pk_output, double *kvec, int kvec_size, double *zvec, int zvec_size, double *out_pk, double *out_pk_cb)
- int fourier_pk_tilt_at_k_and_z (struct background *pba, struct primordial *ppm, struct fourier *pfo, enum pk↩ _outputs pk_output, double k, double z, int index_pk, double *pk_tilt)
- int fourier_sigmas_at_z (struct precision *ppr, struct background *pba, struct fourier *pfo, double R, double z, int index_pk, enum out_sigmas sigma_output, double *result)
- int fourier_k_nl_at_z (struct background *pba, struct fourier *pfo, double z, double *k_nl, double *k_nl_cb)
- int fourier_init (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, struct primordial *ppm, struct fourier *pfo)
- int fourier_free (struct fourier *pfo)
- int fourier_indices (struct precision *ppr, struct background *pba, struct perturbations *ppt, struct primordial *ppm, struct fourier *pfo)
- int fourier_get_k_list (struct precision *ppr, struct perturbations *ppt, struct fourier *pfo)
- int fourier_get_tau_list (struct perturbations *ppt, struct fourier *pfo)
- int fourier_get_source (struct background *pba, struct perturbations *ppt, struct fourier *pfo, int index_k, int index_ic, int index_tp, int index_tau, double **sources, double *source)
- int fourier_pk_linear (struct background *pba, struct perturbations *ppt, struct primordial *ppm, struct fourier *pfo, int index_pk, int index_tau, int k_size, double *lnpk, double *lnpk_ic)
- int fourier_sigmas (struct fourier *pfo, double R, double *lnpk_l, double *ddlnpk_l, int k_size, double k_per↩ _decade, enum out_sigmas sigma_output, double *result)
- int fourier_sigma_at_z (struct background *pba, struct fourier *pfo, double R, double z, int index_pk, double k_per_decade, double *result)

- int fourier_halofit (struct precision *ppr, struct background *pba, struct perturbations *ppt, struct primordial *ppm, struct fourier *pfo, int index_pk, double tau, double *pk_nl, double *lnpk_l, double *ddlnpk_l, double *k_nl, short *nl_corr_not_computable_at_this_k)
- int fourier_halofit_integrate (struct fourier *pfo, double *integrand_array, int integrand_size, int ia_size, int index_ia_k, int index_ia_pk, int index_ia_sum, int index_ia_ddsum, double R, enum halofit_integral_type type, double *sum)
- int fourier_hmcode (struct precision *ppr, struct background *pba, struct perturbations *ppt, struct primordial *ppm, struct fourier *pfo, int index_pk, int index_tau, double tau, double *pk_nl, double **lnpk_l, double **ddlnpk_l, double *k_nl, short *nl_corr_not_computable_at_this_k, struct fourier_workspace *pnw)
- int fourier_hmcode_workspace_init (struct precision *ppr, struct background *pba, struct fourier *pfo, struct fourier_workspace *pnw)
- int fourier_hmcode_workspace_free (struct fourier *pfo, struct fourier_workspace *pnw)
- int fourier_hmcode_dark_energy_correction (struct precision *ppr, struct background *pba, struct fourier *pfo, struct fourier_workspace *pnw)
- int fourier_hmcode_baryonic_feedback (struct fourier *pfo)
- int fourier_hmcode_fill_sigtab (struct precision *ppr, struct background *pba, struct perturbations *ppt, struct primordial *ppm, struct fourier *pfo, int index_tau, double *lnpk_l, double *ddlnpk_l, struct fourier_workspace *pnw)
- int fourier_hmcode_fill_growtab (struct precision *ppr, struct background *pba, struct fourier *pfo, struct fourier_workspace *pnw)
- int fourier_hmcode_growint (struct precision *ppr, struct background *pba, struct fourier *pfo, double a, double w0, double wa, double *growth)
- int fourier_hmcode_window_nfw (struct fourier *pfo, double k, double rv, double c, double *window_nfw)
- int fourier_hmcode_halomassfunction (double nu, double *hmf)
- int fourier_hmcode_sigma8_at_z (struct background *pba, struct fourier *pfo, double z, double *sigma_8, double *sigma_8_cb, struct fourier_workspace *pnw)
- int fourier_hmcode_sigmadisp_at_z (struct background *pba, struct fourier *pfo, double z, double *sigma_disp, double *sigma_disp_cb, struct fourier_workspace *pnw)
- int fourier_hmcode_sigmadisp100_at_z (struct background *pba, struct fourier *pfo, double z, double *sigma_disp_100, double *sigma_disp_100_cb, struct fourier_workspace *pnw)
- int fourier_hmcode_sigmaprime_at_z (struct background *pba, struct fourier *pfo, double z, double *sigma_prime, double *sigma_prime_cb, struct fourier_workspace *pnw)

4.7.1 Detailed Description

Documented fourier module

Julien Lesgourgues, 6.03.2014

New module replacing an older one present up to version 2.0 The new module is located in a better place in the main, allowing it to compute non-linear correction to C_l's and not just $P(k)$. It will also be easier to generalize to new methods. The old implementation of one-loop calculations and TRG calculations has been dropped from this version, they can still be found in older versions.

4.7.2 Function Documentation

4.7.2.1 fourier_pk_at_z()

```
int fourier_pk_at_z (
              struct background * pba,
              struct fourier * pfo,
              enum linear_or_logarithmic mode,
              enum pk_outputs pk_output,
              double z,
              int index_pk,
              double * out_pk,
              double * out_pk_ic )
```

Return the P(k,z) for a given redshift z and pk type (_m, _cb) (linear if pk_output = pk_linear, nonlinear if pk_output = pk_nonlinear)

In the linear case, if there are several initial conditions *and* the input pointer out_pk_ic is not set to NULL, the function also returns the decomposition into different IC contributions.

Hints on input index_pk:

a. if you want the total matter spectrum P_m(k,z), pass in input pfo->index_pk_total (this index is always defined)

b. if you want the power spectrum relevant for galaxy or halos, given by P_cb if there is non-cold-dark-matter (e.g. massive neutrinos) and to P_m otherwise, pass in input pfo->index_pk_cluster (this index is always defined)

c. there is another possible syntax (use it only if you know what you are doing): if pfo->has_pk_m == *TRUE* you may pass pfo->index_pk_m to get P_m if pfo->has_pk_cb == *TRUE* you may pass pfo->index_pk_cb to get P_cb

Output format:

1. if mode = logarithmic (most straightforward for the code): out_pk = ln(P(k)) out_pk_ic[diagonal] = ln(P_ic(k)) out_pk_ic[non-diagonal] = cos(correlation angle icxic)
2. if mode = linear (a conversion is done internally in this function) out_pk = P(k) out_pk_ic[diagonal] = P_ic(k) out_pk_ic[non-diagonal] = P_icxic(k)

Parameters

pba	Input: pointer to background structure
pfo	Input: pointer to fourier structure
mode	Input: linear or logarithmic
pk_output	Input: linear or nonlinear
z	Input: redshift
index_pk	Input: index of pk type (_m, _cb)
out_pk	Output: P(k) returned as out_pk_l[index_k]
out_pk_ic	Output: P_ic(k) returned as out_pk_ic[index_k * pfo->ic_ic_size + index_ic1_ic2]

Returns

the error status

- check whether we need the decomposition into contributions from each initial condition

- case z=0 requiring no interpolation in z
- interpolation in z

--> get value of contormal time tau

-> check that tau is in pre-computed table

--> if ln(tau) much too small, raise an error

--> if ln(tau) too small but within tolerance, round it and get right values without interpolating

--> if ln(tau) much too large, raise an error

--> if ln(tau) too large but within tolerance, round it and get right values without interpolating

-> tau is in pre-computed table: interpolate

--> interpolate P_l(k) at tau from pre-computed array

--> interpolate P_ic_l(k) at tau from pre-computed array

--> interpolate P_nl(k) at tau from pre-computed array

- so far, all output stored in logarithmic format. Eventually, convert to linear one.

--> loop over k

--> convert total spectrum

--> convert contribution of each ic (diagonal elements)

--> convert contribution of each ic (non-diagonal elements)

4.7.2.2 fourier_pk_at_k_and_z()

```
int fourier_pk_at_k_and_z (
              struct background * pba,
              struct primordial * ppm,
              struct fourier * pfo,
              enum pk_outputs pk_output,
              double k,
              double z,
              int index_pk,
              double * out_pk,
              double * out_pk_ic )
```

Return the P(k,z) for a given (k,z) and pk type (_m, _cb) (linear if pk_output = pk_linear, nonlinear if pk_output = pk_nonlinear)

In the linear case, if there are several initial conditions *and* the input pointer out_pk_ic is not set to NULL, the function also returns the decomposition into different IC contributions.

Hints on input index_pk:

a. if you want the total matter spectrum P_m(k,z), pass in input pfo->index_pk_total (this index is always defined)

b. if you want the power spectrum relevant for galaxy or halos, given by P_cb if there is non-cold-dark-matter (e.g. massive neutrinos) and to P_m otherwise, pass in input pfo->index_pk_cluster (this index is always defined)

c. there is another possible syntax (use it only if you know what you are doing): if pfo->has_pk_m == *TRUE* you may pass pfo->index_pk_m to get P_m if pfo->has_pk_cb == *TRUE* you may pass pfo->index_pk_cb to get P_cb

Output format:

```
out_pk = P(k)
out_pk_ic[diagonal] = P_ic(k)
out_pk_ic[non-diagonal] = P_icxic(k)
```

Parameters

pba	Input: pointer to background structure
ppm	Input: pointer to primordial structure
pfo	Input: pointer to fourier structure
pk_output	Input: linear or nonlinear
k	Input: wavenumber in 1/Mpc
z	Input: redshift
index_pk	Input: index of pk type (_m, _cb)
out_pk	Output: pointer to P
out_pk_ic	Ouput: P_ic returned as out_pk_ic_l[index_ic1_ic2]

Returns

the error status

- preliminary: check whether we need the decomposition into contributions from each initial condition
- first step: check that k is in valid range [0:kmax] (the test for z will be done when calling fourier_pk_linear_at_z())
- deal with case k = 0 for which P(k) is set to zero (this non-physical result can be useful for interpolations)
- deal with 0 < k <= kmax
- deal with standard case kmin <= k <= kmax

--> First, get P(k) at the right z (in logarithmic format for more accurate interpolation, and then convert to linear format)

--> interpolate total spectrum

--> convert from logarithmic to linear format

--> interpolate each ic component

--> convert each ic component from logarithmic to linear format

--> deal with case 0 < k < kmin that requires extrapolation P(k) = [some number] * k * P_primordial(k) so P(k) = P(kmin) * (k P_primordial(k)) / (kmin P_primordial(kmin)) (note that the result is accurate only if kmin is such that [a0 kmin] << H0)

This is accurate for the synchronous gauge; TODO: write newtonian gauge case. Also, In presence of isocurvature modes, we assumes for simplicity that the mode with index_ic1_ic2=0 dominates at small k: exact treatment should be written if needed.

--> First, get P(k) at the right z (in linear format)

4.7.2.3 fourier_pks_at_kvec_and_zvec()

```
int fourier_pks_at_kvec_and_zvec (
              struct background * pba,
              struct fourier * pfo,
              enum pk_outputs pk_output,
              double * kvec,
              int kvec_size,
              double * zvec,
              int zvec_size,
              double * out_pk,
              double * out_pk_cb )
```

Return the P(k,z) for a grid of (k_i,z_j) passed in input, for all available pk types (_m, _cb), either linear or nonlinear depending on input.

If there are several initial conditions, this function is not designed to return individual contributions.

The main goal of this routine is speed. Unlike fourier_pk_at_k_and_z(), it performs no extrapolation when an input k_i falls outside the pre-computed range [kmin,kmax]: in that case, it just returns P(k,z)=0 for such a k_i

Parameters

pba	Input: pointer to background structure
pfo	Input: pointer to fourier structure
pk_output	Input: pk_linear or pk_nonlinear
kvec	Input: array of wavenumbers in ascending order (in 1/Mpc)
kvec_size	Input: size of array of wavenumbers
zvec	Input: array of redshifts in arbitrary order
zvec_size	Input: size of array of redshifts
out_pk	Output: P(k_i,z_j) for total matter (if available) in Mpc∗∗3
out_pk_cb	Output: P_cb(k_i,z_j) for cdm+baryons (if available) in Mpc∗∗3

Returns

the error status

Summary:

- define local variables
- Allocate arrays
- Construct table of log(P(k_n,z_j)) for pre-computed wavenumbers but requested redshifts:
- Spline it for interpolation along k
- Construct ln(kvec):
- Loop over first k values. If k<kmin, fill output with zeros. If not, go to next step.
- Deal with case kmin<=k<=kmax. For better performance, do not loop through kvec, but through pre-computed k values.

--> Loop through k_i's that fall in interval [k_n,k_n+1]

--> for each of them, perform spine interpolation

- Loop over possible remaining k values with k > kmax, to fill output with zeros.

4.7.2.4 fourier_pk_tilt_at_k_and_z()

```
int fourier_pk_tilt_at_k_and_z (
             struct background * pba,
             struct primordial * ppm,
             struct fourier * pfo,
             enum pk_outputs pk_output,
             double k,
             double z,
             int index_pk,
             double * pk_tilt )
```

Return the logarithmic slope of P(k,z) for a given (k,z), a given pk type (_m, _cb) (computed with linear P_L if pk_output = pk_linear, nonlinear P_NL if pk_output = pk_nonlinear)

Parameters

pba	Input: pointer to background structure
ppm	Input: pointer to primordial structure
pfo	Input: pointer to fourier structure
pk_output	Input: linear or nonlinear
k	Input: wavenumber in 1/Mpc
z	Input: redshift
index_pk	Input: index of pk type (_m, _cb)
pk_tilt	Output: logarithmic slope of P(k,z)

Returns

the error status

4.7.2.5 fourier_sigmas_at_z()

```
int fourier_sigmas_at_z (
             struct precision * ppr,
             struct background * pba,
             struct fourier * pfo,
             double R,
             double z,
             int index_pk,
```

```
            enum out_sigmas sigma_output,
            double * result )
```

This routine computes the variance of density fluctuations in a sphere of radius R at redshift z, sigma(R,z), or other similar derived quantitites, for one given pk type (_m, _cb).

The integral is performed until the maximum value of k_max defined in the perturbation module. Here there is not automatic checking that k_max is large enough for the result to be well converged. E.g. to get an accurate sigma8 at R = 8 Mpc/h, the user should pass at least about P_k_max_h/Mpc = 1.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pfo	Input: pointer to fourier structure
R	Input: radius in Mpc
z	Input: redshift
index_pk	Input: type of pk (_m, _cb)
sigma_output	Input: quantity to be computed (sigma, sigma', ...)
result	Output: result

Returns

the error status

- allocate temporary array for P(k,z) as a function of k
- get P(k,z) as a function of k, for the right z
- spline it along k
- calll the function computing the sigmas
- free allocated arrays

4.7.2.6 fourier_k_nl_at_z()

```
int fourier_k_nl_at_z (
            struct background * pba,
            struct fourier * pfo,
            double z,
            double * k_nl,
            double * k_nl_cb )
```

Return the value of the non-linearity wavenumber k_nl for a given redshift z

Parameters

pba	Input: pointer to background structure
pfo	Input: pointer to fourier structure
z	Input: redshift
k_nl	Output: k_nl value
k_nl_cb	Output: k_nl value of the cdm+baryon part only, if there is ncdm

Returns

the error status

- convert input redshift into a conformal time
- interpolate the precomputed k_nl array at the needed valuetime
- if needed, do the same for the baryon part only

4.7.2.7 fourier_init()

```
int fourier_init (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             struct primordial * ppm,
             struct fourier * pfo )
```

Initialize the fourier structure, and in particular the nl_corr_density and k_nl interpolation tables.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to therodynamics structure
ppt	Input: pointer to perturbation structure
ppm	Input: pointer to primordial structure
pfo	Input/Output: pointer to initialized fourier structure

Returns

the error status

- Do we want to compute P(k,z)? Propagate the flag has_pk_matter from the perturbations structure to the fourier structure
- preliminary tests

--> This module only makes sense for dealing with scalar perturbations, so it should do nothing if there are no scalars

--> Nothing to be done if we don't want the matter power spectrum

--> check applicability of Halofit and HMcode

- define indices in fourier structure (and allocate some arrays in the structure)
- get the linear power spectrum at each time

--> loop over required pk types (_m, _cb)

--> get the linear power spectrum for this time and this type

--> if interpolation of $P(k,\tau)$ will be needed (as a function of tau), compute array of second derivatives in view of spline interpolation

- compute and store sigma8 (variance of density fluctuations in spheres of radius 8/h Mpc at z=0, always computed by convention using the linear power spectrum)
- get the non-linear power spectrum at each time

--> First deal with the case where non non-linear corrections requested

--> Then go through common preliminary steps to the HALOFIT and HMcode methods

--> allocate temporary arrays for spectra at each given time/redshift

--> Then go through preliminary steps specific to HMcode

--> Loop over decreasing time/growing redhsift. For each time/redshift, compute P_NL(k,z) using either Halofit or HMcode

--> fill the array of nonlinear power spectra (only if we are at a late time where P(k) and T(k) are supposed to be stored, i.e., such that z(tau $<$ z_max_pk)

--> spline the array of nonlinear power spectrum

--> free the nonlinear workspace

- if the nl_method could not be identified

4.7.2.8 fourier_free()

```
int fourier_free (
             struct fourier * pfo )
```

Free all memory space allocated by fourier_init().

Parameters

pfo	Input: pointer to fourier structure (to be freed)

Returns

the error status

4.7.2.9 fourier_indices()

```
int fourier_indices (
             struct precision * ppr,
             struct background * pba,
             struct perturbations * ppt,
             struct primordial * ppm,
             struct fourier * pfo )
```

Define indices in the fourier structure, and when possible, allocate arrays in this structure given the index sizes found here

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
ppm	Input: pointer to primordial structure
pfo	Input/Output: pointer to fourier structure

Returns

the error status

- define indices for initial conditions (and allocate related arrays)
- define flags indices for pk types (_m, _cb). Note: due to some dependencies in HMcode, when pfo->index↩ _pk_cb exists, it must come first (e.g. the calculation of the non-linear P_m depends on sigma_cb so the cb-related quantitites must be evaluated first)
- get list of k values
- get list of tau values
- given previous indices, we can allocate the array of linear power spectrum values
- if interpolation of $P(k,\tau)$ will be needed (as a function of tau), compute also the array of second derivatives in view of spline interpolation
- array of sigma8 values
- if non-linear computations needed, allocate array of non-linear correction ratio R_nl(k,z), k_nl(z) and P_nl(k,z) for each P(k) type

4.7.2.10 fourier_get_k_list()

```
int fourier_get_k_list (
             struct precision * ppr,
             struct perturbations * ppt,
             struct fourier * pfo )
```

Copy list of k from perturbation module, and extended it if necessary to larger k for extrapolation (currently this extrapolation is required only by HMcode)

Parameters

ppr	Input: pointer to precision structure
ppt	Input: pointer to perturbation structure
pfo	Input/Output: pointer to fourier structure

Returns

the error status

- if k extrapolation necessary, compute number of required extra values
- otherwise, same number of values as in perturbation module
- allocate array of k
- fill array of k (not extrapolated)
- fill additional values of k (extrapolated)

4.7.2.11 fourier_get_tau_list()

```
int fourier_get_tau_list (
             struct perturbations * ppt,
             struct fourier * pfo )
```

Copy list of tau from perturbation module

Parameters

ppt	Input: pointer to perturbation structure
pfo	Input/Output: pointer to fourier structure

Returns

the error status

-> for linear calculations: only late times are considered, given the value z_max_pk inferred from the ionput

-> for non-linear calculations: we wills store a correction factor for all times

4.7.2.12 fourier_get_source()

```
int fourier_get_source (
             struct background * pba,
             struct perturbations * ppt,
             struct fourier * pfo,
             int index_k,
```

```
int index_ic,
int index_tp,
int index_tau,
double ** sources,
double * source )
```

Get sources for a given wavenumber (and for a given time, type, ic, mode...) either directly from precomputed valkues (computed ain perturbation module), or by analytic extrapolation

Parameters

pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
pfo	Input: pointer to fourier structure
index_k	Input: index of required k value
index_ic	Input: index of required ic value
index_tp	Input: index of required tp value
index_tau	Input: index of required tau value
sources	Input: array containing the original sources
source	Output: desired value of source

Returns

the error status

- use precomputed values
- extrapolate

--> Get last source and k, which are used in (almost) all methods

--> Get previous source and k, which are used in best methods

--> Extrapolate by assuming the source to vanish Has terrible discontinuity

--> Extrapolate starting from the maximum value, assuming growth $\sim$ ln(k) Has a terrible bend in log slope, discontinuity only in derivative

--> Extrapolate starting from the maximum value, assuming growth $\sim$ ln(k) Here we use k in h/Mpc instead of 1/Mpc as it is done in the CAMB implementation of HMcode Has a terrible bend in log slope, discontinuity only in derivative

--> Extrapolate assuming source $\sim$ ln(a$*$k) where a is obtained from the data at k_0 Mostly continuous derivative, quite good

--> Extrapolate assuming source $\sim$ ln(e+a$*$k) where a is estimated like is done in original HMCode

--> If the user has a complicated model and wants to interpolate differently, they can define their interpolation here and switch to using it instead

4.7.2.13 fourier_pk_linear()

```
int fourier_pk_linear (
              struct background * pba,
              struct perturbations * ppt,
              struct primordial * ppm,
              struct fourier * pfo,
              int index_pk,
              int index_tau,
              int k_size,
              double * lnpk,
              double * lnpk_ic )
```

This routine computes all the components of the matter power spectrum P(k), given the source functions and the primordial spectra, at a given time within the pre-computed table of sources (= Fourier transfer functions) of the perturbation module, for a given type (total matter _m or baryon+CDM _cb), and for the same array of k values as in the pre-computed table.

If the input array of k values pfo->ln_k contains wavemumbers larger than those of the pre-computed table, the sources will be extrapolated analytically.

On the opther hand, if the primordial spectrum has sharp features and needs to be sampled on a finer grid than the sources, this function has to be modified to capture the features.

There are two output arrays, because we consider:

- the total matter (_m) or CDM+baryon (_cb) power spectrum
- in the quantitites labelled _ic, the splitting of one of these spectra in different modes for different initial conditions. If the pointer ln_pk_ic is NULL in input, the function will ignore this part; thus, to get the result, one should allocate the array before calling the function. Then the convention is the following:

– the index_ic1_ic2 labels ordered pairs (index_ic1, index_ic2) (since the primordial spectrum is symmetric in (index_ic1, index_ic2)).

– for diagonal elements (index_ic1 = index_ic2) this arrays contains ln[P(k)] where P(k) is positive by construction.

– for non-diagonal elements this arrays contains the k-dependent cosine of the correlation angle, namely P(k)_↩ (index_ic1, index_ic2)/sqrt[P(k)_index_ic1 P(k)_index_ic2]. E.g. for fully correlated or anti-correlated initial conditions, this non-diagonal element is independent on k, and equal to +1 or -1.

Parameters

pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
ppm	Input: pointer to primordial structure
pfo	Input: pointer to fourier structure
index_pk	Input: index of required P(k) type (_m, _cb)
index_tau	Input: index of time
k_size	Input: wavenumber array size
lnpk	Output: log of matter power spectrum for given type/time, for all wavenumbers
lnpk_ic	Output: log of matter power spectrum for given type/time, for all wavenumbers and initial conditions

Returns

the error status

- allocate temporary vector where the primordial spectrum will be stored
- loop over k values

--> get primordial spectrum

--> initialize a local variable for P_m(k) and P_cb(k) to zero

--> here we recall the relations relevant for the nomalization fo the power spectrum: For adiabatic modes, the curvature primordial spectrum thnat we just read was: P_R(k) = 1/(2pi^2) k^3 < R R > Thus the primordial curvature correlator is given by: < R R > = (2pi^2) k^-3 P_R(k) So the delta_m correlator reads: P(k) = < delta_m delta_m > = (source_m)^2 < R R > = (2pi^2) k^-3 (source_m)^2 P_R(k)

For isocurvature or cross adiabatic-isocurvature parts, one would just replace one or two 'R' by 'S_i's

--> get contributions to P(k) diagonal in the initial conditions

--> get contributions to P(k) non-diagonal in the initial conditions

4.7.2.14 fourier_sigmas()

```
int fourier_sigmas (
              struct fourier * pfo,
              double R,
              double * lnpk_l,
              double * ddlnpk_l,
              int k_size,
              double k_per_decade,
              enum out_sigmas sigma_output,
              double * result )
```

Calculate intermediate quantities for hmcode (sigma, sigma', ...) for a given scale R and a given input P(k).

This function has several differences w.r.t. the standard external function non_linear_sigma (format of input, of output, integration stepsize, management of extrapolation at large k, ...) and is overall more precise for sigma(R).

Parameters

pfo	Input: pointer to fourier structure
R	Input: scale at which to compute sigma
lnpk_l	Input: array of ln(P(k))
ddlnpk_l	Input: its spline along k
k_size	Input: dimension of array lnpk_l, normally pfo->k_size, but inside hmcode it its increased by extrapolation to pfo->k_extra_size
k_per_decade	Input: logarithmic step for the integral (recommended: pass ppr->sigma_k_per_decade)
sigma_output	Input: quantity to be computed (sigma, sigma', ...)
result	Output: result

Returns

the error status

- allocate temporary array for an integral over y(x)
- fill the array with values of k and of the integrand
- spline the integrand
- integrate
- preperly normalize the final result
- free allocated array

4.7.2.15 fourier_sigma_at_z()

```
int fourier_sigma_at_z (
             struct background * pba,
             struct fourier * pfo,
             double R,
             double z,
             int index_pk,
             double k_per_decade,
             double * result )
```

This routine computes the variance of density fluctuations in a sphere of radius R at redshift z, sigma(R,z) for one given pk type (_m, _cb).

Try to use instead fourier_sigmas_at_z(). This function is just maintained for compatibility with the deprecated function harmonic_sigma()

The integral is performed until the maximum value of k_max defined in the perturbation module. Here there is not automatic checking that k_max is large enough for the result to be well converged. E.g. to get an accurate sigma8 at R = 8 Mpc/h, the user should pass at least about P_k_max_h/Mpc = 1.

Parameters

pba	Input: pointer to background structure
pfo	Input: pointer to fourier structure
R	Input: radius in Mpc
z	Input: redshift
index_pk	Input: type of pk (_m, _cb)
k_per_decade	Input: logarithmic step for the integral (recommended: pass ppr->sigma_k_per_decade)
result	Output: result

Returns

the error status

- allocate temporary array for P(k,z) as a function of k

- get P(k,z) as a function of k, for the right z
- spline it along k
- calll the function computing the sigmas
- free allocated arrays

4.7.2.16 fourier_halofit()

```
int fourier_halofit (
             struct precision * ppr,
             struct background * pba,
             struct perturbations * ppt,
             struct primordial * ppm,
             struct fourier * pfo,
             int index_pk,
             double tau,
             double * pk_nl,
             double * lnpk_l,
             double * ddlnpk_l,
             double * k_nl,
             short * nl_corr_not_computable_at_this_k )
```

Calculation of the nonlinear matter power spectrum with Halofit (includes Takahashi 2012 + Bird 2013 revisions).

At high redshift it is possible that the non-linear corrections are so small that they can be computed only by going to very large wavenumbers. Thius, for some combination of (z, k_max), the calculation is not possible. In this case a *FALSE* will be returned in the flag halofit_found_k_max.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
ppm	Input: pointer to primordial structure
pfo	Input: pointer to fourier structure
index_pk	Input: index of component are we looking at (total matter or cdm+baryons?)
tau	Input: conformal time at which we want to do the calculation
pk_nl	Output: non linear spectrum at the relevant time
lnpk_l	Input: array of log(P(k)_linear)
ddlnpk_l	Input: array of second derivative of log(P(k)_linear) wrt k, for spline interpolation
k_nl	Output: non-linear wavenumber
nl_corr_not_computable_at_this_k	Ouput: flag concerning the status of the calculation (*TRUE* if not possible)

Returns

the error status

Determine non linear ratios (from pk)

4.7.2.17 fourier_halofit_integrate()

```
int fourier_halofit_integrate (
             struct fourier * pfo,
             double * integrand_array,
             int integrand_size,
             int ia_size,
             int index_ia_k,
             int index_ia_pk,
             int index_ia_sum,
             int index_ia_ddsum,
             double R,
             enum halofit_integral_type type,
             double * sum )
```

Internal routione of Halofit. In original Halofit, this is equivalent to the function wint(). It performs convolutions of the linear spectrum with two window functions.

Parameters

pfo	Input: pointer to non linear structure
integrand_array	Input: array with k, P_L(k) values
integrand_size	Input: one dimension of that array
ia_size	Input: other dimension of that array
index_ia_k	Input: index for k
index_ia_pk	Input: index for pk
index_ia_sum	Input: index for the result
index_ia_ddsum	Input: index for its spline
R	Input: radius
type	Input: which window function to use
sum	Output: result of the integral

Returns

the error status

4.7.2.18 fourier_hmcode()

```
int fourier_hmcode (
             struct precision * ppr,
             struct background * pba,
             struct perturbations * ppt,
             struct primordial * ppm,
             struct fourier * pfo,
             int index_pk,
             int index_tau,
```

```
double tau,
double * pk_nl,
double ** lnpk_l,
double ** ddlnpk_l,
double * k_nl,
short * nl_corr_not_computable_at_this_k,
struct fourier_workspace * pnw )
```

Computes the nonlinear correction on the linear power spectrum via the method presented in Mead et al. 1505.↩ 07833

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
ppm	Input: pointer to primordial structure
pfo	Input: pointer to fourier structure
index_pk	Input: index of the pk type, either index_m or index_cb
index_tau	Input: index of tau, at which to compute the nl correction
tau	Input: tau, at which to compute the nl correction
pk_nl	Output:nonlinear power spectrum
lnpk_l	Input: logarithm of the linear power spectrum for both index_m and index_cb
ddlnpk_l	Input: spline of the logarithm of the linear power spectrum for both index_m and index_cb
nl_corr_not_computable_at_this↩ _k	Ouput: was the computation doable?
k_nl	Output: nonlinear scale for index_m and index_cb
pnw	Input/Output: pointer to nonlinear workspace

Returns

the error status

include precision parameters that control the number of entries in the growth and sigma tables

Compute background quantitites today

If index_pk_cb, choose Omega0_cb as the matter density parameter. If index_pk_m, choose Omega0_cbn as the matter density parameter.

Call all the relevant background parameters at this tau

Test whether pk_cb has to be taken into account (only if we have massive neutrinos)

Get sigma(R=8 Mpc/h), sigma_disp(R=0), sigma_disp(R=100 Mpc/h) and write them into pfo structure

Initialisation steps for the 1-Halo Power Integral

find nonlinear scales k_nl and r_nl and the effective spectral index n_eff

Calculate halo concentration-mass relation conc(mass) (Bullock et al. 2001)

Compute the nonlinear correction

4.7.2.19 fourier_hmcode_workspace_init()

```
int fourier_hmcode_workspace_init (
             struct precision * ppr,
             struct background * pba,
             struct fourier * pfo,
             struct fourier_workspace * pnw )
```

allocate and fill arrays of nonlinear workspace (currently used only by HMcode)

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pfo	Input: pointer to fourier structure
pnw	Output: pointer to nonlinear workspace

Returns

the error status

- allocate arrays of the nonlinear workspace
- fill table with scale independent growth factor

4.7.2.20 fourier_hmcode_workspace_free()

```
int fourier_hmcode_workspace_free (
             struct fourier * pfo,
             struct fourier_workspace * pnw )
```

deallocate arrays in the nonlinear worksapce (currently used only by HMcode)

Parameters

pfo	Input: pointer to fourier structure
pnw	Input: pointer to nonlinear workspace

Returns

the error status

4.7.2.21 fourier_hmcode_dark_energy_correction()

```
int fourier_hmcode_dark_energy_correction (
             struct precision * ppr,
```

```
            struct background * pba,
            struct fourier * pfo,
            struct fourier_workspace * pnw )
```

set the HMcode dark energy correction (if w is not -1)

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pfo	Input: pointer to fourier structure
pnw	Output: pointer to nonlinear workspace

Returns

the error status

- if there is dynamical Dark Energy (w is not -1) modeled as a fluid
- otherwise, we assume no dynamical Dark Energy (w is -1)

4.7.2.22 fourier_hmcode_baryonic_feedback()

```
int fourier_hmcode_baryonic_feedback (
            struct fourier * pfo )
```

set the HMcode baryonic feedback parameters according to the chosen feedback model

Parameters

pfo	Output: pointer to fourier structure

Returns

the error status

4.7.2.23 fourier_hmcode_fill_sigtab()

```
int fourier_hmcode_fill_sigtab (
            struct precision * ppr,
            struct background * pba,
            struct perturbations * ppt,
            struct primordial * ppm,
            struct fourier * pfo,
            int index_tau,
```

```
                double * lnpk_l,
                double * ddlnpk_l,
                struct fourier_workspace * pnw )
```

Function that fills pnw->rtab, pnw->stab and pnw->ddstab with (r, sigma, ddsigma) logarithmically spaced in r. Called by fourier_init at for all tau to account for scale-dependant growth before fourier_hmcode is called

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
ppm	Input: pointer to primordial structure
pfo	Input: pointer to fourier structure
index_tau	Input: index of tau, at which to compute the nl correction
lnpk_l	Input: logarithm of the linear power spectrum for either index_m or index_cb
ddlnpk_l	Input: spline of the logarithm of the linear power spectrum for either index_m or index_cb
pnw	Output: pointer to nonlinear workspace

Returns

the error status

4.7.2.24 fourier_hmcode_fill_growtab()

```
int fourier_hmcode_fill_growtab (
              struct precision * ppr,
              struct background * pba,
              struct fourier * pfo,
              struct fourier_workspace * pnw )
```

Function that fills pnw->tautable and pnw->growtable with (tau, D(tau)) linearly spaced in scalefactor a. Called by fourier_init at before the loop over tau

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure (will provide the scale independent growth factor)
pfo	Input/Output: pointer to fourier structure
pnw	Output: pointer to nonlinear workspace

Returns

the error status

4.7.2.25 fourier_hmcode_growint()

```
int fourier_hmcode_growint (
              struct precision * ppr,
              struct background * pba,
              struct fourier * pfo,
              double a,
              double w0,
              double wa,
              double * growth )
```

This function finds the scale independent growth factor by integrating the approximate relation d(lnD)/d(lna) = Omega_m(z)^gamma by Linder & Cahn 2007

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pfo	Input: pointer to fourier structure
a	Input: scalefactor
w0	Input: dark energy equation of state today
wa	Input: dark energy equation of state varying with a: w=w0+(1-a)wa
growth	Output: scale independent growth factor at a

Returns

the error status

4.7.2.26 fourier_hmcode_window_nfw()

```
int fourier_hmcode_window_nfw (
              struct fourier * pfo,
              double k,
              double rv,
              double c,
              double * window_nfw )
```

This is the fourier transform of the NFW density profile.

Parameters

pfo	Input: pointer to fourier structure
k	Input: wave vector
rv	Input: virial radius
c	Input: concentration = rv/rs (with scale radius rs)
window_nfw	Output: Window Function of the NFW profile

Returns

the error status

4.7.2.27 fourier_hmcode_halomassfunction()

```
int fourier_hmcode_halomassfunction (
             double nu,
             double * hmf )
```

This is the Sheth-Tormen halo mass function (1999, MNRAS, 308, 119)

Parameters

nu	Input: the ν parameter that depends on the halo mass via $\nu(M) = \delta_c/\sigma(M)$
hmf	Output: Value of the halo mass function at this ν

Returns

the error status

4.7.2.28 fourier_hmcode_sigma8_at_z()

```
int fourier_hmcode_sigma8_at_z (
             struct background * pba,
             struct fourier * pfo,
             double z,
             double * sigma_8,
             double * sigma_8_cb,
             struct fourier_workspace * pnw )
```

Compute sigma8(z)

Parameters

pba	Input: pointer to background structure
pfo	Input: pointer to fourier structure
z	Input: redshift
sigma_8	Output: sigma8(z)
sigma_8_cb	Output: sigma8_cb(z)
pnw	Output: pointer to nonlinear workspace

Returns

the error status

4.7.2.29 fourier_hmcode_sigmadisp_at_z()

```
int fourier_hmcode_sigmadisp_at_z (
             struct background * pba,
             struct fourier * pfo,
             double z,
             double * sigma_disp,
             double * sigma_disp_cb,
             struct fourier_workspace * pnw )
```

Compute sigmadisp(z)

Parameters

pba	Input: pointer to background structure
pfo	Input: pointer to fourier structure
z	Input: redshift
sigma_disp	Output: sigmadisp(z)
sigma_disp_cb	Output: sigmadisp_cb(z)
pnw	Output: pointer to nonlinear workspace

Returns

the error status

4.7.2.30 fourier_hmcode_sigmadisp100_at_z()

```
int fourier_hmcode_sigmadisp100_at_z (
             struct background * pba,
             struct fourier * pfo,
             double z,
             double * sigma_disp_100,
             double * sigma_disp_100_cb,
             struct fourier_workspace * pnw )
```

Compute sigmadisp100(z)

Parameters

pba	Input: pointer to background structure
pfo	Input: pointer to fourier structure
z	Input: redshift
sigma_disp_100	Output: sigmadisp100(z)
sigma_disp_100_cb	Output: sigmadisp100_cb(z)
pnw	Output: pointer to nonlinear workspace

Returns

the error status

4.7.2.31 fourier_hmcode_sigmaprime_at_z()

```
int fourier_hmcode_sigmaprime_at_z (
             struct background * pba,
             struct fourier * pfo,
             double z,
             double * sigma_prime,
             double * sigma_prime_cb,
             struct fourier_workspace * pnw )
```

Compute sigma'(z)

Parameters

pba	Input: pointer to background structure
pfo	Input: pointer to fourier structure
z	Input: redshift
sigma_prime	Output: sigma'(z)
sigma_prime_cb	Output: sigma'_cb(z)
pnw	Output: pointer to nonlinear workspace

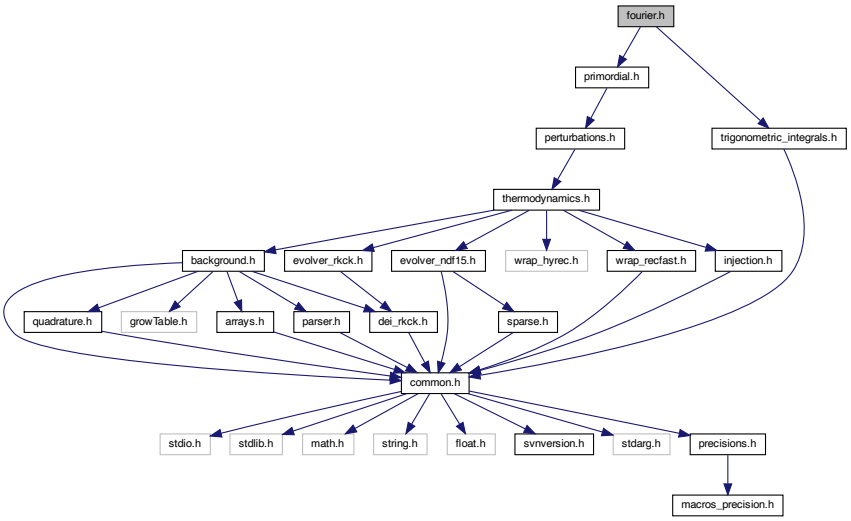
Returns

the error status

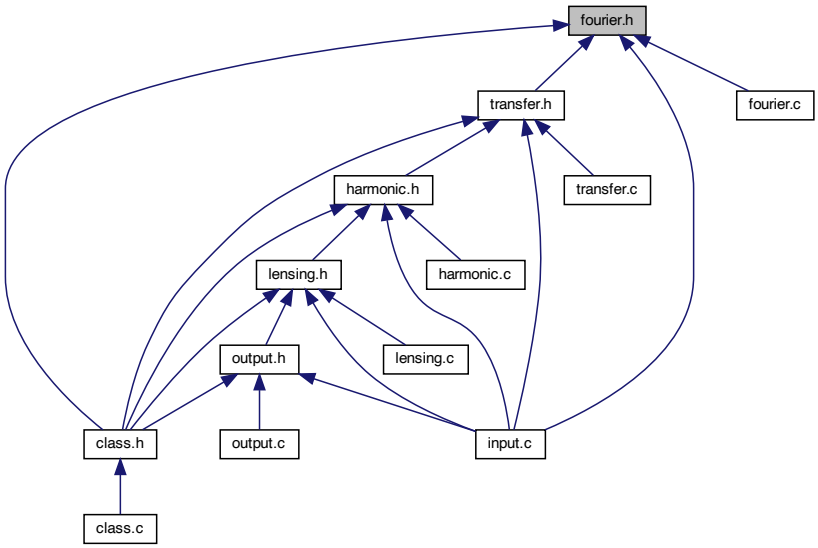
4.8 fourier.h File Reference

```
#include "primordial.h"
#include "trigonometric_integrals.h"
```

Include dependency graph for fourier.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct fourier
- struct fourier_workspace

Macros

- #define _M_EV_TOO_BIG_FOR_HALOFIT_ 10.
- #define _M_SUN_ 1.98847e30

4.8.1 Detailed Description

Documented includes for trg module

4.8.2 Data Structure Documentation

4.8.2.1 struct fourier

Structure containing all information on non-linear spectra.

Once initialized by fourier_init(), contains a table for all two points correlation functions and for all the ai,bj functions (containing the three points correlation functions), for each time and wave-number.

Data Fields

enum non_linear_method	method	method for computing non-linear corrections (none, Halogit, etc.)
enum source_extrapolation	extrapolation_method	method for analytical extrapolation of sources beyond pre-computed range
enum hmcode_baryonic_feedback_model	feedback	
double	c_min	to choose between different baryonic feedback models in hmcode (dmonly, gas cooling, Agn or supernova feedback)
double	eta_0	for HMcode: minimum concentration in Bullock 2001 mass-concentration relation
double	z_infinity	for HMcode: halo bloating parameter
short	has_pk_eq	for HMcode: z value at which Dark Energy correction is evaluated needs to be at early times (default flag: in case wa_fld is defined and non-zero, should we use the pk_eq method?
int	index_md_scalars	set equal to phr->index_md_scalars (useful since this module only deals with scalars)
int	ic_size	for a given mode, ic_size[index_md] = number of initial conditions included in computation
int	ic_ic_size	for a given mode, ic_ic_size[index_md] = number of pairs of (index_ic1, index_ic2) with index_ic2 >= index_ic1; this number is just N(N+1)/2 where N = ic_size[index_md]
short *	is_non_zero	for a given mode, is_non_zero[index_md][index_ic1_ic2] is set to true if the pair of initial conditions (index_ic1, index_ic2) are statistically correlated, or to false if they are uncorrelated
short	has_pk_m	do we want spectra for total matter?
short	has_pk_cb	do we want spectra for cdm+baryons?

Data Fields

int	index_pk_m	index of pk for matter (defined only when has_pk_m is TRUE)
int	index_pk_cb	index of pk for cold dark matter plus baryons (defined only when has_pk_cb is TRUE
int	index_pk_total	always equal to index_pk_m (always defined, useful e.g. for weak lensing spectrum)
int	index_pk_cluster	equal to index_pk_cb if it exists, otherwise to index_pk_m (always defined, useful e.g. for galaxy clustering spectrum)
int	pk_size	k_size = total number of pk
short	has_pk_matter	do we need matter Fourier spectrum?
int	k_size	k_size = total number of k values
double *	k	k[index_k] = list of k values
double *	ln_k	ln_k[index_k] = list of log(k) values
double *	ln_tau	log(tau) array, only needed if user wants some output at z>0, instead of only z=0. This array only covers late times, used for the output of P(k) or T(k), and matching the condition z(tau) < z_max_pk
int	ln_tau_size	number of values in this array

Data Fields

double **	ln_pk_ic_l	Matter power spectrum (linear). Depends on indices index_pk, index_ic1_ic2, index_k, index_tau as: ln_pk_ic_l[index_pk][(index_tau * pfo->k_size + index_k)* pfo->ic_ic_size + index_ic1_ic2] where index-pk labels P(k) types (m = total matter, cb = baryons+CDM), while index_ic1_ic2 labels ordered pairs (index_ic1, index_ic2) (since the primordial spectrum is symmetric in (index_ic1, index_ic2)). • for diagonal elements (index_ic1 = index_ic2) this arrays contains ln[P(k)] where P(k) is positive by construction. • for non-diagonal elements this arrays contains the k-dependent cosine of the correlation angle, namely P(k)_(index_ic1, index_ic2)/sqrt[P(k)_index_ic1 P(k)_index_ic2] This choice is convenient since the sign of the non-diagonal cross-correlation could be negative. For fully correlated or anti-correlated initial conditions, this non-diagonal element is independent on k, and equal to +1 or -1.
double **	ddln_pk_ic_l	second derivative of above array with respect to log(tau), for spline interpolation. So: • for index_ic1 = index_ic, we spline ln[P(k)] vs. ln(k), which is good since this function is usually smooth. • for non-diagonal coefficients, we spline P(k)_(index_ic1, index_ic2)/sqrt[P(k)_index_ic1 P(k)_index_ic2] vs. ln(k), which is fine since this quantity is often assumed to be constant (e.g for fully correlated/anticorrelated initial conditions) or nearly constant, and with arbitrary sign.

Data Fields

double **	ln_pk_l	Total matter power spectrum summed over initial conditions (linear). Only depends on indices index_pk,index_k, index_tau as: ln_pk[index_pk][index_tau * pfo->k_size + index_k]
double **	ddln_pk_l	second derivative of above array with respect to log(tau), for spline interpolation.
double **	ln_pk_nl	Total matter power spectrum summed over initial conditions (nonlinear). Only depends on indices index_pk,index_k, index_tau as: ln_pk[index_pk][index_tau * pfo->k_size + index_k]
double **	ddln_pk_nl	second derivative of above array with respect to log(tau), for spline interpolation.
double *	sigma8	sigma8[index_pk]
int	k_size_extra	
int	tau_size	total number of k values of extrapolated k array (high k) tau_size = number of values
double *	tau	tau[index_tau] = list of time values, covering all the values of the perturbation module
double **	nl_corr_density	nl_corr_density[index_pk][index_tau * ppt->k_size + index_k]
double **	k_nl	wavenumber at which non-linear corrections become important, defined differently by different non_linear_method's
int	index_tau_min_nl	index of smallest value of tau at which nonlinear corrections have been computed (so, for tau<tau_min_nl, the array nl_corr_density only contains some factors 1
int	index_pk_eq_w	index of w in table pk_eq_w_and_Omega
int	index_pk_eq_Omega_m	index of Omega_m in table pk_eq_w_and_Omega
int	pk_eq_size	number of indices in table pk_eq_w_and_Omega
int	pk_eq_tau_size	number of times (and raws in table pk_eq_w_and_Omega)
double *	pk_eq_tau	table of time values
double *	pk_eq_w_and_Omega	table of background quantites
double *	pk_eq_ddw_and_ddOmega	table of second derivatives
short	fourier_verbose	amount of information written in standard output
ErrorMsg	error_message	zone for writing error messages

4.8.2.2 struct fourier_workspace

Structure containing variables used only internally in fourier module by various functions.

Data Fields

double *	rtab	
double *	stab	List of R values
double *	ddstab	List of Sigma Values
double *	growtable	Splined sigma
double *	ztable	
double *	tautable	
double **	sigma_8	
double **	sigma_disp	
double **	sigma_disp_100	
double **	sigma_prime	
double	dark_energy_correction	

4.8.3 Macro Definition Documentation

4.8.3.1 _M_EV_TOO_BIG_FOR_HALOFIT_

```
#define _M_EV_TOO_BIG_FOR_HALOFIT_ 10.
```

above which value of non-CDM mass (in eV) do we stop trusting halofit?

4.8.3.2 _M_SUN_

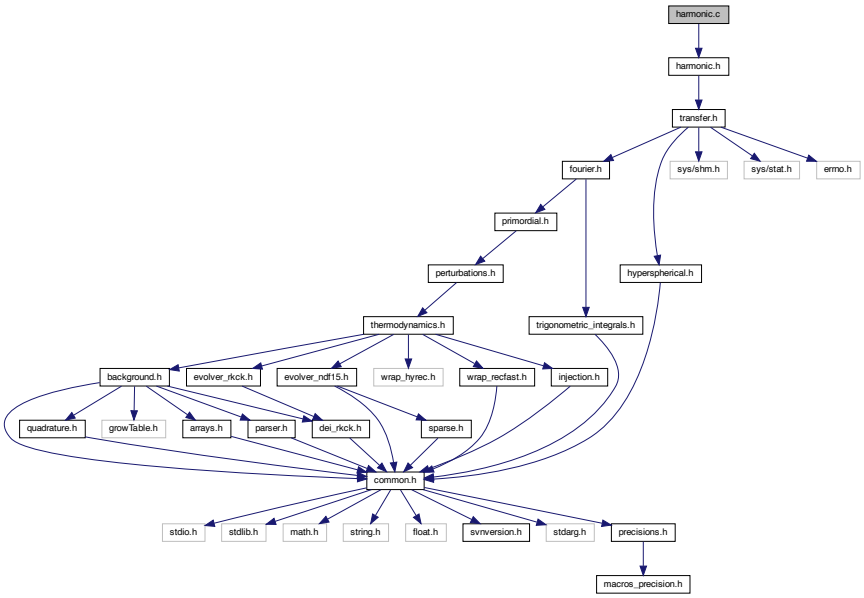
```
#define _M_SUN_ 1.98847e30
```

Solar mass in Kg

4.9 harmonic.c File Reference

```
#include "harmonic.h"
```

Include dependency graph for harmonic.c:



Functions

- int harmonic_cl_at_l (struct harmonic *phr, double l, double *cl_tot, double **cl_md, double **cl_md_ic)
- int harmonic_init (struct precision *ppr, struct background *pba, struct perturbations *ppt, struct primordial *ppm, struct fourier *pfo, struct transfer *ptr, struct harmonic *phr)
- int harmonic_free (struct harmonic *phr)
- int harmonic_indices (struct background *pba, struct perturbations *ppt, struct transfer *ptr, struct primordial *ppm, struct harmonic *phr)
- int harmonic_cls (struct background *pba, struct perturbations *ppt, struct transfer *ptr, struct primordial *ppm, struct harmonic *phr)
- int harmonic_compute_cl (struct background *pba, struct perturbations *ppt, struct transfer *ptr, struct primordial *ppm, struct harmonic *phr, int index_md, int index_ic1, int index_ic2, int index_l, int cl_integrand↩ _num_columns, double *cl_integrand, double *primordial_pk, double *transfer_ic1, double *transfer_ic2)
- int harmonic_pk_at_z (struct background *pba, struct harmonic *phr, enum linear_or_logarithmic mode, double z, double *output_tot, double *output_ic, double *output_cb_tot, double *output_cb_ic)
- int harmonic_pk_at_k_and_z (struct background *pba, struct primordial *ppm, struct harmonic *phr, double k, double z, double *pk_tot, double *pk_ic, double *pk_cb_tot, double *pk_cb_ic)
- int harmonic_pk_nl_at_z (struct background *pba, struct harmonic *phr, enum linear_or_logarithmic mode, double z, double *output_tot, double *output_cb_tot)
- int harmonic_pk_nl_at_k_and_z (struct background *pba, struct primordial *ppm, struct harmonic *phr, double k, double z, double *pk_tot, double *pk_cb_tot)
- int harmonic_fast_pk_at_kvec_and_zvec (struct background *pba, struct harmonic *phr, double *kvec, int kvec_size, double *zvec, int zvec_size, double *pk_tot_out, double *pk_cb_tot_out, int nonlinear)
- int harmonic_sigma (struct background *pba, struct primordial *ppm, struct harmonic *phr, double R, double z, double *sigma)
- int harmonic_sigma_cb (struct background *pba, struct primordial *ppm, struct harmonic *phr, double R, double z, double *sigma_cb)
- int harmonic_tk_at_z (struct background *pba, struct harmonic *phr, double z, double *output)
- int harmonic_tk_at_k_and_z (struct background *pba, struct harmonic *phr, double k, double z, double *output)

4.9.1 Detailed Description

Documented harmonic module

Julien Lesgourgues, 1.11.2019

This module computes the harmonic power spectra C_l^X's given the transfer functions and the primordial spectra.

The following functions can be called from other modules:

1. harmonic_init() at the beginning (but after transfer_init())
2. harmonic_cl_at_l() at any time for computing individual C_l's at any l
3. harmonic_free() at the end

4.9.2 Function Documentation

4.9.2.1 harmonic_cl_at_l()

```
int harmonic_cl_at_l (
              struct harmonic * phr,
              double l,
              double * cl_tot,
              double ** cl_md,
              double ** cl_md_ic )
```

Anisotropy power spectra C_l's for all types, modes and initial conditions.

This routine evaluates all the C_l's at a given value of l by interpolating in the pre-computed table. When relevant, it also sums over all initial conditions for each mode, and over all modes.

This function can be called from whatever module at whatever time, provided that harmonic_init() has been called before, and harmonic_free() has not been called yet.

Parameters

phr	Input: pointer to harmonic structure (containing pre-computed table)
l	Input: multipole number
cl_tot	Output: total C_l's for all types (TT, TE, EE, etc..)
cl_md	Output: C_l's for all types (TT, TE, EE, etc..) decomposed mode by mode (scalar, tensor, ...) when relevant
cl_md_ic	Output: C_l's for all types (TT, TE, EE, etc..) decomposed by pairs of initial conditions (adiabatic, isocurvatures) for each mode (usually, only for the scalar mode) when relevant

Returns

the error status

Summary:

- define local variables
- (a) treat case in which there is only one mode and one initial condition. Then, only cl_tot needs to be filled.
- (b) treat case in which there is only one mode with several initial condition. Fill cl_md_ic[index_md=0] and sum it to get cl_tot.
- (c) loop over modes
- --> (c.1.) treat case in which the mode under consideration has only one initial condition. Fill cl_md[index↩ _md].
- --> (c.2.) treat case in which the mode under consideration has several initial conditions. Fill cl_md_↩ ic[index_md] and sum it to get cl_md[index_md]
- --> (c.3.) add contribution of cl_md[index_md] to cl_tot

4.9.2.2 harmonic_init()

```
int harmonic_init (
              struct precision * ppr,
              struct background * pba,
              struct perturbations * ppt,
              struct primordial * ppm,
              struct fourier * pfo,
              struct transfer * ptr,
              struct harmonic * phr )
```

This routine initializes the harmonic structure (in particular, computes table of anisotropy and Fourier spectra $C_l^X, P(k), ...$)

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure (will provide H, Omega_m at redshift of interest)
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
ppm	Input: pointer to primordial structure
pfo	Input: pointer to fourier structure
phr	Output: pointer to initialized harmonic structure

Returns

the error status

Summary:

- check that we really want to compute at least one spectrum
- initialize indices and allocate some of the arrays in the harmonic structure
- deal with C_l's, if any

- a pointer to the fourier structure is stored in the spectra structure. This odd, unusual and unelegant feature has been introduced in v2.8 in order to keep in use some deprecated functions harmonic_pk_...() that are now pointing at new function fourier_pk_...(). In the future, if the deprecated functions are removed, it will be possible to remove also this pointer.

4.9.2.3 harmonic_free()

```
int harmonic_free (
             struct harmonic * phr )
```

This routine frees all the memory space allocated by harmonic_init().

To be called at the end of each run, only when no further calls to harmonic_cls_at_l(), harmonic_pk_at_z(), harmonic_pk_at_k_and_z() are needed.

Parameters

phr	Input: pointer to harmonic structure (which fields must be freed)

Returns

the error status

4.9.2.4 harmonic_indices()

```
int harmonic_indices (
             struct background * pba,
             struct perturbations * ppt,
             struct transfer * ptr,
             struct primordial * ppm,
             struct harmonic * phr )
```

This routine defines indices and allocates tables in the harmonic structure

Parameters

pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
ppm	Input: pointer to primordial structure
phr	Input/output: pointer to harmonic structure

Returns

the error status

4.9.2.5 harmonic_cls()

```
int harmonic_cls (
             struct background * pba,
             struct perturbations * ppt,
             struct transfer * ptr,
             struct primordial * ppm,
             struct harmonic * phr )
```

This routine computes a table of values for all harmonic spectra C_l's, given the transfer functions and primordial spectra.

Parameters

pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
ppm	Input: pointer to primordial structure
phr	Input/Output: pointer to harmonic structure

Returns

the error status

Summary:

- define local variables
- allocate pointers to arrays where results will be stored
- store values of l
- loop over modes (scalar, tensors, etc). For each mode:
- --> (a) store number of l values for this mode
- --> (b) allocate arrays where results will be stored
- --> (c) loop over initial conditions
- —> loop over l values defined in the transfer module. For each l, compute the C_l's for all types (TT, TE, ...) by convolving primordial spectra with transfer functions. This elementary task is assigned to harmonic_compute_cl()
- --> (d) now that for a given mode, all possible C_l's have been computed, compute second derivative of the array in which they are stored, in view of spline interpolation.

4.9.2.6 harmonic_compute_cl()

```
int harmonic_compute_cl (
              struct background * pba,
              struct perturbations * ppt,
              struct transfer * ptr,
              struct primordial * ppm,
              struct harmonic * phr,
              int index_md,
              int index_ic1,
              int index_ic2,
              int index_l,
              int cl_integrand_num_columns,
              double * cl_integrand,
              double * primordial_pk,
              double * transfer_ic1,
              double * transfer_ic2 )
```

This routine computes the C_l's for a given mode, pair of initial conditions and multipole, but for all types (TT, TE...), by convolving the transfer functions with the primordial spectra.

Parameters

pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
ppm	Input: pointer to primordial structure
phr	Input/Output: pointer to harmonic structure (result stored here)
index_md	Input: index of mode under consideration
index_ic1	Input: index of first initial condition in the correlator
index_ic2	Input: index of second initial condition in the correlator
index_l	Input: index of multipole under consideration
cl_integrand_num_columns	Input: number of columns in cl_integrand
cl_integrand	Input: an allocated workspace
primordial_pk	Input: table of primordial spectrum values
transfer_ic1	Input: table of transfer function values for first initial condition
transfer_ic2	Input: table of transfer function values for second initial condition

Returns

the error status

4.9.2.7 harmonic_pk_at_z()

```
int harmonic_pk_at_z (
              struct background * pba,
              struct harmonic * phr,
              enum linear_or_logarithmic mode,
              double z,
```

```
              double * output_tot,
              double * output_ic,
              double * output_cb_tot,
              double * output_cb_ic )
```

Matter power spectrum for arbitrary redshift and for all initial conditions.

This function is deprecated since v2.8. Try using fourier_pk_at_z() instead.

Parameters

pba	Input: pointer to background structure (used for converting z into tau)
phr	Input: pointer to harmonic structure (containing pre-computed table)
mode	Input: linear or logarithmic
z	Input: redshift
output_tot	Output: total matter power spectrum P(k) in Mpc^3 (linear mode), or its logarithms (logarithmic mode)
output_ic	Output: for each pair of initial conditions, matter power spectra P(k) in Mpc^3 (linear mode), or their logarithms and cross-correlation angles (logarithmic mode)
output_cb_tot	Output: CDM+baryon power spectrum P_cb(k) in Mpc^3 (linear mode), or its logarithms (logarithmic mode)
output_cb_ic	Output: for each pair of initial conditions, CDM+baryon power spectra P_cb(k) in Mpc^3 (linear mode), or their logarithms and cross-correlation angles (logarithmic mode)

Returns

the error status

4.9.2.8 harmonic_pk_at_k_and_z()

```
int harmonic_pk_at_k_and_z (
              struct background * pba,
              struct primordial * ppm,
              struct harmonic * phr,
              double k,
              double z,
              double * pk_tot,
              double * pk_ic,
              double * pk_cb_tot,
              double * pk_cb_ic )
```

Matter power spectrum for arbitrary wavenumber, redshift and initial condition.

This function is deprecated since v2.8. Try using fourier_pk_linear_at_k_and_z() instead.

Parameters

pba	Input: pointer to background structure (used for converting z into tau)
ppm	Input: pointer to primordial structure (used only in the case 0 $<$ k $<$ kmin)
phr	Input: pointer to harmonic structure (containing pre-computed table)

Parameters

k	Input: wavenumber in 1/Mpc
z	Input: redshift
pk_tot	Output: total matter power spectrum P(k) in Mpc^3
pk_ic	Output: for each pair of initial conditions, matter power spectra P(k) in Mpc^3
pk_cb_tot	Output: b+CDM power spectrum P(k) in Mpc^3
pk_cb_ic	Output: for each pair of initial conditions, b+CDM power spectra P(k) in Mpc^3

Returns

the error status

4.9.2.9 harmonic_pk_nl_at_z()

```
int harmonic_pk_nl_at_z (
              struct background * pba,
              struct harmonic * phr,
              enum linear_or_logarithmic mode,
              double z,
              double * output_tot,
              double * output_cb_tot )
```

Non-linear total matter power spectrum for arbitrary redshift.

This function is deprecated since v2.8. Try using fourier_pk_at_z() instead.

Parameters

pba	Input: pointer to background structure (used for converting z into tau)
phr	Input: pointer to harmonic structure (containing pre-computed table)
mode	Input: linear or logarithmic
z	Input: redshift
output_tot	Output: total matter power spectrum P(k) in Mpc^3 (linear mode), or its logarithms (logarithmic mode)
output_cb_tot	Output: b+CDM power spectrum P(k) in Mpc^3 (linear mode), or its logarithms (logarithmic mode)

Returns

the error status

4.9.2.10 harmonic_pk_nl_at_k_and_z()

```
int harmonic_pk_nl_at_k_and_z (
              struct background * pba,
```

```
              struct primordial * ppm,
              struct harmonic * phr,
              double k,
              double z,
              double * pk_tot,
              double * pk_cb_tot )
```

Non-linear total matter power spectrum for arbitrary wavenumber and redshift.

This function is deprecated since v2.8. Try using fourier_pk_at_k_and_z() instead.

Parameters

pba	Input: pointer to background structure (used for converting z into tau)
ppm	Input: pointer to primordial structure (used only in the case 0 $<$ k $<$ kmin)
phr	Input: pointer to harmonic structure (containing pre-computed table)
k	Input: wavenumber in 1/Mpc
z	Input: redshift
pk_tot	Output: total matter power spectrum P(k) in Mpc^3
pk_cb_tot	Output: b+CDM power spectrum P(k) in Mpc^3

Returns

the error status

4.9.2.11 harmonic_fast_pk_at_kvec_and_zvec()

```
int harmonic_fast_pk_at_kvec_and_zvec (
              struct background * pba,
              struct harmonic * phr,
              double * kvec,
              int kvec_size,
              double * zvec,
              int zvec_size,
              double * pk_tot_out,
              double * pk_cb_tot_out,
              int nonlinear )
```

Return the P(k,z) for a grid of (k_i,z_j) passed in input, for all available pk types (_m, _cb), either linear or nonlinear depending on input.

This function is deprecated since v2.8. Try using fourier_pks_at_kvec_and_zvec() instead.

Parameters

pba	Input: pointer to background structure
phr	Input: pointer to harmonic structure
kvec	Input: array of wavenumbers in ascending order (in 1/Mpc)
kvec_size	Input: size of array of wavenumbers
zvec	Input: array of redshifts in arbitrary order
zvec_size	Input: size of array of redshifts
pk_tot_out	Output: P(k_i,z_j) for total matter (if available) in Mpc$**$3
pk_cb_tot_out	Output: P_cb(k_i,z_j) for cdm+baryons (if available) in Mpc$**$3
nonlinear	Input: *TRUE* or *FALSE* (to output nonlinear or linear P(k,z))

Returns

the error status

4.9.2.12 harmonic_sigma()

```
int harmonic_sigma (
             struct background * pba,
             struct primordial * ppm,
             struct harmonic * phr,
             double R,
             double z,
             double * sigma )
```

This routine computes sigma(R) given P(k) for total matter power spectrum (does not check that k_max is large enough)

This function is deprecated since v2.8. Try using fourier_sigmas_at_z() instead.

Parameters

pba	Input: pointer to background structure
ppm	Input: pointer to primordial structure
phr	Input: pointer to harmonic structure
R	Input: radius in Mpc
z	Input: redshift
sigma	Output: variance in a sphere of radius R (dimensionless)

Returns

the error status

4.9.2.13 harmonic_sigma_cb()

```
int harmonic_sigma_cb (
             struct background * pba,
             struct primordial * ppm,
             struct harmonic * phr,
             double R,
             double z,
             double * sigma_cb )
```

This routine computes sigma(R) given P(k) for baryon+cdm power spectrum (does not check that k_max is large enough)

This function is deprecated since v2.8. Try using fourier_sigmas_at_z() instead.

Parameters

pba	Input: pointer to background structure
ppm	Input: pointer to primordial structure
phr	Input: pointer to harmonic structure
R	Input: radius in Mpc
z	Input: redshift
sigma_cb	Output: variance in a sphere of radius R (dimensionless)

Returns

the error status

4.9.2.14 harmonic_tk_at_z()

```
int harmonic_tk_at_z (
             struct background * pba,
             struct harmonic * phr,
             double z,
             double * output )
```

Obsolete function, superseeded by perturbations_sources_at_tau() (at the time of the switch, this function was anyway never used anywhere)

Parameters

pba	Input: pointer to background structure (used for converting z into tau)
phr	Input: pointer to harmonic structure (containing pre-computed table)
z	Input: redshift
output	Output: matter transfer functions

Returns

the error status

4.9.2.15 harmonic_tk_at_k_and_z()

```
int harmonic_tk_at_k_and_z (
             struct background * pba,
             struct harmonic * phr,
             double k,
             double z,
             double * output )
```

Obsolete function, superseeded by perturbations_sources_at_tau() (at the time of the switch, this function was anyway never used anywhere)

Parameters

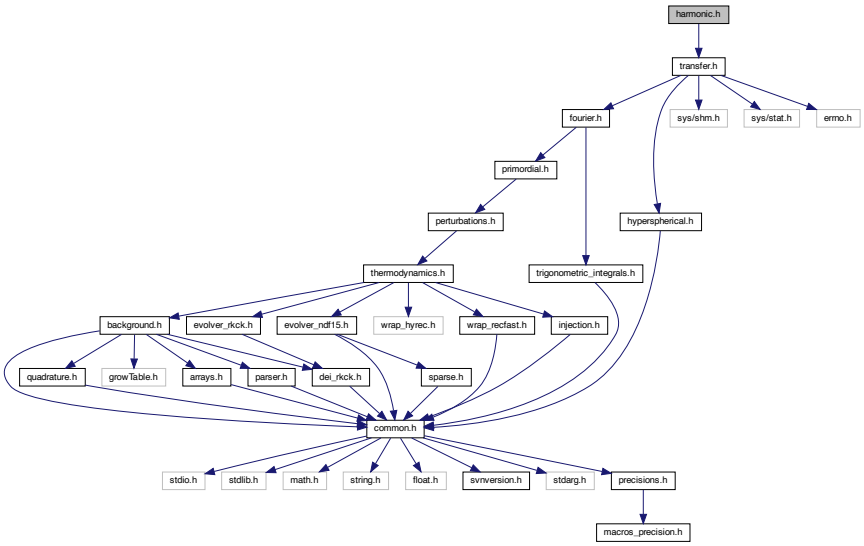
pba	Input: pointer to background structure (used for converting z into tau)
phr	Input: pointer to harmonic structure (containing pre-computed table)
k	Input: wavenumber in 1/Mpc
z	Input: redshift
output	Output: matter transfer functions

Returns

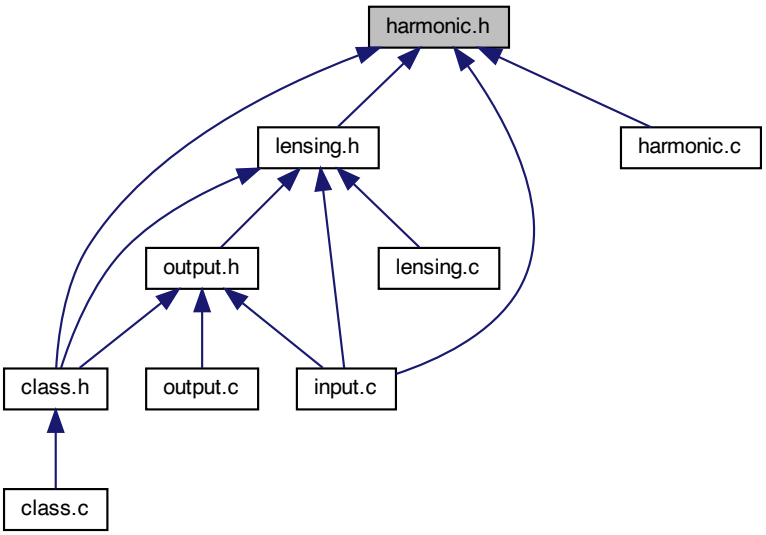
the error status

4.10 harmonic.h File Reference

`#include "transfer.h"`
Include dependency graph for harmonic.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct harmonic

4.10.1 Detailed Description

Documented includes for harmonic module

4.10.2 Data Structure Documentation

4.10.2.1 struct harmonic

Structure containing everything about anisotropy and Fourier power spectra that other modules need to know.

Once initialized by harmonic_init(), contains a table of all C_l's and P(k) as a function of multipole/wavenumber, mode (scalar/tensor...), type (for C_l's: TT, TE...), and pairs of initial conditions (adiabatic, isocurvatures...).

Data Fields

int	non_diag	sets the number of cross-correlation spectra that you want to calculate: 0 means only auto-correlation, 1 means only adjacent bins, and number of bins minus one means all correlations
int	md_size	number of modes (scalar, tensor, ...) included in computation
int	index_md_scalars	index for scalar modes
int *	ic_size	for a given mode, ic_size[index_md] = number of initial conditions included in computation

Data Fields

int *	ic_ic_size	for a given mode, ic_ic_size[index_md] = number of pairs of (index_ic1, index_ic2) with index_ic2 >= index_ic1; this number is just N(N+1)/2 where N = ic_size[index_md]
short **	is_non_zero	for a given mode, is_non_zero[index_md][index_ic1_ic2] is set to true if the pair of initial conditions (index_ic1, index_ic2) are statistically correlated, or to false if they are uncorrelated
int	has_tt	do we want C_l^{TT}? (T = temperature)
int	has_ee	do we want C_l^{EE}? (E = E-polarization)
int	has_te	do we want C_l^{TE}?
int	has_bb	do we want C_l^{BB}? (B = B-polarization)
int	has_pp	do we want $C_l^{\phi\phi}$? (ϕ = CMB lensing potential)
int	has_tp	do we want $C_l^{T\phi}$?
int	has_ep	do we want $C_l^{E\phi}$?
int	has_dd	do we want C_l^{dd}? (d = density)
int	has_td	do we want C_l^{Td}?
int	has_pd	do we want $C_l^{\phi d}$?
int	has_ll	do we want C_l^{ll}? (l = galaxy lensing potential)
int	has_tl	do we want C_l^{Tl}?
int	has_dl	do we want C_l^{dl}?
int	index_ct_tt	index for type C_l^{TT}
int	index_ct_ee	index for type C_l^{EE}
int	index_ct_te	index for type C_l^{TE}
int	index_ct_bb	index for type C_l^{BB}
int	index_ct_pp	index for type $C_l^{\phi\phi}$
int	index_ct_tp	index for type $C_l^{T\phi}$
int	index_ct_ep	index for type $C_l^{E\phi}$
int	index_ct_dd	first index for type C_l^{dd}((d_size*d_size-(d_size-non_diag)*(d_size-non_diag-1)/2) values)
int	index_ct_td	first index for type C_l^{Td}(d_size values)
int	index_ct_pd	first index for type C_l^{pd}(d_size values)
int	index_ct_ll	first index for type C_l^{ll}((d_size*d_size-(d_size-non_diag)*(d_size-non_diag-1)/2) values)
int	index_ct_tl	first index for type C_l^{Tl}(d_size values)
int	index_ct_dl	first index for type C_l^{dl}(d_size values)
int	d_size	number of bins for which density Cl's are computed
int	ct_size	number of C_l types requested
int *	l_size	number of multipole values for each requested mode, l_size[index_md]
int	l_size_max	greatest of all l_size[index_md]
double *	l	list of multipole values l[index_l]
int **	l_max_ct	last multipole (given as an input) at which we want to output C_l's for a given mode and type; l[index_md][l_size[index_md]-1] can be larger than l_max[index_md], in order to ensure a better interpolation with no boundary effects

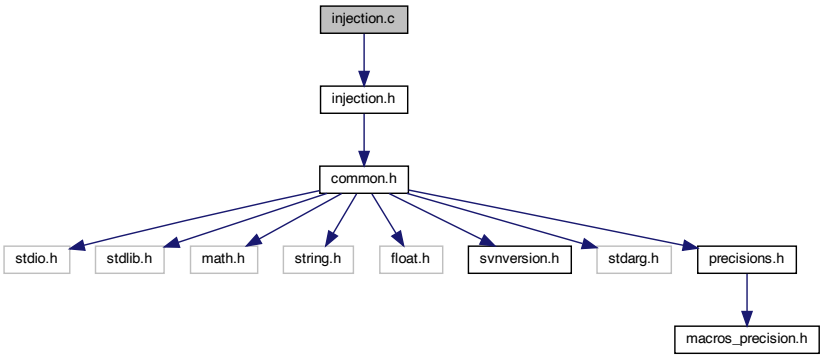
Data Fields

int *	l_max	last multipole (given as an input) at which we want to output C_l's for a given mode (maximized over types); l[index_md][l_size[index_md]-1] can be larger than l_max[index_md], in order to ensure a better interpolation with no boundary effects
int	l_max_tot	last multipole (given as an input) at which we want to output C_l's (maximized over modes and types); l[index_md][l_size[index_md]-1] can be larger than l_max[index_md], in order to ensure a better interpolation with no boundary effects
double **	cl	table of anisotropy spectra for each mode, multipole, pair of initial conditions and types, cl[index_md][(index_l * phr->ic_ic_size[index_md] + index_ic1_ic2) * phr->ct_size + index_ct]
double **	ddcl	second derivatives of previous table with respect to l, in view of spline interpolation
struct fourier *	pfo	a pointer to the fourier structure is stored in the harmonic structure. This odd, unusual and unelegant feature has been introduced in v2.8 in order to keep in use some deprecated functions harmonic_pk_...() that are now pointing at new function fourier_pk_...(). In the future, if the deprecated functions are removed, it will be possible to remove also this pointer.
short	harmonic_verbose	flag regulating the amount of information sent to standard output (none if set to zero)
ErrorMsg	error_message	zone for writing error messages

4.11 injection.c File Reference

```
#include "injection.h"
```

Include dependency graph for injection.c:



4.11.1 Detailed Description

Documented exotic energy injection module

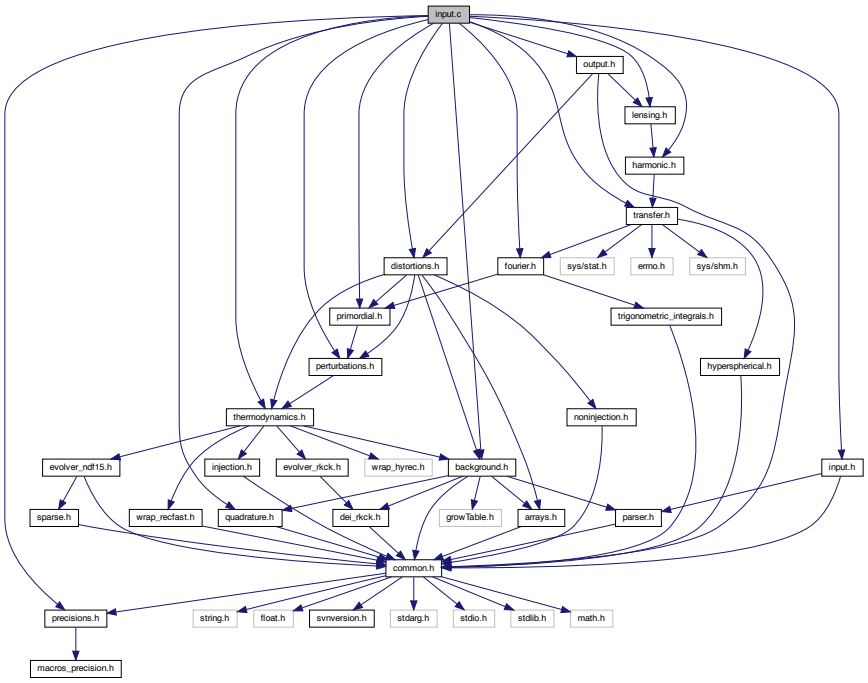
written by Nils Schoeneberg and Matteo Lucca, 27.02.2019

The main goal of this module is to calculate the deposited energy in form of heating, ionization and Lyman alpha processes from exotic energy injection processes. For more details see the description in the README file.

4.12 input.c File Reference

```
#include "input.h"
#include "quadrature.h"
#include "background.h"
#include "thermodynamics.h"
#include "perturbations.h"
#include "transfer.h"
#include "primordial.h"
#include "harmonic.h"
#include "fourier.h"
#include "lensing.h"
#include "distortions.h"
#include "output.h"
#include "precisions.h"
```

Include dependency graph for input.c:



Functions

- int input_init (int argc, char **argv, struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, struct transfer *ptr, struct primordial *ppm, struct harmonic *phr, struct fourier *pfo, struct lensing *ple, struct distortions *psd, struct output *pop, ErrorMsg errmsg)
- int input_find_file (int argc, char **argv, struct file_content *fc, ErrorMsg errmsg)
- int input_set_root (char *input_file, struct file_content **ppfc_input, struct file_content *pfc_setroot, ErrorMsg errmsg)
- int input_read_from_file (struct file_content *pfc, struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, struct transfer *ptr, struct primordial *ppm, struct harmonic *phr, struct fourier *pfo, struct lensing *ple, struct distortions *psd, struct output *pop, ErrorMsg errmsg)
- int input_shooting (struct file_content *pfc, struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, struct transfer *ptr, struct primordial *ppm, struct harmonic *phr, struct fourier *pfo, struct lensing *ple, struct distortions *psd, struct output *pop, int input_verbose, int *has_shooting, ErrorMsg errmsg)

- int input_needs_shooting_for_target (struct file_content *pfc, enum target_names target_name, double target_value, int *needs_shooting, ErrorMsg errmsg)
- int input_find_root (double *xzero, int *fevals, double tol_x_rel, struct fzerofun_workspace *pfzw, ErrorMsg errmsg)
- int input_fzerofun_1d (double input, void *pfzw, double *output, ErrorMsg error_message)
- int input_fzero_ridder (int(*func)(double x, void *param, double *y, ErrorMsg error_message), double x1, double x2, double xtol, void *param, double *Fx1, double *Fx2, double *xzero, int *fevals, ErrorMsg error_message)
- int input_get_guess (double *xguess, double *dxdy, struct fzerofun_workspace *pfzw, ErrorMsg errmsg)
- int input_try_unknown_parameters (double *unknown_parameter, int unknown_parameters_size, void *voidpfzw, double *output, ErrorMsg errmsg)
- int input_read_precisions (struct file_content *pfc, struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, struct transfer *ptr, struct primordial *ppm, struct harmonic *phr, struct fourier *pfo, struct lensing *ple, struct distortions *psd, struct output *pop, ErrorMsg errmsg)
- int input_read_parameters (struct file_content *pfc, struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, struct transfer *ptr, struct primordial *ppm, struct harmonic *phr, struct fourier *pfo, struct lensing *ple, struct distortions *psd, struct output *pop, ErrorMsg errmsg)
- int input_read_parameters_general (struct file_content *pfc, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, struct distortions *psd, ErrorMsg errmsg)
- int input_read_parameters_species (struct file_content *pfc, struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, int input_verbose, ErrorMsg errmsg)
- int input_read_parameters_injection (struct file_content *pfc, struct precision *ppr, struct thermodynamics *pth, ErrorMsg errmsg)
- int input_read_parameters_nonlinear (struct file_content *pfc, struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, struct fourier *pfo, int input_verbose, ErrorMsg errmsg)
- int input_prepare_pk_eq (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct fourier *pfo, int input_verbose, ErrorMsg errmsg)
- int input_read_parameters_primordial (struct file_content *pfc, struct perturbations *ppt, struct primordial *ppm, ErrorMsg errmsg)
- int input_read_parameters_spectra (struct file_content *pfc, struct precision *ppr, struct background *pba, struct primordial *ppm, struct perturbations *ppt, struct transfer *ptr, struct harmonic *phr, struct output *pop, ErrorMsg errmsg)
- int input_read_parameters_lensing (struct file_content *pfc, struct precision *ppr, struct perturbations *ppt, struct transfer *ptr, struct lensing *ple, ErrorMsg errmsg)
- int input_read_parameters_distortions (struct file_content *pfc, struct precision *ppr, struct distortions *psd, ErrorMsg errmsg)
- int input_read_parameters_additional (struct file_content *pfc, struct precision *ppr, struct background *pba, struct thermodynamics *pth, ErrorMsg errmsg)
- int input_read_parameters_output (struct file_content *pfc, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, struct transfer *ptr, struct primordial *ppm, struct harmonic *phr, struct fourier *pfo, struct lensing *ple, struct distortions *psd, struct output *pop, ErrorMsg errmsg)
- int input_write_info (struct file_content *pfc, struct output *pop, ErrorMsg errmsg)
- int input_default_params (struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, struct transfer *ptr, struct primordial *ppm, struct harmonic *phr, struct fourier *pfo, struct lensing *ple, struct distortions *psd, struct output *pop)

4.12.1 Detailed Description

Documented input module.

Julien Lesgourgues, 27.08.2010

- internal organization of the module structured and improved by Nils Schoeneberg and Matteo Lucca, 07.03.2019

4.12.2 Function Documentation

4.12.2.1 input_init()

```
int input_init (
             int argc,
             char ** argv,
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             struct transfer * ptr,
             struct primordial * ppm,
             struct harmonic * phr,
             struct fourier * pfo,
             struct lensing * ple,
             struct distortions * psd,
             struct output * pop,
             ErrorMsg errmsg )
```

Initialize input parameters from external file.

Parameters

argc	Input: Number of command line arguments
argv	Input: Command line argument strings
ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
ppm	Input: pointer to primordial structure
phr	Input: pointer to harmonic structure
pfo	Input: pointer to fourier structure
ple	Input: pointer to lensing structure
psd	Input: pointer to distorsion structure
pop	Input: pointer to output structure
errmsg	Input/Output: Error message

Returns

the error status

Summary:

Define local variables

Find and read input file

Initialize all parameters given the input 'file_content' structure. If its size is null, all parameters take their default values.

Free local struture

4.12.2.2 input_find_file()

```
int input_find_file (
             int argc,
             char ** argv,
             struct file_content * fc,
             ErrorMsg errmsg )
```

Find and read external file (xxx.ini or xxx.pre) containing the input parameters. All data is stored in the local structure 'file_content'.

Parameters

argc	Input: Number of command line arguments
argv	Input: Command line argument strings
fc	Output: file_content structure
errmsg	Input/Output: Error message

Returns

the error status

Summary:

Define local variables

Initialize the two file_content structures (for input parameters and precision parameters) to some null content. If no arguments are passed, they will remain null and inform input_init that all parameters take default values.

If some arguments are passed, identify eventually some 'xxx.ini' and 'xxx.pre' files, and store their name.

If there is an 'xxx.ini' file, read it and store its content.

If there is an 'xxx.pre' file, read it and store its content.

If one or two files were read, merge their contents in a single 'file_content' structure.

Free local strutures

4.12.2.3 input_set_root()

```
int input_set_root (
             char * input_file,
             struct file_content ** ppfc_input,
             struct file_content * pfc_setroot,
             ErrorMsg errmsg )
```

Sets the 'root' variable in the input file content (this will be the beginning of the name of all output files for the current CLASS run)

Parameters

input_file	Input: filename of the input file
ppfc_input	Input/Output: pointer to (pointer to input file structure)
pfc_setroot	Input: pointer to an allocated temporary file content that will be used here
errmsg	Input/Output: the error message

Returns

the error status

Define local variables

Check whether a root name has been set, and wether overwrite_root is true

If root has not been set, use the default of 'output/<this-filename>'

If we don't want to overwrite the root name, check now for the existence of output for the given root name + N

For each 'filenum', test if it exists. Only stop if it has not been found.

If we do want to overwrite, just take the given root name

4.12.2.4 input_read_from_file()

```
int input_read_from_file (
             struct file_content * pfc,
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             struct transfer * ptr,
             struct primordial * ppm,
             struct harmonic * phr,
             struct fourier * pfo,
             struct lensing * ple,
             struct distortions * psd,
             struct output * pop,
             ErrorMsg errmsg )
```

Initialize each parameter, first to its default values, and then from what can be interpreted from the values passed in the input 'file_content' structure. If its size is null, all parameters keep their default values.

Parameters

pfc	Input: pointer to local structure
ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
ppm	Input: pointer to primordial structure
phr	Input: pointer to harmonic structure
pfo	Input: pointer to fourier structure
ple	Input: pointer to lensing structure
psd	Input: pointer to distorsion structure
pop	Input: pointer to output structure
errmsg	Input/Output: Error message

Returns

the error status

Summary:

- Define local variables

Set default values Before getting into the assignment of parameters and the shooting, we want to already fix our precision parameters. No precision parameter should depend on any input parameter

Find out if shooting necessary and, eventually, shoot and initialize read parameters

If no shooting is necessary, initialize read parameters without it

Write info on the read/unread parameters. This is the correct place to do it, since we want it to happen after all the shooting business, and after the final reading of all parameters

4.12.2.5 input_shooting()

```
int input_shooting (
              struct file_content * pfc,
              struct precision * ppr,
              struct background * pba,
              struct thermodynamics * pth,
              struct perturbations * ppt,
              struct transfer * ptr,
              struct primordial * ppm,
              struct harmonic * phr,
              struct fourier * pfo,
              struct lensing * ple,
              struct distortions * psd,
              struct output * pop,
              int input_verbose,
              int * has_shooting,
              ErrorMsg errmsg )
```

In CLASS, we call 'shooting' the process of doing preliminary runs of parts of the code in order to find numerically the value of an input variable which cannot be inferred analytically from other input variables passed by the user.

A typical example is when the user passes theta_s, the angular scale of the sound horizon at decoupling. This quantity be passed instead of the hubble parameter h, but only if we run CLASS until the thermodynamics module to figure out how h and theta_s relate numerically. The code starts from a guess for h, and runs to find the corresponding theta_s. It adjusts h, shoots again, and repeats this process until it finds some h giving the correct theta_s within some tolerance.

This function contains the overall structure to handle these steps.

Parameters

pfc	Input/Output: pointer to file content, with input parameters before/after the shooting
ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure

Parameters

ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
ppm	Input: pointer to primordial structure
phr	Input: pointer to harmonic structure
pfo	Input: pointer to fourier structure
ple	Input: pointer to lensing structure
psd	Input: pointer to distorsion structure
pop	Input: pointer to output structure
input_verbose	Input: Verbosity of input
has_shooting	Output: do we need shooting?
errmsg	Input/Output: Error message

Returns

the error status

Summary:

Define local variables

Do we need to fix unknown parameters?

In the case of unknown parameters, start shooting...

Go through all cases with unknown parameters

If there is only one parameter, we use a more efficient Newton method for 1D cases

Otherwise we do multidimensional shooting

Read all parameters from the fc obtained through shooting

Set status of shooting

Free arrays allocated

4.12.2.6 input_needs_shooting_for_target()

```
int input_needs_shooting_for_target (
             struct file_content * pfc,
             enum target_names target_name,
             double target_value,
             int * needs_shooting,
             ErrorMsg errmsg )
```

Related to 'shooting': for each target, check whether it is sufficient to stick to the default value of the unkown parameter (for instance: if the target parameter is a density and the target value is zero, the unkown parameter should remain zero like in the default)

Parameters

pfc	Input: pointer to local structure
target_name	Input: list of possible target names
target_value	Input: list of possible target values
needs_shooting	Output: needs shooting?
errmsg	Input/Output: Error message

Returns

the error status

4.12.2.7 input_find_root()

```
int input_find_root (
              double * xzero,
              int * fevals,
              double tol_x_rel,
              struct fzerofun_workspace * pfzw,
              ErrorMsg errmsg )
```

Related to 'shooting': Find the root of a one-dimensional function. This function starts from a first guess, then uses a few steps to bracket the root, and then calls another function to actually get the root.

Parameters

xzero	Output: root x such that f(x)=0 up to tolerance (f(x) = input_fzerofun_1d)
fevals	Output: number of iterations (that is, of CLASS runs) needed to find the root
tol_x_rel	Input : Relative tolerance compared to bracket of root that is used to find root.
pfzw	Input : pointer to workspace containing targets, unkown parameters and other relevant information
errmsg	Input/Output: Error message

Returns

the error status

Summary:

Define local variables

Fisrt we do our guess

Then we do a linear hunt for the boundaries

Find root using Ridders method (Exchange for bisection if you are old-school)

4.12.2.8 input_fzerofun_1d()

```
int input_fzerofun_1d (
              double input,
              void * pfzw,
              double * output,
              ErrorMsg error_message )
```

Related to 'shooting': defines 1d function of which we want to find the root during the shooting. The function is simply: "prediction of CLASS for a target parameter y given a parameter x - targeted value of y"

Parameters

input	Input: value of x
pfzw	Input: pointer to workspace containing targets, unkown parameters and other relevant information
output	Ouput: f(x) = y - y_targeted
error_message	Input/Output: Error message

Returns

the error status

4.12.2.9 input_fzero_ridder()

```
int input_fzero_ridder (
              int(*)(double x, void *param, double *y, ErrorMsg error_message) func,
              double x1,
              double x2,
              double xtol,
              void * param,
              double * Fx1,
              double * Fx2,
              double * xzero,
              int * fevals,
              ErrorMsg error_message )
```

Related to 'shooting': using Ridders' method, return the root x of a function f(x) known to lie between x1 and x2, up to some tolerance. Note that this function is very generic and could easily be moved to the tools (and be used in other modules).

Parameters

func	Input: function y=f(x), with arguments: x, pointer to y, and another pointer containing several fixed parameters
x1	Input: lower boundary x1<x
x2	Input: upper boundary x<x2
xtol	Input: tolerance: |x- true root|<xtol
param	Input: fixed parameters passed to f(x)
Fx1	Input: f(x1)
Fx2	Input: f(x2)
xzero	Output: root x
fevals	Output: number of iterations (that is, of CLASS runs) needed to find the root
error_message	Input/Output: Error message

Returns

the error status

Summary:

Define local variables

4.12.2.10 input_get_guess()

```
int input_get_guess (
             double * xguess,
             double * dxdy,
             struct fzerofun_workspace * pfzw,
             ErrorMsg errmsg )
```

Related to 'shooting': we define here a reasonable analytic guess for each unknown parameter as a function of its target parameter. We must also estimate dxdy, i.e. how the unknown parameter responds to the target parameter. This can simply be estimated as the derivative of the guess formula.

Parameters

xguess	Output: guess for unkown parameter x given target parameter y
dxdy	Output: guess for derivative dx/dy
pfzw	Input : pointer to workspace containing targets, unkown parameters and other relevant information
errmsg	Input/Output: Error message

Returns

the error status

Summary:

Define local variables

Estimate dxdy

Update pb to reflect guess

- Deallocate everything allocated by input_read_parameters

4.12.2.11 input_try_unknown_parameters()

```
int input_try_unknown_parameters (
             double * unknown_parameter,
             int unknown_parameters_size,
             void * voidpfzw,
             double * output,
             ErrorMsg errmsg )
```

Related to 'shooting': when there is one or more targets, call CLASS up to the highest needed computation stage, for a given set of unknown parameters; obtain the corresponding target parameters; and return the vector of each [target - targeted_value].

Parameters

unknown_parameter	Input: vector of unkownn parameters x
unknown_parameters_size	Input: size of this vector
voidpfzw	Input: pointer to workspace containing targets, unkown parameters and other relevant information
output	Output: vector of target parameters y
errmsg	Input/Output: Error message

Returns

the error status

Summary

Define local variables

Read input parameters

Optimise flags for sigma8 calculation.

Shoot forward into class up to required stage

Get the corresponding shoot variable and put into output

In case scalar field is used to fill, pba->Omega0_scf is not equal to pfzw->target_value[i].

Free structures

Set filecontent to unread

Free pointers allocated on input if neccessary

Some pointers in ppt may not be allocated if has_perturbations is *FALSE*, but this is handled in perturbations_↩ free_input as neccessary.

4.12.2.12 input_read_precisions()

```
int input_read_precisions (
             struct file_content * pfc,
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             struct transfer * ptr,
             struct primordial * ppm,
             struct harmonic * phr,
             struct fourier * pfo,
             struct lensing * ple,
             struct distortions * psd,
             struct output * pop,
             ErrorMsg errmsg )
```

Initialize the precision parameter structure.

All precision parameters used in the other modules are listed here and assigned here a default value.

Parameters

pfc	Input: pointer to local structure
ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer to perturbations structure
ptr	Input: pointer to transfer structure
ppm	Input: pointer to primordial structure
phr	Input: pointer to harmonic structure
pfo	Input: pointer to non-linear structure
ple	Input: pointer to lensing structure
pop	Input: pointer to output structure
psd	Input: pointer to distorsion structure
errmsg	Input: Error message

Returns

the error status

Summary:

- Define local variables
- Automatic estimate of machine precision

Read all precision parameters from input (these very concise lines parse all precision parameters thanks to the macros defined in macros_precision.h)

4.12.2.13 input_read_parameters()

```
int input_read_parameters (
             struct file_content * pfc,
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             struct transfer * ptr,
             struct primordial * ppm,
             struct harmonic * phr,
             struct fourier * pfo,
             struct lensing * ple,
             struct distortions * psd,
             struct output * pop,
             ErrorMsg errmsg )
```

If entries are passed in file_content structure, carefully read and interpret each of them, and tune the relevant input parameters accordingly

Parameters

pfc	Input: pointer to local structure
ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
ppm	Input: pointer to primordial structure
phr	Input: pointer to harmonic structure
pfo	Input: pointer to fourier structure
ple	Input: pointer to lensing structure
psd	Input: pointer to distorsion structure
pop	Input: pointer to output structure
errmsg	Input: Error message

Returns

the error status

Summary:

Define local variables

Set all input parameters to default values

Read verbose for input structure

Read the general parameters of the background, thermodynamics, and perturbation structures This function is exclusively for those parameters, NOT related to any physical species

Read the parameters for each physical species (has to be called after the general read)

Read parameters for exotic energy injection quantities

Read parameters for nonlinear quantities

Read parameters for primordial quantities

Read parameters for spectra quantities

Read parameters for lensing quantities

Read parameters for distortions quantities

Read obsolete parameters

Read parameters for output quantities

4.12.2.14 input_read_parameters_general()

```
int input_read_parameters_general (
             struct file_content * pfc,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             struct distortions * psd,
             ErrorMsg errmsg )
```

Read general parameters related to class, including

- background, thermo, and perturbation quantities NOT associated to any particular species
- calculationary quantities like the gauge/recombination code
- output options

Parameters

pfc	Input: pointer to local structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer to perturbation structure
psd	Input: pointer to distorsion structure
errmsg	Input: Error message

Returns

the error status

Summary:

- Define local variables

1) List of output spectra requested

1.a) Terms contributing to the temperature spectrum

1.a.1) Split value of redshift z at which the isw is considered as late or early

1.b) Obsevable number count fluctuation spectrum

1.c) Transfer function of additional metric fluctuations

2) Perturbed recombination

3) Modes

3.a) List of initial conditions for scalars

3.b) List of initial conditions for scalars

4) Gauge

4.a) Set gauge

4.b) Do we want density and velocity transfer functions in Nbody gauge?

5) h in [-] and H_0/c in [1/Mpc = h/2997.9 = h∗10^5/c]

6) Primordial helium fraction

7) Recombination parameters

7.a) Photo-ionization dependence for recfast

8) Reionization parametrization

8.a) Reionization parameters if reio_parametrization=reio_camb

8.b) Reionization parameters if reio_parametrization=reio_bins_tanh

8.c) reionization parameters if reio_parametrization=reio_many_tanh

8.d) reionization parameters if reio_parametrization=reio_many_tanh

9) Damping scale

10) Varying fundamental constants

4.12.2.15 input_read_parameters_species()

```
int input_read_parameters_species (
             struct file_content * pfc,
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             int input_verbose,
             ErrorMsg errmsg )
```

Read the parameters for each physical species

Parameters

pfc	Input: pointer to local structure
ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer to perturbation structure
input_verbose	Input: verbosity of input
errmsg	Input: Error message

Returns

the error status

Summary:

- Define local variables

1) Omega_0_g (photons) and T_cmb

2) Omega_0_b (baryons)

3) Omega_0_ur (ultra-relativistic species / massless neutrino)

We want to keep compatibility with old input files, and as such 'N_eff' is still an allowed parameter name, although it is deprecated and its use is discouraged.

3.a) Case of non-standard properties

4) Omega_0_cdm (CDM)

5) Non-cold relics (ncdm)

5.a) Number of non-cold relics

5.b) Check if filenames for interpolation tables are given

5.b.1) Check if filenames for interpolation tables are given

5.c) (optional) p.s.d.-parameters

5.d) Mass or Omega of each ncdm species

5.e) Temperatures

5.f) Chemical potentials

5.g) Degeneracy of each ncdm species

5.h) Quadrature modes, 0 is qm_auto

5.h.1) qmax, if relevant

5.h.2) Number of momentum bins

Last step of 5) (i.e. NCDM) – Calculate the masses and momenta

6) Omega_0_k (effective fractional density of curvature)

7.1) Decaying DM into DR

7.1.a) Omega_0_dcdmdr (DCDM, i.e. decaying CDM)

7.1.b) Omega_ini_dcdm or omega_ini_dcdm

7.1.c) Gamma in same units as H0, i.e. km/(s Mpc)

7.2) Interacting dark matter & dark radiation, ETHOS-parametrization/NADM parametrization, see explanatory.ini

7.2.a) Omega_0_idr

7.2.b) stat_f_idr

- Omega_0_idm_dr (DM interacting with DR)

7.2.d)

7.2.e)

7.2.e.3/4)

Simply set 7.2.e.3/4)

7.2.e.3) n_index_idm_dr

7.2.e.4) idr_nature

7.2.f) Strength of self interactions

7.2.g) Read alpha_idm_dr or alpha_dark

8) Dark energy Omega_0_lambda (cosmological constant), Omega0_fld (dark energy fluid), Omega0_scf (scalar field)

8.a) If Omega fluid is different from 0

8.a.1) PPF approximation

8.a.2) Equation of state

8.a.2.2) Equation of state of the fluid in 'CLP' case

8.a.2.3) Equation of state of the fluid in 'EDE' case

8.b) If Omega scalar field (SCF) is different from 0

8.b.1) Additional SCF parameters

8.b.2) SCF initial conditions from attractor solution

8.b.3) SCF tuning parameter

8.b.4) Shooting parameter

4.12.2.16 input_read_parameters_injection()

```
int input_read_parameters_injection (
             struct file_content * pfc,
             struct precision * ppr,
             struct thermodynamics * pth,
             ErrorMsg errmsg )
```

Read the parameters of injection structure (These are all exotic processes of energy injection)

Parameters

pfc	Input: pointer to local structure
ppr	Input: pointer to precision structure
pth	Input: pointer to thermodynamics structure
errmsg	Input: Error message

Returns

the error status

Summary:

- Define local variables

1) DM annihilation

1.a) Annihilation efficiency

1.a.1) Model energy fraction absorbed by the gas as a function of redhsift

2) DM decay

2.a) Fraction

2.b) Decay width

3) PBH evaporation

3.a) Fraction

3.b) Mass

4) PBH matter accretion

4.a) Fraction

4.b) Mass

4.c) Recipe

4.c.1) Additional parameters specific for spherical accretion

4.c.2) Additional parameters specific for disk accretion

5) Injection efficiency

6) deposition function

6.a) External file

4.12.2.17 input_read_parameters_nonlinear()

```
int input_read_parameters_nonlinear (
             struct file_content * pfc,
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             struct fourier * pfo,
             int input_verbose,
             ErrorMsg errmsg )
```

Read the parameters of fourier structure.

Parameters

pfc	Input: pointer to local structure
ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer to perturbations structure
pfo	Input: pointer to fourier structure
input_verbose	Input: verbosity of input
errmsg	Input: Error message

Returns

the error status

Define local variables

1) Non-linearity

- special steps if we want Halofit with wa_fld non-zero: so-called "Pk_equal method" of 0810.0190 and 1601.↩ 07230

4.12.2.18 input_prepare_pk_eq()

```
int input_prepare_pk_eq (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct fourier * pfo,
             int input_verbose,
             ErrorMsg errmsg )
```

Perform preliminary steps fur using the method called Pk_equal, described in 0810.0190 and 1601.07230, extending the range of validity of HALOFIT from constant w to (w0,wa) models. In that case, one must compute here some effective values of w0_eff(z_i) and Omega_m_eff(z_i), that will be interpolated later at arbitrary redshift in the non-linear module.

Returns table of values [z_i, tau_i, w0_eff_i, Omega_m_eff_i] stored in fourier structure.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
pfo	Input/Output: pointer to fourier structure
input_verbose	Input: verbosity of this input module
errmsg	Input/Ouput: error message

Returns

the error status

Summary:

Define local variables

Store the true cosmological parameters (w0, wa) somwhere before using temporarily some fake ones in this function

The fake calls of the background and thermodynamics module will be done in non-verbose mode

Allocate indices and arrays for storing the results

Call the background module in order to fill a table of tau_i[z_i]

Loop over z_i values. For each of them, we will call the background and thermodynamics module for fake models. The goal is to find, for each z_i, and effective w0_eff[z_i] and Omega_m_eff{z_i], such that: the true model with (w0,wa) and the equivalent model with (w0_eff[z_i],0) have the same conformal distance between z_i and z_↩ recombination, namely chi = tau[z_i] - tau_rec. It is thus necessary to call both the background and thermodynamics module for each fake model and to re-compute tau_rec for each of them. Once the eqauivalent model is found we compute and store Omega_m_effa(z_i) of the equivalent model

Restore cosmological parameters (w0, wa) to their true values before main call to CLASS modules

Spline the table for later interpolation

4.12.2.19 input_read_parameters_primordial()

```
int input_read_parameters_primordial (
              struct file_content * pfc,
              struct perturbations * ppt,
              struct primordial * ppm,
              ErrorMsg errmsg )
```

Read the parameters of primordial structure.

Parameters

pfc	Input: pointer to local structure
ppt	Input: pointer to perturbations structure
ppm	Input: pointer to primordial structure
errmsg	Input: Error message

Returns

the error status

Summary:

Define local variables

1) Primordial spectrum type

1.a) Pivot scale in Mpc-1

1.b) For type 'analytic_Pk'

1.b.1) For scalar perturbations

1.b.1.1) Adiabatic perturbations

1.b.1.2) Isocurvature/entropy perturbations

1.b.1.3) Cross-correlation between different adiabatic/entropy mode

1.b.2) For tensor perturbations

1.c) For type 'inflation_V'

1.c.1) Type of potential

1.c.2) Coefficients of the Taylor expansion

1.d) For type 'inflation_H'

1.e) For type 'inflation_V_end'

1.e.1) Value of the field at the minimum of the potential

1.e.2) Shape of the potential

1.e.3) Parameters of the potential

1.e.4) How much the scale factor a or the product (aH) increases between Hubble crossing for the pivot scale (during inflation) and the end of inflation

1.e.5) Should the inflation module do its nomral job of numerical integration ('numerical') or use analytical slow-roll formulas to infer the primordial spectrum from the potential ('analytical')?

1.f) For type 'two_scales'

1.f.1) Wavenumbers

1.f.2) Amplitudes for the adiabatic primordial spectrum

1.f.3) Isocurvature amplitudes

1.f.4) Uncorrelated or anti-correlated?

1.g) For type 'external_Pk'

1.g.1) Command generating the table

1.g.2) Command generating the table

4.12.2.20 input_read_parameters_spectra()

```
int input_read_parameters_spectra (
             struct file_content * pfc,
             struct precision * ppr,
             struct background * pba,
             struct primordial * ppm,
             struct perturbations * ppt,
             struct transfer * ptr,
             struct harmonic * phr,
             struct output * pop,
             ErrorMsg errmsg )
```

Read the parameters of harmonic structure.

Parameters

pfc	Input: pointer to local structure
ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
ppm	Input: pointer to primordial structure
ppt	Input: pointer to perturbations structure
ptr	Input: pointer to transfer structure
phr	Input: pointer to harmonic structure
pop	Input: pointer to output structure
errmsg	Input: Error message

Returns

the error status

Summary:

Define local variables

1) Maximum l for CLs

2) Parameters for the the matter density number count

2.a) Selection functions W(z) of each redshift bin

2.b) Selection function

2.c) Source number counts evolution

3) Power spectrum P(k)

3.a) Maximum k in P(k)

3.a.1) Maximum k in primordial P(k)

3.b) Redshift values

3.c) Maximum redshift

4.12.2.21 input_read_parameters_lensing()

```
int input_read_parameters_lensing (
             struct file_content * pfc,
             struct precision * ppr,
             struct perturbations * ppt,
             struct transfer * ptr,
             struct lensing * ple,
             ErrorMsg errmsg )
```

Read the parameters of lensing structure.

Parameters

pfc	Input: pointer to local structure
ppr	Input: pointer to precision structure
ppt	Input: pointer to perturbations structure
ptr	Input: pointer to transfer structure
ple	Input: pointer to lensing structure
errmsg	Input: Error message

Returns

the error status

Summary:

Define local variables

1) Lensed spectra?

2) Should the lensed spectra be rescaled (either with just A_L, or otherwise with amplitude, and tilt and pivot scale in k space)

4.12.2.22 input_read_parameters_distortions()

```
int input_read_parameters_distortions (
             struct file_content * pfc,
             struct precision * ppr,
             struct distortions * psd,
             ErrorMsg errmsg )
```

Read free parameters of distortions structure.

Parameters

pfc	Input: pointer to local structure
ppr	Input: pointer to precision structure
psd	Input: pointer to distortions structure
errmsg	Input: Error message

Returns

the error status

Summary:

Define local variables

1) Branching ratio approximation

1.a.1) Number of multipoles in PCA expansion

1.a.2) Detector name

1.a.3) Detector specifics

1.a.3.1) From file

1.a.3.2) User defined

2) Only calculate exotic energy injections and no LCDM processes for spectral distortions ?

3) Include g distortions?

4) Set g-distortions to zero?

5) Include SZ effect from reionization?

5.a) Type of calculation

4.12.2.23 input_read_parameters_additional()

```
int input_read_parameters_additional (
             struct file_content * pfc,
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             ErrorMsg errmsg )
```

Read obsolete/additional parameters that are not assigned to a specific structure

Parameters

pfc	Input: pointer to local structure
ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
errmsg	Input: Error message

Returns

the error status

Summary:

Define local variables

Here we can place all obsolete (deprecated) names for the precision parameters that will still be read as of the current version. There is however, no guarantee that this will be true for future versions as well. The new parameter names should be used preferrably.

Here are slgihtly more obsolete parameters, these will not even be read, only give an error message

Test additional input parameters related to precision parameters

4.12.2.24 input_read_parameters_output()

```
int input_read_parameters_output (
              struct file_content * pfc,
              struct background * pba,
              struct thermodynamics * pth,
              struct perturbations * ppt,
              struct transfer * ptr,
              struct primordial * ppm,
              struct harmonic * phr,
              struct fourier * pfo,
              struct lensing * ple,
              struct distortions * psd,
              struct output * pop,
              ErrorMsg errmsg )
```

Read the parameters of output structure.

Parameters

pfc	Input: pointer to local structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer to perturbations structure
ptr	Input: pointer to transfer structure
ppm	Input: pointer to primordial structure
phr	Input: pointer to harmonic structure
pfo	Input: pointer to non-linear structure
ple	Input: pointer to lensing structure
psd	Input: pointer to distorsion structure
pop	Input: pointer to output structure
errmsg	Input: Error message

Returns

the error status

Summary:

Define local variables

1) Output for external files

1.a) File name

1.b) Headers

1.c) Format

1.d) Background quantities

1.e) Thermodynamics quantities

1.f) Table of perturbations for certain wavenumbers k

1.g) Primordial spectra

1.h) Exotic energy injection output

1.i) Non-injected photon injection

1.k) Spectral Distortions

2) Verbosity

4.12.2.25 input_write_info()

```
int input_write_info (
             struct file_content * pfc,
             struct output * pop,
             ErrorMsg errmsg )
```

Write the info related to the used and unused parameters Additionally, write the warnings for unused parameters

Parameters

pfc	Input: pointer to local structure
pop	Input: pointer to output structure
errmsg	Input: Error message

Returns

the error status

Summary:

Define local variables

4.12.2.26 input_default_params()

```
int input_default_params (
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             struct transfer * ptr,
             struct primordial * ppm,
             struct harmonic * phr,
             struct fourier * pfo,
             struct lensing * ple,
             struct distortions * psd,
             struct output * pop )
```

All default parameter values (for input parameters)

Parameters

pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
ppm	Input: pointer to primordial structure
phr	Input: pointer to harmonic structure
pfo	Input: pointer to fourier structure
ple	Input: pointer to lensing structure
psd	Input: pointer to distorsion structure
pop	Input: pointer to output structure

Returns

the error status

the error status

Summary:

- Define local variables

Default to input_read_parameters_general

1) Output spectra

1.a) 'tCl' case

1.a.1) Split value of redshift z at which the isw is considered as late or early

1.b) 'nCl' (or 'dCl') case

1.c) 'dTk' (or 'mTk') case

2) Perturbed recombination

3) Modes

3.a) Initial conditions for scalars

3.b) Initial conditions for tensors

4.a) Gauge

4.b) N-body gauge

5) Hubble parameter

6) Primordial Helium fraction

7) Recombination algorithm

8) Parametrization of reionization

8.a) 'reio_camb' or 'reio_half_tanh' case

8.b) 'reio_bins_tanh' case

8.c) 'reio_many_tanh' case

8.d) 'reio_inter' case

9) Damping scale

10) Varying fundamental constants

Default to input_read_parameters_species

1) Photon density

2) Baryon density

3) Ultra-relativistic species / massless neutrino density, assuming as default value N_eff=3.044 (see 2008.01074 and 2012.02726. This value is more accurate than the previous default value of 3.046)

3.a) Effective squared sound speed and viscosity parameter

4) CDM density

5) ncdm sector

5.a) Number of distinct species

5.b) List of names of psd files

5.c) Analytic distribution function

5.d) --> See read_parameters_background

5.e) ncdm temperature

5.f) ncdm chemical potential

5.g) ncdm degeneracy parameter

5.h) --> See read_parameters_background

6) Curvature density

7.1) Decaying CDM into Dark Radiation = dcdm+dr

7.1.a) Current fractional density of dcdm+dr

7.1.c) Decay constant

7.2) Interacting Dark Matter

7.2.a) Current fractional density of idm_dr+idr

7.2.b) Current temperature of idm_dr+idr

7.2.c) ETHOS parameters of idm_dr+idr

7.2.d) Approximation mode of idr

7.2.g, 7.2.h)

9) Dark energy contributions

8.a) Omega fluid

8.a.1) PPF approximation

9.a.2) Equation of state

9.a.2.1) 'CLP' case

9.a.2.2) 'EDE' case

9.b) Omega scalar field

9.b.1) Potential parameters and initial conditions

9.b.2) Initial conditions from attractor solution

9.b.3) Tuning parameter

9.b.4) Shooting parameter

Deafult to input_read_parameters_heating

1) DM annihilation

1.a) Energy fraction absorbed by the gas

1.a.1) Redshift dependence

2) DM decay

2.a) Fraction

2.b) Decay width

3) PBH evaporation

3.a) Fraction

3.b) Mass

4) PBH accretion

4.a) Fraction

4.b) Mass

4.c) Recipe

4.c.1) Additional parameters for spherical accretion

4.c.1) Additional parameters for disk accretion

5) Injection efficiency

6) Deposition function

6.1) External file

Default to input_read_parameters_nonlinear

1) Non-linearity

Default to input_read_parameters_primordial

1) Primordial spectrum type

1.a) Pivot scale in Mpc-1

1.b) For type 'analytic_Pk'

1.b.1) For scalar perturbations

1.b.1.1) Adiabatic perturbations

1.b.1.2) Isocurvature/entropy perturbations

1.b.1.3) Cross-correlation between different adiabatic/entropy mode

1.b.2) For tensor perturbations

1.c) For type 'inflation_V'

1.c.2) Coefficients of the Taylor expansion

1.d) For type 'inflation_H'

1.e) For type 'inflation_V_end'

1.e.1) Value of the field at the minimum of the potential

1.e.2) Shape of the potential

1.e.4) Increase of scale factor or (aH) between Hubble crossing at pivot scale and end of inflation

1.e.5) Nomral numerical integration or analytical slow-roll formulas?

1.g) For type 'external_Pk'

1.g.1) Command generating the table

1.g.2) Parameters to be passed to the command

Default to input_read_parameters_spectra

1) Maximum l for CLs

2) Parameters for the the matter density number count

2.a) Selection functions W(z) of each redshift bin

2.b) Selection function

2.c) Source number counts evolution

3) Power spectrum P(k)

3.a) Maximum k in P(k)

3.a) Maximum k in P(k) primordial

3.b) Redshift values

3.c) Maximum redshift

Default to input_read_parameters_lensing

1) Lensing

2) Should the lensed spectra be rescaled?

Default to input_read_parameters_distortions

1) Branching ratio approximation

1.a.1) Number of multipoles in PCA expansion

1.a.2) Detector noise file name

1.a.3) Detector name

1.3.a.1) Detector nu min

1.3.a.2) Detector nu max

1.3.a.3) Detector nu delta/bin number

1.3.a.1) Detector noise

2) Only exotic species?

3) Include g distortion in total calculation?

4) Additional y or mu parameters?

5) Include SZ effect from reionization?

5.a) What type of approximation you want to use for the SZ effect?

Default to input_read_additional

Default to input_read_parameters_output

1) Output for external files

1.a) File name

1.b) Headers

1.c) Format

1.d) Background quantities

1.e) Thermodynamics quantities

1.f) Table of perturbations for certain wavenumbers k

1.g) Primordial spectra

1.h) Exotic energy injection function

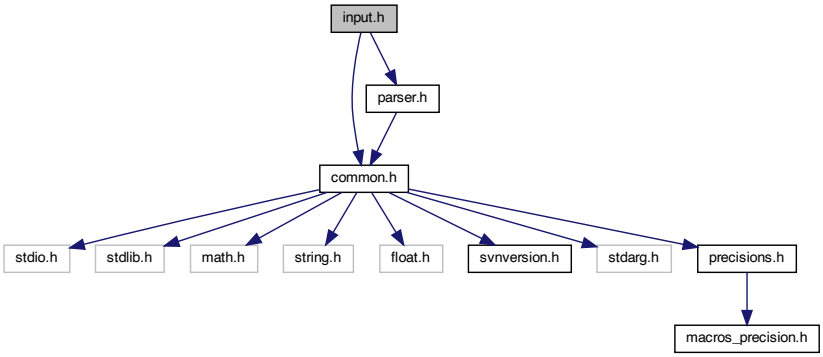
1.i) Spectral distortions

2) Verbosity

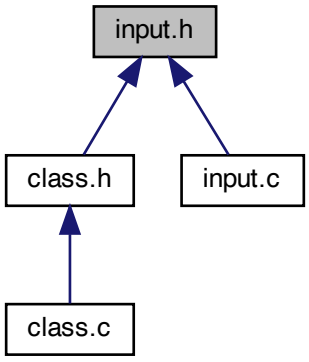
4.13 input.h File Reference

```
#include "common.h"
#include "parser.h"
```

Include dependency graph for input.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct fzerofun_workspace

Enumerations

- enum target_names

4.13.1 Detailed Description

Documented includes for input module

4.13.2 Data Structure Documentation

4.13.2.1 struct fzerofun_workspace

Structure for all temporary parameters for background fzero function

4.13.3 Enumeration Type Documentation

4.13.3.1 target_names

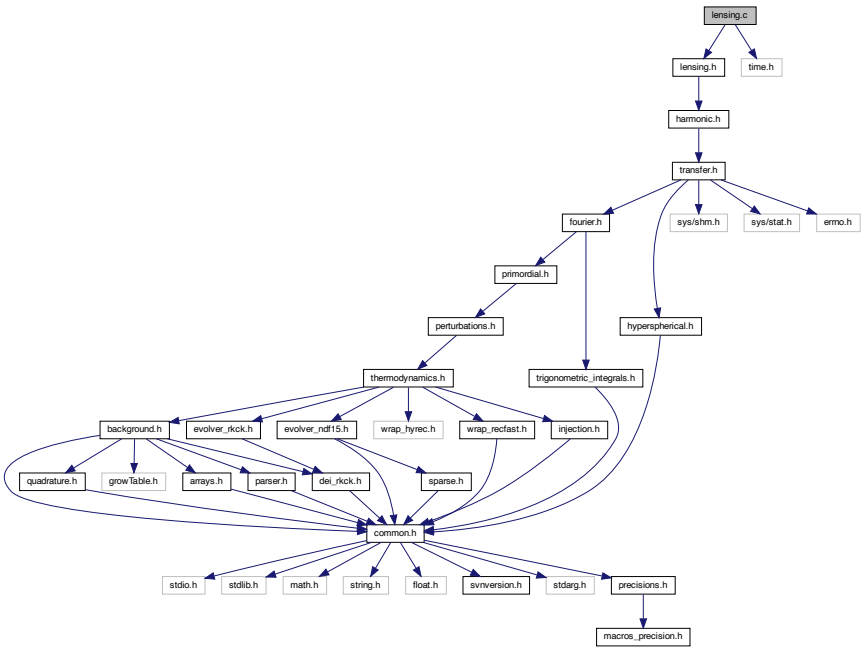
```
enum target_names
```

For shooting method: definition of the possible targets

4.14 lensing.c File Reference

```
#include "lensing.h"
#include <time.h>
```

Include dependency graph for lensing.c:



Functions

- int lensing_cl_at_l (struct lensing *ple, int l, double *cl_lensed)
- int lensing_init (struct precision *ppr, struct perturbations *ppt, struct harmonic *phr, struct fourier *pfo, struct lensing *ple)
- int lensing_free (struct lensing *ple)
- int lensing_indices (struct precision *ppr, struct harmonic *phr, struct lensing *ple)
- int lensing_lensed_cl_tt (double *ksi, double **d00, double *w8, int nmu, struct lensing *ple)
- int lensing_addback_cl_tt (struct lensing *ple, double *cl_tt)
- int lensing_lensed_cl_te (double *ksiX, double **d20, double *w8, int nmu, struct lensing *ple)
- int lensing_addback_cl_te (struct lensing *ple, double *cl_te)
- int lensing_lensed_cl_ee_bb (double *ksip, double *ksim, double **d22, double **d2m2, double *w8, int nmu, struct lensing *ple)
- int lensing_addback_cl_ee_bb (struct lensing *ple, double *cl_ee, double *cl_bb)
- int lensing_d00 (double *mu, int num_mu, int lmax, double **d00)
- int lensing_d11 (double *mu, int num_mu, int lmax, double **d11)
- int lensing_d1m1 (double *mu, int num_mu, int lmax, double **d1m1)
- int lensing_d2m2 (double *mu, int num_mu, int lmax, double **d2m2)
- int lensing_d22 (double *mu, int num_mu, int lmax, double **d22)
- int lensing_d20 (double *mu, int num_mu, int lmax, double **d20)
- int lensing_d31 (double *mu, int num_mu, int lmax, double **d31)
- int lensing_d3m1 (double *mu, int num_mu, int lmax, double **d3m1)
- int lensing_d3m3 (double *mu, int num_mu, int lmax, double **d3m3)
- int lensing_d40 (double *mu, int num_mu, int lmax, double **d40)
- int lensing_d4m2 (double *mu, int num_mu, int lmax, double **d4m2)
- int lensing_d4m4 (double *mu, int num_mu, int lmax, double **d4m4)

4.14.1 Detailed Description

Documented lensing module

Simon Prunet and Julien Lesgourgues, 6.12.2010

This module computes the lensed temperature and polarization anisotropy power spectra $C_l^X, P(k), ...$'s given the unlensed temperature, polarization and lensing potential spectra.

Follows Challinor and Lewis full-sky method, astro-ph/0502425

The following functions can be called from other modules:

1. lensing_init() at the beginning (but after harmonic_init())
2. lensing_cl_at_l() at any time for computing Cl_lensed at any l
3. lensing_free() at the end

4.14.2 Function Documentation

4.14.2.1 lensing_cl_at_l()

```
int lensing_cl_at_l (
             struct lensing * ple,
             int l,
             double * cl_lensed )
```

Anisotropy power spectra C_l's for all types, modes and initial conditions. SO FAR: ONLY SCALAR

This routine evaluates all the lensed C_l's at a given value of l by picking it in the pre-computed table. When relevant, it also sums over all initial conditions for each mode, and over all modes.

This function can be called from whatever module at whatever time, provided that lensing_init() has been called before, and lensing_free() has not been called yet.

Parameters

ple	Input: pointer to lensing structure
l	Input: multipole number
cl_lensed	Output: lensed C_l's for all types (TT, TE, EE, etc..)

Returns

the error status

4.14.2.2 lensing_init()

```
int lensing_init (
             struct precision * ppr,
             struct perturbations * ppt,
             struct harmonic * phr,
             struct fourier * pfo,
             struct lensing * ple )
```

This routine initializes the lensing structure (in particular, computes table of lensed anisotropy spectra C_l^X)

Parameters

ppr	Input: pointer to precision structure
ppt	Input: pointer to perturbation structure (just in case, not used in current version...)
phr	Input: pointer to harmonic structure
pfo	Input: pointer to fourier structure
ple	Output: pointer to initialized lensing structure

Returns

the error status

Summary:

- Define local variables
- check that we really want to compute at least one spectrum
- initialize indices and allocate some of the arrays in the lensing structure
- put all precision variables hare; will be stored later in precision structure
- Last element in μ will be for $\mu = 1$, needed for sigma2. The rest will be chosen as roots of a Gauss-Legendre quadrature
- allocate array of μ values, as well as quadrature weights
- Compute $d^l_{mm'}(\mu)$
- Allocate main contiguous buffer
- compute $Cgl(\mu)$, $Cgl2(\mu)$ and sigma2(μ)
- Locally store unlensed temperature cl_{tt} and potential cl_{pp} spectra
- Compute sigma2 (μ) and Cgl2(μ)
- compute ksi, ksi+, ksi-, ksiX
- --> ksi is for TT
- --> ksiX is for TE
- --> ksip, ksim for EE, BB
- compute lensed C_l's by integration
- spline computed C_l's in view of interpolation
- Free lots of stuff
- Exit

4.14.2.3 lensing_free()

```
int lensing_free (
             struct lensing * ple )
```

This routine frees all the memory space allocated by lensing_init().

To be called at the end of each run, only when no further calls to lensing_cl_at_l() are needed.

Parameters

ple	Input: pointer to lensing structure (which fields must be freed)

Returns

the error status

4.14.2.4 lensing_indices()

```
int lensing_indices (
             struct precision * ppr,
             struct harmonic * phr,
             struct lensing * ple )
```

This routine defines indices and allocates tables in the lensing structure

Parameters

ppr	Input: pointer to precision structure
phr	Input: pointer to harmonic structure
ple	Input/output: pointer to lensing structure

Returns

the error status

4.14.2.5 lensing_lensed_cl_tt()

```
int lensing_lensed_cl_tt (
             double * ksi,
             double ** d00,
             double * w8,
             int nmu,
             struct lensing * ple )
```

This routine computes the lensed power spectra by Gaussian quadrature

Parameters

ksi	Input: Lensed correlation function (ksi[index_mu])
d00	Input: Legendre polynomials (d^l_{00}[l][index_mu])
w8	Input: Legendre quadrature weights (w8[index_mu])
nmu	Input: Number of quadrature points (0<=index_mu<=nmu)
ple	Input/output: Pointer to the lensing structure

Returns

the error status

Integration by Gauss-Legendre quadrature.

4.14.2.6 lensing_addback_cl_tt()

```
int lensing_addback_cl_tt (
             struct lensing * ple,
             double * cl_tt )
```

This routine adds back the unlensed cl_{tt} power spectrum Used in case of fast (and BB inaccurate) integration of correlation functions.

Parameters

ple	Input/output: Pointer to the lensing structure
cl↩ _tt	Input: Array of unlensed power spectrum

Returns

the error status

4.14.2.7 lensing_lensed_cl_te()

```
int lensing_lensed_cl_te (
              double * ksiX,
              double ** d20,
              double * w8,
              int nmu,
              struct lensing * ple )
```

This routine computes the lensed power spectra by Gaussian quadrature

Parameters

ksiX	Input: Lensed correlation function (ksiX[index_mu])
d20	Input: Wigner d-function (d^l_{20}[l][index_mu])
w8	Input: Legendre quadrature weights (w8[index_mu])
nmu	Input: Number of quadrature points (0<=index_mu<=nmu)
ple	Input/output: Pointer to the lensing structure

Returns

the error status

Integration by Gauss-Legendre quadrature.

4.14.2.8 lensing_addback_cl_te()

```
int lensing_addback_cl_te (
              struct lensing * ple,
              double * cl_te )
```

This routine adds back the unlensed cl_{te} power spectrum Used in case of fast (and BB inaccurate) integration of correlation functions.

Parameters

ple	Input/output: Pointer to the lensing structure
cl↩ _te	Input: Array of unlensed power spectrum

Returns

the error status

4.14.2.9 lensing_lensed_cl_ee_bb()

```
int lensing_lensed_cl_ee_bb (
              double * ksip,
              double * ksim,
              double ** d22,
              double ** d2m2,
              double * w8,
              int nmu,
              struct lensing * ple )
```

This routine computes the lensed power spectra by Gaussian quadrature

Parameters

ksip	Input: Lensed correlation function (ksi+[index_mu])
ksim	Input: Lensed correlation function (ksi-[index_mu])
d22	Input: Wigner d-function (d^l_{22}[l][index_mu])
d2m2	Input: Wigner d-function (d^l_{2-2}[l][index_mu])
w8	Input: Legendre quadrature weights (w8[index_mu])
nmu	Input: Number of quadrature points (0<=index_mu<=nmu)
ple	Input/output: Pointer to the lensing structure

Returns

the error status

Integration by Gauss-Legendre quadrature.

4.14.2.10 lensing_addback_cl_ee_bb()

```
int lensing_addback_cl_ee_bb (
              struct lensing * ple,
              double * cl_ee,
              double * cl_bb )
```

This routine adds back the unlensed cl_{ee}, cl_{bb} power spectra Used in case of fast (and BB inaccurate) integration of correlation functions.

Parameters

ple	Input/output: Pointer to the lensing structure
cl_ee	Input: Array of unlensed power spectrum
cl_bb	Input: Array of unlensed power spectrum

Returns

the error status

4.14.2.11 lensing_d00()

```
int lensing_d00 (
            double * mu,
            int num_mu,
            int lmax,
            double ** d00 )
```

This routine computes the d00 term

Parameters

mu	Input: Vector of cos(beta) values
num_mu	Input: Number of cos(beta) values
lmax	Input: maximum multipole
d00	Input/output: Result is stored here

Wigner d-functions, computed by recurrence actual recurrence on $\sqrt{(2l+1)/2}d^l_{mm}$, for stability Formulae from Kostelec & Rockmore 2003

4.14.2.12 lensing_d11()

```
int lensing_d11 (
            double * mu,
            int num_mu,
            int lmax,
            double ** d11 )
```

This routine computes the d11 term

Parameters

mu	Input: Vector of cos(beta) values
num_mu	Input: Number of cos(beta) values
lmax	Input: maximum multipole
d11	Input/output: Result is stored here

Wigner d-functions, computed by recurrence actual recurrence on $\sqrt{(2l+1)/2}d^l_{mm}$, for stability Formulae from Kostelec & Rockmore 2003

4.14.2.13 lensing_d1m1()

```
int lensing_d1m1 (
            double * mu,
            int num_mu,
            int lmax,
            double ** d1m1 )
```

This routine computes the d1m1 term

Parameters

mu	Input: Vector of cos(beta) values
num_mu	Input: Number of cos(beta) values
lmax	Input: maximum multipole
d1m1	Input/output: Result is stored here

Wigner d-functions, computed by recurrence actual recurrence on $\sqrt{(2l+1)/2}d^l_{mm}$, for stability Formulae from Kostelec & Rockmore 2003

4.14.2.14 lensing_d2m2()

```
int lensing_d2m2 (
            double * mu,
            int num_mu,
            int lmax,
            double ** d2m2 )
```

This routine computes the d2m2 term

Parameters

mu	Input: Vector of cos(beta) values
num_mu	Input: Number of cos(beta) values
lmax	Input: maximum multipole
d2m2	Input/output: Result is stored here

Wigner d-functions, computed by recurrence actual recurrence on $\sqrt{(2l+1)/2}d^l_{mm}$, for stability Formulae from Kostelec & Rockmore 2003

4.14.2.15 lensing_d22()

```
int lensing_d22 (
            double * mu,
            int num_mu,
```

```
            int lmax,
            double ** d22 )
```

This routine computes the d22 term

Parameters

mu	Input: Vector of cos(beta) values
num_mu	Input: Number of cos(beta) values
lmax	Input: maximum multipole
d22	Input/output: Result is stored here

Wigner d-functions, computed by recurrence actual recurrence on $\sqrt{(2l+1)/2}d^l_{mm}$, for stability Formulae from Kostelec & Rockmore 2003

4.14.2.16 lensing_d20()

```
int lensing_d20 (
            double * mu,
            int num_mu,
            int lmax,
            double ** d20 )
```

This routine computes the d20 term

Parameters

mu	Input: Vector of cos(beta) values
num_mu	Input: Number of cos(beta) values
lmax	Input: maximum multipole
d20	Input/output: Result is stored here

Wigner d-functions, computed by recurrence actual recurrence on $\sqrt{(2l+1)/2}d^l_{mm}$, for stability Formulae from Kostelec & Rockmore 2003

4.14.2.17 lensing_d31()

```
int lensing_d31 (
            double * mu,
            int num_mu,
            int lmax,
            double ** d31 )
```

This routine computes the d31 term

Parameters

mu	Input: Vector of cos(beta) values
num_mu	Input: Number of cos(beta) values
lmax	Input: maximum multipole
d31	Input/output: Result is stored here

Wigner d-functions, computed by recurrence actual recurrence on $\sqrt{(2l+1)/2}d^l_{mm}$, for stability Formulae from Kostelec & Rockmore 2003

4.14.2.18 lensing_d3m1()

```
int lensing_d3m1 (
            double * mu,
            int num_mu,
            int lmax,
            double ** d3m1 )
```

This routine computes the d3m1 term

Parameters

mu	Input: Vector of cos(beta) values
num_mu	Input: Number of cos(beta) values
lmax	Input: maximum multipole
d3m1	Input/output: Result is stored here

Wigner d-functions, computed by recurrence actual recurrence on $\sqrt{(2l+1)/2}d^l_{mm}$, for stability Formulae from Kostelec & Rockmore 2003

4.14.2.19 lensing_d3m3()

```
int lensing_d3m3 (
            double * mu,
            int num_mu,
            int lmax,
            double ** d3m3 )
```

This routine computes the d3m3 term

Parameters

mu	Input: Vector of cos(beta) values
num_mu	Input: Number of cos(beta) values
lmax	Input: maximum multipole
d3m3	Input/output: Result is stored here

Wigner d-functions, computed by recurrence actual recurrence on $\sqrt{(2l+1)/2}d^l_{mm}$, for stability Formulae from Kostelec & Rockmore 2003

4.14.2.20 lensing_d40()

```
int lensing_d40 (
            double * mu,
            int num_mu,
```

```
            int lmax,
            double ** d40 )
```

This routine computes the d40 term

Parameters

mu	Input: Vector of cos(beta) values
num_mu	Input: Number of cos(beta) values
lmax	Input: maximum multipole
d40	Input/output: Result is stored here

Wigner d-functions, computed by recurrence actual recurrence on $\sqrt{(2l+1)/2}d^l_{mm}$, for stability Formulae from Kostelec & Rockmore 2003

4.14.2.21 lensing_d4m2()

```
int lensing_d4m2 (
            double * mu,
            int num_mu,
            int lmax,
            double ** d4m2 )
```

This routine computes the d4m2 term

Parameters

mu	Input: Vector of cos(beta) values
num_mu	Input: Number of cos(beta) values
lmax	Input: maximum multipole
d4m2	Input/output: Result is stored here

Wigner d-functions, computed by recurrence actual recurrence on $\sqrt{(2l+1)/2}d^l_{mm}$, for stability Formulae from Kostelec & Rockmore 2003

4.14.2.22 lensing_d4m4()

```
int lensing_d4m4 (
            double * mu,
            int num_mu,
            int lmax,
            double ** d4m4 )
```

This routine computes the d4m4 term

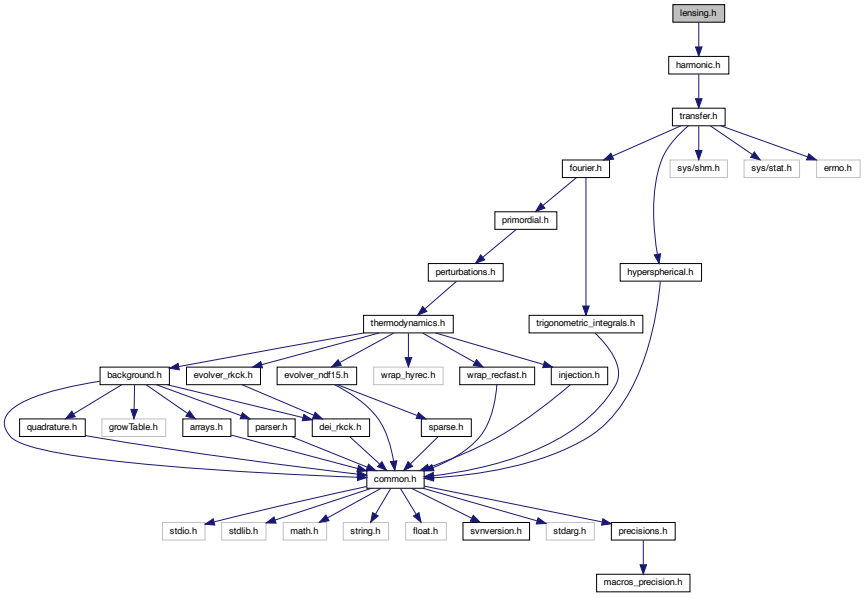
Parameters

mu	Input: Vector of cos(beta) values
num_mu	Input: Number of cos(beta) values
lmax	Input: maximum multipole
d4m4	Input/output: Result is stored here

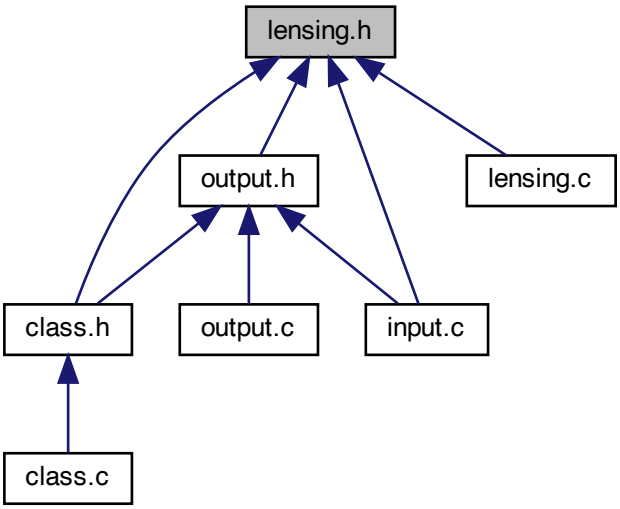
Wigner d-functions, computed by recurrence actual recurrence on $\sqrt{(2l+1)/2}d^l_{mm}$, for stability Formulae from Kostelec & Rockmore 2003

4.15 lensing.h File Reference

`#include "harmonic.h"`
Include dependency graph for lensing.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct lensing

4.15.1 Detailed Description

Documented includes for harmonic module

4.15.2 Data Structure Documentation

4.15.2.1 struct lensing

Structure containing everything about lensed spectra that other modules need to know.

Once initialized by lensing_init(), contains a table of all lensed C_l's for the all modes (scalar/tensor), all types (TT, TE...), and all pairs of initial conditions (adiabatic, isocurvatures...). FOR THE MOMENT, ASSUME ONLY SCALAR & ADIABATIC

Data Fields

short	has_lensed_cls	do we need to compute lensed C_l's at all ?
int	has_tt	do we want lensed C_l^{TT}? (T = temperature)
int	has_ee	do we want lensed C_l^{EE}? (E = E-polarization)
int	has_te	do we want lensed C_l^{TE}?
int	has_bb	do we want C_l^{BB}? (B = B-polarization)
int	has_pp	do we want $C_l^{\phi\phi}$? (ϕ = CMB lensing potential)
int	has_tp	do we want $C_l^{T\phi}$?
int	has_dd	do we want C_l^{dd}? (d = matter density)
int	has_td	do we want C_l^{Td}?
int	has_ll	do we want C_l^{ll}? (l = lensing potential)
int	has_tl	do we want C_l^{Tl}?
int	index_lt_tt	index for type C_l^{TT}
int	index_lt_ee	index for type C_l^{EE}
int	index_lt_te	index for type C_l^{TE}
int	index_lt_bb	index for type C_l^{BB}
int	index_lt_pp	index for type $C_l^{\phi\phi}$
int	index_lt_tp	index for type $C_l^{T\phi}$
int	index_lt_dd	index for type C_l^{dd}
int	index_lt_td	index for type C_l^{Td}
int	index_lt_ll	index for type C_l^{dd}
int	index_lt_tl	index for type C_l^{Td}
int	lt_size	number of C_l types requested
int	l_unlensed_max	last multipole in all calculations (same as in harmonic module)
int	l_lensed_max	last multipole at which lensed spectra are computed

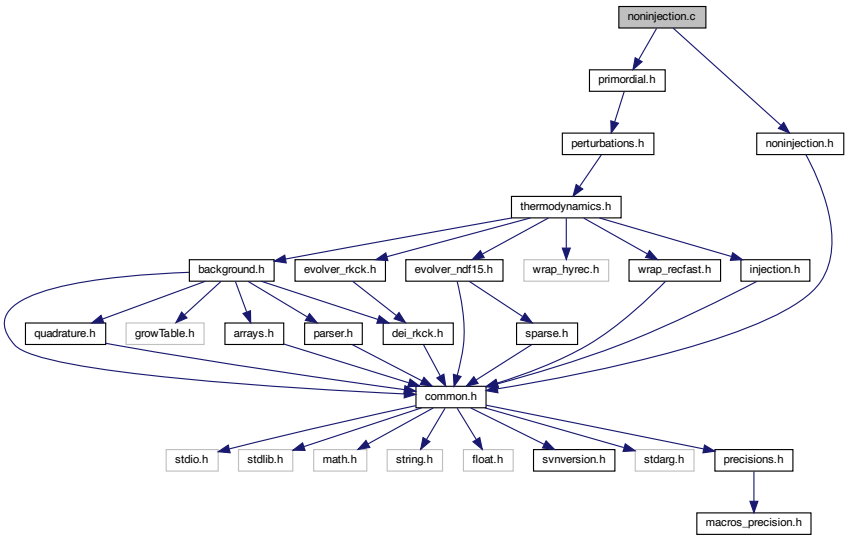
Data Fields

int	l_size	number of l values
int *	l_max_lt	last multipole (given as an input) at which we want to output C_l's for a given mode and type
double *	l	table of multipole values l[index_l]
double *	cl_lens	table of anisotropy spectra for each multipole and types, cl[index_l * ple->lt_size + index_lt]
double *	ddcl_lens	second derivatives for interpolation
short	lensing_verbose	flag regulating the amount of information sent to standard output (none if set to zero)
ErrorMsg	error_message	zone for writing error messages

4.16 noninjection.c File Reference

```
#include "primordial.h"
#include "noninjection.h"
```

Include dependency graph for noninjection.c:



4.16.1 Detailed Description

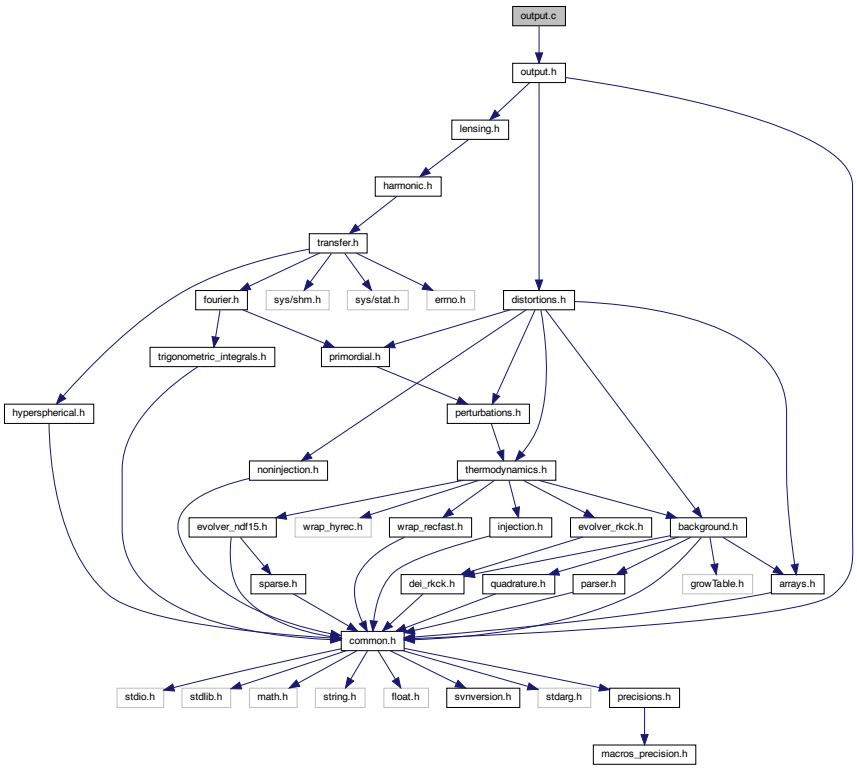
Documented non-exotic energy injection module

written by Nils Schoeneberg and Matteo Lucca, 27.02.2019

The main goal of this module is to calculate the non-injected energy for the photon evolution equation For more details see the description in the README file.

4.17 output.c File Reference

`#include "output.h"`
Include dependency graph for output.c:



Functions

- int output_init (struct background ∗pba, struct thermodynamics ∗pth, struct perturbations ∗ppt, struct primordial ∗ppm, struct transfer ∗ptr, struct harmonic ∗phr, struct fourier ∗pfo, struct lensing ∗ple, struct distortions ∗psd, struct output ∗pop)
- int output_cl (struct background ∗pba, struct perturbations ∗ppt, struct harmonic ∗phr, struct lensing ∗ple, struct output ∗pop)
- int output_pk (struct background ∗pba, struct perturbations ∗ppt, struct fourier ∗pfo, struct output ∗pop, enum pk_outputs pk_output)
- int output_tk (struct background ∗pba, struct perturbations ∗ppt, struct output ∗pop)
- int output_heating (struct injection ∗pin, struct noninjection ∗pni, struct output ∗pop)
- int output_distortions (struct distortions ∗psd, struct output ∗pop)
- int output_print_data (FILE ∗out, char titles[_MAXTITLESTRINGLENGTH_], double ∗dataptr, int size_↩ dataptr)
- int output_open_cl_file (struct harmonic ∗phr, struct output ∗pop, FILE ∗∗clfile, FileName filename, char ∗first_line, int lmax)
- int output_one_line_of_cl (struct background ∗pba, struct harmonic ∗phr, struct output ∗pop, FILE ∗clfile, double l, double ∗cl, int ct_size)
- int output_open_pk_file (struct background ∗pba, struct fourier ∗pfo, struct output ∗pop, FILE ∗∗pkfile, File↩ Name filename, char ∗first_line, double z)
- int output_one_line_of_pk (FILE ∗pkfile, double one_k, double one_pk)

4.17.1 Detailed Description

Documented output module

Julien Lesgourgues, 26.08.2010

This module writes the output in files.

The following functions can be called from other modules or from the main:

1. output_init() (must be called after harmonic_init())
2. output_total_cl_at_l() (can be called even before output_init())

No memory needs to be deallocated after that, hence there is no output_free() routine like in other modules.

4.17.2 Function Documentation

4.17.2.1 output_init()

```
int output_init (
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             struct primordial * ppm,
             struct transfer * ptr,
             struct harmonic * phr,
             struct fourier * pfo,
             struct lensing * ple,
             struct distortions * psd,
             struct output * pop )
```

This routine writes the output in files.

Parameters

pba	Input: pointer to background structure (needed for calling harmonic_pk_at_z())
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer perturbation structure
ppm	Input: pointer to primordial structure
ptr	Input: pointer to transfer structure
phr	Input: pointer to harmonic structure
pfo	Input: pointer to fourier structure
ple	Input: pointer to lensing structure
psd	Input: pointer to distortions structure
pop	Input: pointer to output structure

Summary:

- check that we really want to output at least one file
- deal with all anisotropy power spectra C_l's
- deal with all Fourier matter power spectra P(k)'s
- deal with density and matter power spectra
- deal with background quantities
- deal with thermodynamics quantities
- deal with perturbation quantities
- deal with primordial spectra
- deal with heating
- deal with spectral distortions

4.17.2.2 output_cl()

```
int output_cl (
               struct background * pba,
               struct perturbations * ppt,
               struct harmonic * phr,
               struct lensing * ple,
               struct output * pop )
```

This routines writes the output in files for anisotropy power spectra C_l's.

Parameters

pba	Input: pointer to background structure (needed for T_{cmb})
ppt	Input: pointer perturbation structure
phr	Input: pointer to harmonic structure
ple	Input: pointer to lensing structure
pop	Input: pointer to output structure

Summary:

- define local variables
- first, allocate all arrays of files and C_l's
- second, open only the relevant files, and write a heading in each of them
- third, perform loop over l. For each multipole, get all C_l's by calling harmonic_cl_at_l() and distribute the results to relevant files
- finally, close files and free arrays of files and C_l's

4.17.2.3 output_pk()

```
int output_pk (
               struct background * pba,
               struct perturbations * ppt,
               struct fourier * pfo,
               struct output * pop,
               enum pk_outputs pk_output )
```

This routines writes the output in files for Fourier matter power spectra P(k)'s (linear or non-linear)

Parameters

pba	Input: pointer to background structure (needed for calling harmonic_pk_at_z())
ppt	Input: pointer perturbation structure
pfo	Input: pointer to fourier structure
pop	Input: pointer to output structure
pk_output	Input: pk_linear or pk_nonlinear

Summary:

- define local variables
- preliminary: check whether we need to output the decomposition into contributions from each initial condition
- allocate arrays to store the P(k)
- allocate pointer to output files
- loop over pk type (_cb, _m)
- loop over z
- first, check that requested redshift z_pk is consistent
- second, open only the relevant files and write a header in each of them
- third, compute P(k) for each k
- fourth, write in files
- fifth, close files

4.17.2.4 output_tk()

```
int output_tk (
               struct background * pba,
               struct perturbations * ppt,
               struct output * pop )
```

This routines writes the output in files for matter transfer functions $T_i(k)$'s.

Parameters

pba	Input: pointer to background structure (needed for calling harmonic_pk_at_z())
ppt	Input: pointer perturbation structure
pop	Input: pointer to output structure

Summary:

- define local variables
- first, check that requested redshift z_pk is consistent
- second, open only the relevant files, and write a heading in each of them
- free memory and close files

4.17.2.5 output_heating()

```
int output_heating (
             struct injection * pin,
             struct noninjection * pni,
             struct output * pop )
```

Local variables

4.17.2.6 output_distortions()

```
int output_distortions (
             struct distortions * psd,
             struct output * pop )
```

Local variables

4.17.2.7 output_print_data()

```
int output_print_data (
             FILE * out,
             char titles[_MAXTITLESTRINGLENGTH_],
             double * dataptr,
             int size_dataptr )
```

Summary

- First we print the titles
- Then we print the data

4.17.2.8 output_open_cl_file()

```
int output_open_cl_file (
              struct harmonic * phr,
              struct output * pop,
              FILE ** clfile,
              FileName filename,
              char * first_line,
              int lmax )
```

This routine opens one file where some C_l's will be written, and writes a heading with some general information concerning its content.

Parameters

phr	Input: pointer to harmonic structure
pop	Input: pointer to output structure
clfile	Output: returned pointer to file pointer
filename	Input: name of the file
first_line	Input: text describing the content (mode, initial condition..)
lmax	Input: last multipole in the file (the first one is assumed to be 2)

Returns

the error status

Summary

- First we deal with the entries that are dependent of format type
- Next deal with entries that are independent of format type

4.17.2.9 output_one_line_of_cl()

```
int output_one_line_of_cl (
              struct background * pba,
              struct harmonic * phr,
              struct output * pop,
              FILE * clfile,
              double l,
              double * cl,
              int ct_size )
```

This routine write one line with l and all C_l's for all types (TT, TE...)

Parameters

pba	Input: pointer to background structure (needed for T_{cmb})
phr	Input: pointer to harmonic structure
pop	Input: pointer to output structure
clfile	Input: file pointer
l	Input: multipole
cl	Input: C_l's for all types
ct_size	Input: number of types

Returns

the error status

4.17.2.10 output_open_pk_file()

```
int output_open_pk_file (
             struct background * pba,
             struct fourier * pfo,
             struct output * pop,
             FILE ** pkfile,
             FileName filename,
             char * first_line,
             double z )
```

This routine opens one file where some P(k)'s will be written, and writes a heading with some general information concerning its content.

Parameters

pba	Input: pointer to background structure (needed for h)
pfo	Input: pointer to fourier structure
pop	Input: pointer to output structure
pkfile	Output: returned pointer to file pointer
filename	Input: name of the file
first_line	Input: text describing the content (initial conditions, ...)
z	Input: redshift of the output

Returns

the error status

4.17.2.11 output_one_line_of_pk()

```
int output_one_line_of_pk (
             FILE * pkfile,
             double one_k,
             double one_pk )
```

This routine writes one line with k and P(k)

Parameters

pkfile	Input: file pointer
one_k	Input: wavenumber
one_pk	Input: matter power spectrum

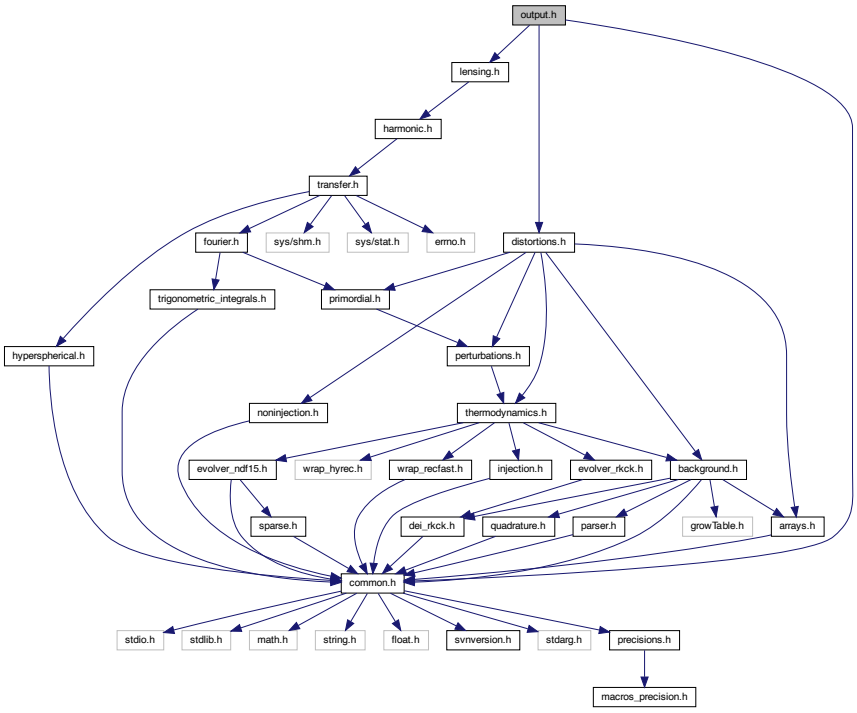
Returns

the error status

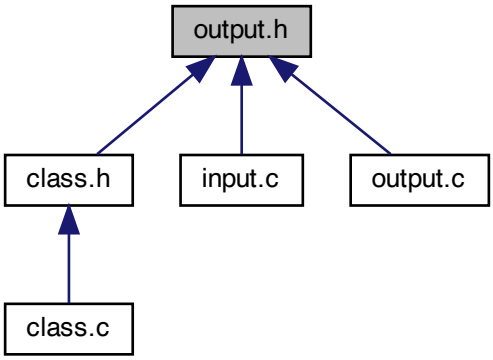
4.18 output.h File Reference

```
#include "common.h"
#include "lensing.h"
#include "distortions.h"
```

Include dependency graph for output.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct output

Macros

- #define _Z_PK_NUM_MAX_ 100

4.18.1 Detailed Description

Documented includes for output module

4.18.2 Data Structure Documentation

4.18.2.1 struct output

Structure containing various informations on the output format, all of them initialized by user in input module.

Data Fields

char	root[_FILENAMESIZE_-32]	root for all file names
int	z_pk_num	number of redshift at which P(k,z) and T_i(k,z) should be written
double	z_pk[_Z_PK_NUM_MAX_]	value(s) of redshift at which P(k,z) and T_i(k,z) should be written
short	write_header	flag stating whether we should write a header in output files
enum file_format	output_format	which format for output files (definitions, order of columns, etc.)
short	write_background	flag for outputing background evolution in file

Data Fields

short	write_thermodynamics	flag for outputing thermodynamical evolution in file
short	write_perturbations	flag for outputing perturbations of selected wavenumber(s) in file(s)
short	write_primordial	flag for outputing scalar/tensor primordial spectra in files
short	write_exotic_injection	flag for outputing exotic energy injection/deposition in files
short	write_noninjection	flag for outputing non-injected contributions in files
short	write_distortions	flag for outputing spectral distortions in files
short	output_verbose	flag regulating the amount of information sent to standard output (none if set to zero)
ErrorMsg	error_message	zone for writing error messages

4.18.3 Macro Definition Documentation

4.18.3.1 _Z_PK_NUM_MAX_

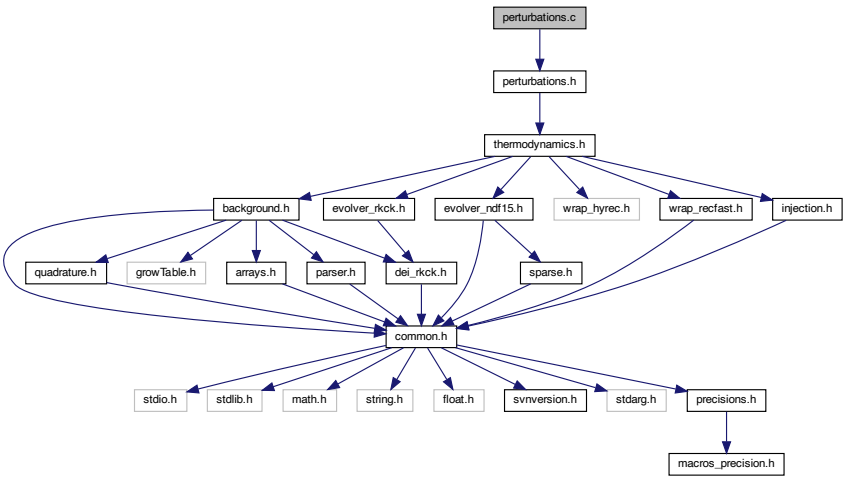
```
#define _Z_PK_NUM_MAX_ 100
```

Maximum number of values of redshift at which the spectra will be written in output files

4.19 perturbations.c File Reference

```
#include "perturbations.h"
```

Include dependency graph for perturbations.c:



Functions

- int perturbations_sources_at_tau (struct perturbations *ppt, int index_md, int index_ic, int index_tp, double tau, double *psource)
- int perturbations_output_data (struct background *pba, struct perturbations *ppt, enum file_format output_format, double z, int number_of_titles, double *data)
- int perturbations_output_titles (struct background *pba, struct perturbations *ppt, enum file_format output_format, char titles[_MAXTITLESTRINGLENGTH_])
- int perturbations_output_firstline_and_ic_suffix (struct perturbations *ppt, int index_ic, char first_line[_LINE_LENGTH_MAX_], char ic_suffix[_SUFFIXNAMESIZE_])
- int perturbations_init (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt)
- int perturbations_free_input (struct perturbations *ppt)
- int perturbations_free (struct perturbations *ppt)
- int perturbations_indices (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt)
- int perturbations_timesampling_for_sources (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt)
- int perturbations_get_k_list (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt)
- int perturbations_workspace_init (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, int index_md, struct perturbations_workspace *ppw)
- int perturbations_workspace_free (struct perturbations *ppt, int index_md, struct perturbations_workspace *ppw)
- int perturbations_solve (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, int index_md, int index_ic, int index_k, struct perturbations_workspace *ppw)
- int perturbations_prepare_k_output (struct background *pba, struct perturbations *ppt)
- int perturbations_find_approximation_number (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, int index_md, double k, struct perturbations_workspace *ppw, double tau_ini, double tau_end, int *interval_number, int *interval_number_of)
- int perturbations_find_approximation_switches (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, int index_md, double k, struct perturbations_workspace *ppw, double tau_ini, double tau_end, double precision, int interval_number, int *interval_number_of, double *interval_limit, int **interval_approx)
- int perturbations_vector_init (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, int index_md, int index_ic, double k, double tau, struct perturbations_workspace *ppw, int *pa_old)
- int perturbations_vector_free (struct perturbations_vector *pv)
- int perturbations_initial_conditions (struct precision *ppr, struct background *pba, struct perturbations *ppt, int index_md, int index_ic, double k, double tau, struct perturbations_workspace *ppw)
- int perturbations_approximations (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, int index_md, double k, double tau, struct perturbations_workspace *ppw)
- int perturbations_timescale (double tau, void *parameters_and_workspace, double *timescale, ErrorMsg error_message)
- int perturbations_einstein (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, int index_md, double k, double tau, double *y, struct perturbations_workspace *ppw)
- int perturbations_total_stress_energy (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, int index_md, double k, double *y, struct perturbations_workspace *ppw)
- int perturbations_sources (double tau, double *y, double *dy, int index_tau, void *parameters_and_workspace, ErrorMsg error_message)
- int perturbations_print_variables (double tau, double *y, double *dy, void *parameters_and_workspace, ErrorMsg error_message)
- int perturbations_derivs (double tau, double *y, double *dy, void *parameters_and_workspace, ErrorMsg error_message)
- int perturbations_tca_slip_and_shear (double *y, void *parameters_and_workspace, ErrorMsg error_message)

- int perturbations_rsa_delta_and_theta (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, double k, double *y, double a_prime_over_a, double *pvecthermo, struct perturbations_workspace *ppw, ErrorMsg error_message)
- int perturbations_rsa_idr_delta_and_theta (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, double k, double *y, double a_prime_over_a, double *pvecthermo, struct perturbations_workspace *ppw, ErrorMsg error_message)

4.19.1 Detailed Description

Documented perturbation module

Julien Lesgourgues, 23.09.2010

Deals with the perturbation evolution. This module has two purposes:

- at the beginning; to initialize the perturbations, i.e. to integrate the perturbation equations, and store temporarily the terms contributing to the source functions as a function of conformal time. Then, to perform a few manipulations of these terms in order to infer the actual source functions $S^X(k,\tau)$, and to store them as a function of conformal time inside an interpolation table.
- at any time in the code; to evaluate the source functions at a given conformal time (by interpolating within the interpolation table).

Hence the following functions can be called from other modules:

1. perturbations_init() at the beginning (but after background_init() and thermodynamics_init())
2. perturbations_sources_at_tau() at any later time
3. perturbations_free() at the end, when no more calls to perturbations_sources_at_tau() are needed

4.19.2 Function Documentation

4.19.2.1 perturbations_sources_at_tau()

```
int perturbations_sources_at_tau (
             struct perturbations * ppt,
             int index_md,
             int index_ic,
             int index_tp,
             double tau,
             double * psource )
```

Source function $S^X(k,\tau)$ at a given conformal time tau.

Evaluate source functions at given conformal time tau by reading the pre-computed table and interpolating.

Parameters

ppt	Input: pointer to perturbation structure containing interpolation tables
index_md	Input: index of requested mode
index_ic	Input: index of requested initial condition
index_tp	Input: index of requested source function type
tau	Input: any value of conformal time
psource	Output: vector (already allocated) of source function as a function of k

Returns

the error status

Summary:

- define local variables
- interpolate in pre-computed table contained in ppt
- linear interpolation at early times (z>z_max_pk), available, but actually never used by default version of CLASS
- more accurate spline interpolation at late times (z<z_max_pk), used in the calculation of output quantitites like transfer functions T(k,z) or power spectra P(k,z)

4.19.2.2 perturbations_output_data()

```
int perturbations_output_data (
              struct background * pba,
              struct perturbations * ppt,
              enum file_format output_format,
              double z,
              int number_of_titles,
              double * data )
```

Function called by the output module or the wrappers, which returns all the source functions $S^X(k,\tau)$ at a given conformal time tau corresponding to the input redshift z.

Parameters

pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
output_format	Input: choice of ordering and normalisation for the output quantities
z	Input: redshift
number_of_titles	Input: number of requested source functions (found in perturbations_output_titles)
data	Output: vector of all source functions for all k values and initial conditions (previously allocated with the right size)

Returns

the error status

- compute $T_i(k)$ for each k (if several ic's, compute it for each ic; if z_pk = 0, this is done by directly reading inside the pre-computed table; if not, this is done by interpolating the table at the correct value of tau.
- store data

4.19.2.3 perturbations_output_titles()

```
int perturbations_output_titles (
              struct background * pba,
              struct perturbations * ppt,
              enum file_format output_format,
              char titles[_MAXTITLESTRINGLENGTH_] )
```

Fill array of strings with the name of the requested 'mTk, vTk' functions (transfer functions as a function of wavenumber for fixed times).

Parameters

pba	Input: pointer to the background structure
ppt	Input: pointer to the perturbation structure
output_format	Input: flag for the format
titles	Output: name strings

Returns

the error status

4.19.2.4 perturbations_output_firstline_and_ic_suffix()

```
int perturbations_output_firstline_and_ic_suffix (
              struct perturbations * ppt,
              int index_ic,
              char first_line[_LINE_LENGTH_MAX_],
              char ic_suffix[_SUFFIXNAMESIZE_] )
```

Fill strings that will be used when writing the transfer functions and the spectra in files (in the file names and in the comment at the beginning of each file).

Parameters

ppt	Input: pointer to the perturbation structure
index_ic	Input: index of the initial condition
first_line	Output: line of comment
ic_suffix	Output: suffix for the output file name

Returns

the error status

4.19.2.5 perturbations_init()

```
int perturbations_init (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt )
```

Initialize the perturbations structure, and in particular the table of source functions.

Main steps:

- given the values of the flags describing which kind of perturbations should be considered (modes↩ : scalar/vector/tensor, initial conditions, type of source functions needed...), initialize indices and wavenumber list
- define the time sampling for the output source functions
- for each mode (scalar/vector/tensor): initialize the indices of relevant perturbations, integrate the differential system, compute and store the source functions.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Output: Initialized perturbation structure

Returns

the error status

Summary:

- define local variables
- perform preliminary checks
- initialize all indices and lists in perturbations structure using perturbations_indices()
- define the common time sampling for all sources using perturbations_timesampling_for_sources()
- if we want to store perturbations for given k values, write titles and allocate storage
- create an array of workspaces in multi-thread case
- loop over modes (scalar, tensors, etc). For each mode:

- --> (a) create a workspace (one per thread in multi-thread case)
- --> (b) initialize indices of vectors of perturbations with perturbations_indices_of_current_vectors()
- --> (c) loop over initial conditions and wavenumbers; for each of them, evolve perturbations and compute source functions with perturbations_solve()
- spline the source array with respect to the time variable

4.19.2.6 perturbations_free_input()

```
int perturbations_free_input (
             struct perturbations * ppt )
```

Free all memory space allocated by input.

Called by perturbations_free(), during shooting and if shooting failed

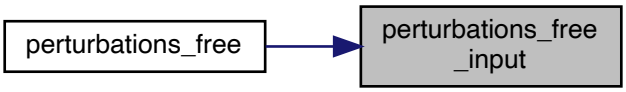
Parameters

ppt	Input: perturbation structure with input pointers to be freed

Returns

the error status

Here is the caller graph for this function:



4.19.2.7 perturbations_free()

```
int perturbations_free (
             struct perturbations * ppt )
```

Free all memory space allocated by perturbations_init().

To be called at the end of each run, only when no further calls to perturbations_sources_at_tau() are needed.

Parameters

ppt	Input: perturbation structure to be freed

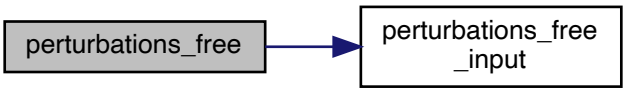
Returns

the error status

Stuff related to perturbations output:

- Free non-NULL pointers

Here is the call graph for this function:



4.19.2.8 perturbations_indices()

```
int perturbations_indices (
              struct precision * ppr,
              struct background * pba,
              struct thermodynamics * pth,
              struct perturbations * ppt )
```

Initialize all indices and allocate most arrays in perturbations structure.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input/Output: Initialized perturbation structure

Returns

the error status

Summary:

- define local variables
- count modes (scalar, vector, tensor) and assign corresponding indices
- allocate array of number of types for each mode, ppt->tp_size[index_md]
- allocate array of number of initial conditions for each mode, ppt->ic_size[index_md]
- allocate array of arrays of source functions for each mode, ppt->source[index_md]
- initialize variables for the output of k values
- initialization of all flags to false (will eventually be set to true later)
- source flags and indices, for sources that all modes have in common (temperature, polarization, ...). For temperature, the term t2 is always non-zero, while other terms are non-zero only for scalars and vectors. For polarization, the term e is always non-zero, while the term b is only for vectors and tensors.
- define k values with perturbations_get_k_list()
- loop over modes. Initialize flags and indices which are specific to each mode.
- (a) scalars
- --> source flags and indices, for sources that are specific to scalars

gamma is not neccessary for converting output to Nbody gauge but is included anyway.

- --> count scalar initial conditions (for scalars: ad, cdi, nid, niv; for tensors: only one) and assign corresponding indices
- (b) vectors
- --> source flags and indices, for sources that are specific to vectors
- --> initial conditions for vectors
- (c) tensors
- --> source flags and indices, for sources that are specific to tensors
- --> only one initial condition for tensors
- (d) for each mode, allocate array of arrays of source functions for each initial conditions and wavenumber, (ppt->source[index_md])[index_ic][index_type]

4.19.2.9 perturbations_timesampling_for_sources()

```
int perturbations_timesampling_for_sources (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt )
```

Define time sampling for source functions.

For each type, compute the list of values of tau at which sources will be sampled. Knowing the number of tau values, allocate all arrays of source functions.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input/Output: Initialized perturbation structure

Returns

the error status

Summary:

- define local variables
- allocate background/thermodynamics vectors
- first, just count the number of sampling points in order to allocate the array containing all values
- (a) if CMB requested, first sampling point = when the universe stops being opaque; otherwise, start sampling gravitational potential at recombination [however, if perturbed recombination is requested, we also need to start the system before recombination. Otherwise, the initial conditions for gas temperature and ionization fraction perturbations (delta_T = 1/3 delta_b, delta_x_e) are not valid].
- (b) next sampling point = previous + ppr->perturbations_sampling_stepsize $*$ timescale_source, where:
- --> if CMB requested: timescale_source1 = $|g/\dot{g}| = |\dot{\kappa} - \ddot{\kappa}/\dot{\kappa}|^{-1}$; timescale_source2 = $|2\ddot{a}/a - (\dot{a}/a)^2|^{-1/2}$ (to sample correctly the late ISW effect; and timescale_source=1/(1/timescale_source1+1/timescale_source2); repeat till today.
- --> if CMB not requested: timescale_source = 1/aH; repeat till today.
- --> infer total number of time steps, ppt->tau_size
- --> allocate array of time steps, ppt->tau_sampling[index_tau]
- --> repeat the same steps, now filling the array with each tau value:
- --> (b.1.) first sampling point = when the universe stops being opaque
- --> (b.2.) next sampling point = previous + ppr->perturbations_sampling_stepsize $*$ timescale_source, where timescale_source1 = $|g/\dot{g}| = |\dot{\kappa} - \ddot{\kappa}/\dot{\kappa}|^{-1}$; timescale_source2 = $|2\ddot{a}/a - (\dot{a}/a)^2|^{-1/2}$ (to sample correctly the late ISW effect; and timescale_source=1/(1/timescale_source1+1/timescale_source2); repeat till today. If CMB not requested: timescale_source = 1/aH; repeat till today.
- last sampling point = exactly today
- check the maximum redshift z_max_pk at which the Fourier transfer functions $T_i(k,z)$ should be computable by interpolation. If it is equal to zero, only $T_i(k, z=0)$ needs to be computed. If it is higher, we will store a table of log(tau) in the relevant time range, generously encompassing the range 0<z<z_max_pk, and used for the intepolation of sources
- loop over modes, initial conditions and types. For each of them, allocate array of source functions.

4.19.2.10 perturbations_get_k_list()

```
int perturbations_get_k_list (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt )
```

Define the number of comoving wavenumbers using the information passed in the precision structure.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer to perturbation structure

Returns

the error status

Summary:

- allocate arrays related to k list for each mode
- scalar modes
- --> find k_max (as well as k_max_cmb[ppt->index_md_scalars], k_max_cl[ppt->index_md_scalars])
- --> test that result for k_min, k_max make sense
- vector modes
- --> find k_max (as well as k_max_cmb[ppt->index_md_vectors], k_max_cl[ppt->index_md_vectors])
- --> test that result for k_min, k_max make sense
- tensor modes
- --> find k_max (as well as k_max_cmb[ppt->index_md_tensors], k_max_cl[ppt->index_md_tensors])
- --> test that result for k_min, k_max make sense
- If user asked for k_output_values, add those to all k lists:
- --> Find indices in ppt->k[index_md] corresponding to 'k_output_values'. We are assuming that ppt->k is sorted and growing, and we have made sure that ppt->k_output_values is also sorted and growing.
- --> Decide if we should add k_output_value now. This has to be this complicated, since we can only compare the k-values when both indices are in range.
- --> The two MIN statements are here because in a normal run, the cl and cmb arrays contain a single k value larger than their respective k_max. We are mimicking this behavior.
- finally, find the global k_min and k_max for the ensemble of all modes 9scalars, vectors, tensors)

4.19.2.11 perturbations_workspace_init()

```
int perturbations_workspace_init (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             int index_md,
             struct perturbations_workspace * ppw )
```

Initialize a perturbations_workspace structure. All fields are allocated here, with the exception of the perturbations_vector '-->pv' field, which is allocated separately in perturbations_vector_init. We allocate one such perturbations_workspace structure per thread and per mode (scalar/../tensor). Then, for each thread, all initial conditions and wavenumbers will use the same workspace.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to the thermodynamics structure
ppt	Input: pointer to the perturbation structure
index_md	Input: index of mode under consideration (scalar/.../tensor)
ppw	Input/Output: pointer to perturbations_workspace structure which fields are allocated or filled here

Returns

the error status

Summary:

- define local variables
- Compute maximum l_max for any multipole
- Allocate s_l[ ] array for freestreaming of multipoles (see arXiv:1305.3261) and initialize to 1.0, which is the K=0 value.
- define indices of metric perturbations obeying constraint equations (this can be done once and for all, because the vector of metric perturbations is the same whatever the approximation scheme, unlike the vector of quantities to be integrated, which is allocated separately in perturbations_vector_init)
- allocate some workspace in which we will store temporarily the values of background, thermodynamics, metric and source quantities at a given time
- count number of approximations, initialize their indices, and allocate their flags
- For definiteness, initialize approximation flags to arbitrary values (correct values are overwritten in pertub_↩ find_approximation_switches)
- allocate fields where some of the perturbations are stored

4.19.2.12 perturbations_workspace_free()

```
int perturbations_workspace_free (
             struct perturbations * ppt,
             int index_md,
             struct perturbations_workspace * ppw )
```

Free the perturbations_workspace structure (with the exception of the perturbations_vector '-->pv' field, which is freed separately in perturbations_vector_free).

Parameters

ppt	Input: pointer to the perturbation structure
index_md	Input: index of mode under consideration (scalar/.../tensor)
ppw	Input: pointer to perturbations_workspace structure to be freed

Returns

the error status

4.19.2.13 perturbations_solve()

```
int perturbations_solve (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             int index_md,
             int index_ic,
             int index_k,
             struct perturbations_workspace * ppw )
```

Solve the perturbation evolution for a given mode, initial condition and wavenumber, and compute the corresponding source functions.

For a given mode, initial condition and wavenumber, this function finds the time ranges over which the perturbations can be described within a given approximation. For each such range, it initializes (or redistributes) perturbations using perturbations_vector_init(), and integrates over time. Whenever a "source sampling time" is passed, the source terms are computed and stored in the source table using perturbations_sources().

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to the thermodynamics structure
ppt	Input/Output: pointer to the perturbation structure (output source functions S(k,tau) written here)
index_md	Input: index of mode under consideration (scalar/.../tensor)
index_ic	Input: index of initial condition under consideration (ad, iso...)
index_k	Input: index of wavenumber
ppw	Input: pointer to perturbations_workspace structure containing index values and workspaces

Returns

the error status

Summary:

- define local variables
- initialize indices relevant for back/thermo tables search
- get wavenumber value
- If non-zero curvature, update array of free-streaming coefficients ppw->s_l
- maximum value of tau for which sources are calculated for this wavenumber
- using bisection, compute minimum value of tau for which this wavenumber is integrated
- find the number of intervals over which approximation scheme is constant
- fill the structure containing all fixed parameters, indices and workspaces needed by perturbations_derivs
- check whether we need to print perturbations to a file for this wavenumber
- loop over intervals over which approximation scheme is uniform. For each interval:
- --> (a) fix the approximation scheme
- --> (b) get the previous approximation scheme. If the current interval starts from the initial time tau_ini, the previous approximation is set to be a NULL pointer, so that the function perturbations_vector_init() knows that perturbations must be initialized
- --> (c) define the vector of perturbations to be integrated over. If the current interval starts from the initial time tau_ini, fill the vector with initial conditions for each mode. If it starts from an approximation switching point, redistribute correctly the perturbations from the previous to the new vector of perturbations.
- --> (d) integrate the perturbations over the current interval.
- if perturbations were printed in a file, close the file
- fill the source terms array with zeros for all times between the last integrated time tau_max and tau_today.
- free quantities allocated at the beginning of the routine

4.19.2.14 perturbations_prepare_k_output()

```
int perturbations_prepare_k_output (
             struct background * pba,
             struct perturbations * ppt )
```

Fill array of strings with the name of the 'k_output_values' functions (transfer functions as a function of time, for fixed values of k).

Parameters

pba	Input: pointer to the background structure
ppt	Input/Output: pointer to the perturbation structure

Returns

the error status

Write titles for all perturbations that we would like to print/store.

Fluid

4.19.2.15 perturbations_find_approximation_number()

```
int perturbations_find_approximation_number (
              struct precision * ppr,
              struct background * pba,
              struct thermodynamics * pth,
              struct perturbations * ppt,
              int index_md,
              double k,
              struct perturbations_workspace * ppw,
              double tau_ini,
              double tau_end,
              int * interval_number,
              int * interval_number_of )
```

For a given mode and wavenumber, find the number of intervals of time between tau_ini and tau_end such that the approximation scheme (and the number of perturbation equations) is uniform.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to the thermodynamics structure
ppt	Input: pointer to the perturbation structure
index_md	Input: index of mode under consideration (scalar/.../tensor)
k	Input: index of wavenumber
ppw	Input: pointer to perturbations_workspace structure containing index values and workspaces
tau_ini	Input: initial time of the perturbation integration
tau_end	Input: final time of the perturbation integration
interval_number	Output: total number of intervals
interval_number_of	Output: number of intervals with respect to each particular approximation

Returns

the error status

Summary:

- fix default number of intervals to one (if no approximation switch)
- loop over each approximation and add the number of approximation switching times

4.19.2.16 perturbations_find_approximation_switches()

```
int perturbations_find_approximation_switches (
              struct precision * ppr,
              struct background * pba,
              struct thermodynamics * pth,
              struct perturbations * ppt,
              int index_md,
              double k,
              struct perturbations_workspace * ppw,
              double tau_ini,
              double tau_end,
              double precision,
              int interval_number,
              int * interval_number_of,
              double * interval_limit,
              int ** interval_approx )
```

For a given mode and wavenumber, find the values of time at which the approximation changes.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to the thermodynamics structure
ppt	Input: pointer to the perturbation structure
index_md	Input: index of mode under consideration (scalar/.../tensor)
k	Input: index of wavenumber
ppw	Input: pointer to perturbations_workspace structure containing index values and workspaces
tau_ini	Input: initial time of the perturbation integration
tau_end	Input: final time of the perturbation integration
precision	Input: tolerance on output values
interval_number	Input: total number of intervals
interval_number_of	Input: number of intervals with respect to each particular approximation
interval_limit	Output: value of time at the boundary of the intervals: tau_ini, tau_switch1, ..., tau_end
interval_approx	Output: value of approximations in each interval

Returns

the error status

Summary:

- write in output arrays the initial time and approximation
- if there are no approximation switches, just write final time and return
- if there are switches, consider approximations one after each other. Find switching time by bisection. Store all switches in arbitrary order in array unsorted_tau_switch[ ]
- now sort interval limits in correct order
- store each approximation in chronological order

4.19.2.17 perturbations_vector_init()

```
int perturbations_vector_init (
              struct precision * ppr,
              struct background * pba,
              struct thermodynamics * pth,
              struct perturbations * ppt,
              int index_md,
              int index_ic,
              double k,
              double tau,
              struct perturbations_workspace * ppw,
              int * pa_old )
```

Initialize the field '-->pv' of a perturbations_workspace structure, which is a perturbations_vector structure. This structure contains indices and values of all quantities which need to be integrated with respect to time (and only them: quantities fixed analytically or obeying constraint equations are NOT included in this vector). This routine distinguishes between two cases:

--> the input pa_old is set to the NULL pointer:

This happens when we start integrating over a new wavenumber and we want to set initial conditions for the perturbations. Then, it is assumed that ppw-->pv is not yet allocated. This routine allocates it, defines all indices, and then fills the vector ppw-->pv-->y with the initial conditions defined in perturbations_initial_conditions.

--> the input pa_old is not set to the NULL pointer and describes some set of approximations:

This happens when we need to change approximation scheme while integrating over a given wavenumber. The new approximation described by ppw-->pa is then different from pa_old. Then, this routine allocates a new vector with a new size and new index values; it fills this vector with initial conditions taken from the previous vector passed as an input in ppw-->pv, and eventually with some analytic approximations for the new variables appearing at this time; then the new vector comes in replacement of the old one, which is freed.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to the thermodynamics structure
ppt	Input: pointer to the perturbation structure
index_md	Input: index of mode under consideration (scalar/.../tensor)
index_ic	Input: index of initial condition under consideration (ad, iso...)
k	Input: wavenumber
tau	Input: conformal time
ppw	Input/Output: workspace containing in input the approximation scheme, the background/thermodynamics/metric quantities, and eventually the previous vector y; and in output the new vector y.
pa_old	Input: NULL is we need to set y to initial conditions for a new wavenumber; points towards a perturbations_approximations if we want to switch of approximation.

Returns

the error status

Summary:

- define local variables
- allocate a new perturbations_vector structure to which ppw-->pv will point at the end of the routine
- initialize pointers to NULL (they will be allocated later if needed), relevant for perturbations_vector_free()
- define all indices in this new vector (depends on approximation scheme, described by the input structure ppw-->pa)
- (a) metric perturbations V or h_v depending on gauge
- (b) metric perturbation h is a propagating degree of freedom, so h and hdot are included in the vector of ordinary perturbations, no in that of metric perturbations
- allocate vectors for storing the values of all these quantities and their time-derivatives at a given time
- specify which perturbations are needed in the evaluation of source terms
- case of setting initial conditions for a new wavenumber
- --> (a) check that current approximation scheme is consistent with initial conditions
- --> (b) let ppw-->pv points towards the perturbations_vector structure that we just created
- --> (c) fill the vector ppw-->pv-->y with appropriate initial conditions
- case of switching approximation while a wavenumber is being integrated
- --> (a) for the scalar mode:
- —> (a.1.) check that the change of approximation scheme makes sense (note: before calling this routine there is already a check that we wish to change only one approximation flag at a time)
- —> (a.2.) some variables (b, cdm, fld, ...) are not affected by any approximation. They need to be reconducted whatever the approximation switching is. We treat them here. Below we will treat other variables case by case.
- --> (b) for the vector mode
- —> (b.1.) check that the change of approximation scheme makes sense (note: before calling this routine there is already a check that we wish to change only one approximation flag at a time)
- —> (b.2.) some variables (gw, gwdot, ...) are not affected by any approximation. They need to be reconducted whatever the approximation switching is. We treat them here. Below we will treat other variables case by case.
- --> (c) for the tensor mode
- —> (c.1.) check that the change of approximation scheme makes sense (note: before calling this routine there is already a check that we wish to change only one approximation flag at a time)
- —> (c.2.) some variables (gw, gwdot, ...) are not affected by any approximation. They need to be reconducted whatever the approximation switching is. We treat them here. Below we will treat other variables case by case.
- --> (d) free the previous vector of perturbations
- --> (e) let ppw-->pv points towards the perturbations_vector structure that we just created

4.19.2.18 perturbations_vector_free()

```
int perturbations_vector_free (
             struct perturbations_vector * pv )
```

Free the perturbations_vector structure.

Parameters

pv	Input: pointer to perturbations_vector structure to be freed

Returns

the error status

4.19.2.19 perturbations_initial_conditions()

```
int perturbations_initial_conditions (
             struct precision * ppr,
             struct background * pba,
             struct perturbations * ppt,
             int index_md,
             int index_ic,
             double k,
             double tau,
             struct perturbations_workspace * ppw )
```

For each mode, wavenumber and initial condition, this function initializes in the vector all values of perturbed variables (in a given gauge). It is assumed here that all values have previously been set to zero, only non-zero values are set here.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
ppt	Input: pointer to the perturbation structure
index_md	Input: index of mode under consideration (scalar/.../tensor)
index_ic	Input: index of initial condition under consideration (ad, iso...)
k	Input: wavenumber
tau	Input: conformal time
ppw	Input/Output: workspace containing in input the approximation scheme, the background/thermodynamics/metric quantities, and eventually the previous vector y; and in output the new vector y.

Returns

the error status

Summary:

--> Declare local variables

--> For scalars

- (a) compute relevant background quantities: compute rho_r, rho_m, rho_nu (= all relativistic except photons), and their ratio.

- (b) starts by setting everything in synchronous gauge. If another gauge is needed, we will perform a gauge transformation below.
- --> (b.1.) adiabatic
- —> Canonical field (solving for the perturbations): initial perturbations set to zero, they should reach the attractor soon enough.
- —> TODO: Incorporate the attractor IC from 1004.5509. delta_phi $= -(a/k)^2/\phi'(\rho+p)\theta$, delta_phi_prime $= a^2/\phi'$ (delta_rho_phi + V'delta_phi), and assume theta, delta_rho as for perfect fluid with $c_s^2 = 1$ and w = 1/3 (ASSUMES radiation TRACKING)
- --> (b.2.) Cold dark matter Isocurvature
- --> (b.3.) Baryon Isocurvature
- --> (b.4.) Neutrino density Isocurvature
- --> (b.5.) Neutrino velocity Isocurvature
- (c) If the needed gauge is really the synchronous gauge, we need to affect the previously computed value of eta to the actual variable eta
- (d) If the needed gauge is the newtonian gauge, we must compute alpha and then perform a gauge transformation for each variable
- (e) In any gauge, we should now implement the relativistic initial conditions in ur and ncdm variables

--> For tensors

tensor initial conditions take into account the fact that scalar (resp. tensor) C_l's are related to the real space power spectrum of curvature (resp. of the tensor part of metric perturbations)

$$< R(x)R(x) > \sum_{ij} < h_{ij}(x)h^{ij}(x) >$$

In momentum space it is conventional to use the modes R(k) and h(k) where the quantity h obeying to the equation of propagation:

$$h" + \frac{2a'}{a}h + [k2 + 2K]h = 12\pi Ga2(\rho + p)\sigma = 8\pi Ga2p\pi$$

and the power spectra in real space and momentum space are related through:

$$< R(x)R(x) >= \int \frac{dk}{k}\left[\frac{k^3}{2\pi^2} < R(k)R(k)^* >\right] = \int \frac{dk}{k}\mathcal{P}_R(k)$$

$$\sum_{ij} < h_{ij}(x)h^{ij}(x) >= \frac{dk}{k}\left[\frac{k^3}{2\pi^2}F\left(\frac{k^2}{K}\right) < h(k)h(k)^* >\right] = \int \frac{dk}{k}F\left(\frac{k^2}{K}\right)\mathcal{P}_h(k)$$

where $\mathcal{P}_R$ and $\mathcal{P}_h$ are the dimensionless spectrum of curvature R, and F is a function of k2/K, where K is the curvature parameter. F is equal to one in flat space (K=0), and coming from the contraction of the laplacian eigentensor Q_{ij} with itself. We will give F explicitly below.

Similarly the scalar (S) and tensor (T) C_l's are given by

$$C_l^S = 4\pi \int \frac{dk}{k} [\Delta_l^S(q)]^2 \mathcal{P}_R(k)$$

$$C_l^T = 4\pi \int \frac{dk}{k} [\Delta_l^T(q)]^2 F\left(\frac{k^2}{K}\right) \mathcal{P}_h(k)$$

The usual convention for the tensor-to-scalar ratio $r = A_t/A_s$ at pivot scale = 16 epsilon in single-field inflation is such that for constant $\mathcal{P}_R(k)$ and $\mathcal{P}_h(k)$,

$$r = 6\frac{\mathcal{P}_h(k)}{\mathcal{P}_R(k)}$$

so

$$\mathcal{P}_h(k) = \frac{\mathcal{P}_R(k) r}{6} = \frac{A_s r}{6} = \frac{A_t}{6}$$

A priori it would make sense to say that for a power-law primordial spectrum there is an extra factor $(k/k_{pivot})^{n_t}$ (and eventually running and so on and so forth...)

However it has been shown that the minimal models of inflation in a negatively curved bubble lead to $\mathcal{P}_h(k) = \tanh(\pi * \nu/2)$. In open models it is customary to define the tensor tilt in a non-flat universe as a deviation from this behavior rather than from true scale-invariance in the above sense.

Hence we should have

$$\mathcal{P}_h(k) = \frac{A_t}{6} [\tanh(\pi * \frac{\nu}{2})](k/k_{pivot})^{(n_t + ...)}$$

where the brackets

$$[...]$$

mean "if K<0"

Then

$$C_l^T = 4\pi \int \frac{dk}{k} [\Delta_l^T(q)]^2 F\left(\frac{k^2}{K}\right) \frac{A_t}{6} [\tanh(\pi * \frac{\nu}{2})](k/k_{pivot})^{(n_t + ...)}$$

In the code, it is then a matter of choice to write:

- In the primordial module: $\mathcal{P}_h(k) = \frac{A_t}{6} \tanh(\pi * \frac{\nu}{2})(k/k^*)^{n_T}$
- In the perturbation initial conditions: $h = 1$
- In the harmonic module: $C_l^T = \frac{4}{\pi} \int \frac{dk}{k} [\Delta_l^T(q)]^2 F\left(\frac{k^2}{K}\right) \mathcal{P}_h(k)$

or:

- In the primordial module: $\mathcal{P}_h(k) = A_t(k/k^*)^{n_T}$

- In the perturbation initial conditions: $h = \sqrt{[F\left(\frac{k^2}{K}\right)/6]\tanh\left(\pi * \frac{\nu}{2}\right)}$
- In the harmonic module: $C_l^T = \frac{4}{\pi}\int \frac{dk}{k}[\Delta_l^T(q)]^2\mathcal{P}_h(k)$

We choose this last option, such that the primordial and harmonic module differ minimally in flat and non-flat space. Then we must impose

$$h = \sqrt{\left(\frac{F}{6}\right)\tanh\left(\pi * \frac{\nu}{2}\right)}$$

The factor F is found to be given by:

$$\sum_{ij} < h_{ij}(x)h^{ij}(x) >= \int \frac{dk}{k}\frac{k2(k2-K)}{(k2+3K)(k2+2K)}\mathcal{P}_h(k)$$

Introducing as usual $q2 = k2 - 3K$ and using qdq = kdk gives

$$\sum_{ij} < h_{ij}(x)h^{ij}(x) >= \int \frac{dk}{k}\frac{(q2-3K)(q2-4K)}{q2(q2-K)}\mathcal{P}_h(k)$$

Using qdq = kdk this is equivalent to

$$\sum_{ij} < h_{ij}(x)h^{ij}(x) >= \int \frac{dq}{q}\frac{q2-4K}{q2-K}\mathcal{P}_h(k(q))$$

Finally, introducing $\nu = q/\sqrt{|K|}$ and sgnK=SIGN(k) $= \pm 1$, this could also be written

$$\sum_{ij} < h_{ij}(x)h^{ij}(x) >= \int \frac{d\nu}{\nu}\frac{(\nu2-4sgnK)}{(\nu2-sgnK)}\mathcal{P}_h(k(\nu))$$

Equation (43,44) of Hu, Seljak, White, Zaldarriaga is equivalent to absorbing the above factor $(\nu2-4sgnK)/(\nu2-sgnK)$ in the definition of the primordial spectrum. Since the initial condition should be written in terms of k rather than nu, they should read

$$h = \sqrt{[k2(k2-K)]/[(k2+3K)(k2+2K)]/6 * \tanh\left(\pi * \frac{\nu}{2}\right)}$$

We leave the freedom to multiply by an arbitrary number ppr->gw_ini. The standard convention corresponding to standard definitions of r, A_T, n_T is however ppr->gw_ini=1.

4.19.2.20 perturbations_approximations()

```
int perturbations_approximations (
              struct precision * ppr,
              struct background * pba,
              struct thermodynamics * pth,
              struct perturbations * ppt,
              int index_md,
              double k,
              double tau,
              struct perturbations_workspace * ppw )
```

Evaluate background/thermodynamics at τ, infer useful flags / time scales for integrating perturbations.

Evaluate background quantities at τ, as well as thermodynamics for scalar mode; infer useful flags and time scales for integrating the perturbations:

- check whether tight-coupling approximation is needed.
- check whether radiation (photons, massless neutrinos...) perturbations are needed.
- choose step of integration: step = ppr->perturbations_integration_stepsize $*$ min_time_scale, where min_↩ time_scale = smallest time scale involved in the equations. There are three time scales to compare:
  1. that of recombination, $\tau_c = 1/\kappa'$
  2. Hubble time scale, $\tau_h = a/a'$
  3. Fourier mode, $\tau_k = 1/k$

So, in general, min_time_scale = $\min(\tau_c, \tau_b, \tau_h, \tau_k)$.

However, if $\tau_c \ll \tau_h$ and $\tau_c \ll \tau_k$, we can use the tight-coupling regime for photons and write equations in such way that the time scale τ_c becomes irrelevant (no effective mass term in $1/\tau_c$). Then, the smallest scale in the equations is only $\min(\tau_h, \tau_k)$. In practise, it is sufficient to use only the condition $\tau_c \ll \tau_h$.

Also, if $\rho_{matter} \gg \rho_{radiation}$ and $k \gg aH$, we can switch off radiation perturbations (i.e. switch on the free-streaming approximation) and then the smallest scale is simply τ_h.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer to the perturbation structure
index_md	Input: index of mode under consideration (scalar/.../tensor)
k	Input: wavenumber
tau	Input: conformal time
ppw	Input/Output: in output contains the approximation to be used at this time

Returns

the error status

Summary:

- define local variables
- compute Fourier mode time scale = $\tau_k = 1/k$
- evaluate background quantities with background_at_tau() and Hubble time scale $\tau_h = a/a'$
- for scalar modes:
- --> (a) evaluate thermodynamical quantities with thermodynamics_at_z()
- —> (b.1.) if $\kappa' = 0$, recombination is finished; tight-coupling approximation must be off
- —> (b.2.) if $\kappa' \neq 0$, recombination is not finished: check tight-coupling approximation
- —-> (b.2.a) compute recombination time scale for photons, $\tau_\gamma = 1/\kappa'$
- —-> (b.2.b) check whether tight-coupling approximation should be on
- --> (c) free-streaming approximations
- for tensor modes:
- --> (a) evaluate thermodynamical quantities with thermodynamics_at_z()
- —> (b.1.) if $\kappa' = 0$, recombination is finished; tight-coupling approximation must be off
- —> (b.2.) if $\kappa' \neq 0$, recombination is not finished: check tight-coupling approximation
- —-> (b.2.a) compute recombination time scale for photons, $\tau_\gamma = 1/\kappa'$
- —-> (b.2.b) check whether tight-coupling approximation should be on

4.19.2.21 perturbations_timescale()

```
int perturbations_timescale (
              double tau,
              void * parameters_and_workspace,
              double * timescale,
              ErrorMsg error_message )
```

Compute typical timescale over which the perturbation equations vary. Some integrators (e.g. Runge-Kunta) benefit from calling this routine at each step in order to adapt the next step.

This is one of the few functions in the code which is passed to the generic_integrator() routine. Since generic_integrator() should work with functions passed from various modules, the format of the arguments is a bit special:

- fixed parameters and workspaces are passed through a generic pointer. generic_integrator() doesn't know the content of this pointer.
- the error management is a bit special: errors are not written as usual to pth->error_message, but to a generic error_message passed in the list of arguments.

Parameters

tau	Input: conformal time
parameters_and_workspace	Input: fixed parameters (e.g. indices), workspace, approximation used, etc.
timescale	Output: perturbation variation timescale (given the approximation used)
error_message	Output: error message

Summary:

- define local variables
- extract the fields of the parameter_and_workspace input structure
- compute Fourier mode time scale = $\tau_k = 1/k$
- evaluate background quantities with background_at_tau() and Hubble time scale $\tau_h = a/a'$
- for scalars modes:
- --> compute recombination time scale for photons, $\tau_\gamma = 1/\kappa'$
- for vector modes:
- --> compute recombination time scale for photons, $\tau_\gamma = 1/\kappa'$
- for tensor modes:
- --> compute recombination time scale for photons, $\tau_\gamma = 1/\kappa'$

4.19.2.22 perturbations_einstein()

```
int perturbations_einstein (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             int index_md,
             double k,
             double tau,
             double * y,
             struct perturbations_workspace * ppw )
```

Compute metric perturbations (those not integrated over time) using Einstein equations

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer to the perturbation structure
index_md	Input: index of mode under consideration (scalar/.../tensor)
k	Input: wavenumber
tau	Input: conformal time
y	Input: vector of perturbations (those integrated over time) (already allocated)
ppw	Input/Output: in output contains the updated metric perturbations

Returns

the error status

Summary:

- define local variables
- define wavenumber and scale factor related quantities
- sum up perturbations from all species
- for scalar modes:
- --> infer metric perturbations from Einstein equations
- for vector modes
- for tensor modes

4.19.2.23 perturbations_total_stress_energy()

```
int perturbations_total_stress_energy (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             int index_md,
             double k,
             double * y,
             struct perturbations_workspace * ppw )
```

Summary:

- define local variables

Variables used for FLD and PPF

- wavenumber and scale factor related quantities
- for scalar modes
- --> (a) deal with approximation schemes
- —> (a.1.) photons
- -—> (a.1.1.) no approximation
- -—> (a.1.2.) radiation streaming approximation
- -—> (a.1.3.) tight coupling approximation
- —> (a.2.) ur
- —> (a.3.) baryon pressure perturbation
- —> (a.4.) interacting dark radiation
- --> (b) compute the total density, velocity and shear perturbations

We must gauge transform the pressure perturbation from the fluid rest-frame to the gauge we are working in

The equation is too stiff for Runge-Kutta when c_gamma_k_H_square is large. Use the asymptotic solution Gamma=Gamma'=0 in that case.

We must now check the stiffenss criterion again and set Gamma_prime_fld accordingly.

Now construct the pressure perturbation, see 1903.xxxxx.

Construct energy density and pressure for DE (_fld) and the rest (_t). Also compute derivatives.

Compute background quantities X,Y,Z and their derivatives.

Construct theta_t and its derivative from the Euler equation

Analytic derivative of the equation for ppw->rho_plus_p_theta_fld above.

We can finally compute the pressure perturbation using the Euler equation for theta_fld

- for vector modes
- --> photon contribution to vector sources:
- --> baryons
- for tensor modes
- --> photon contribution to gravitational wave source:
- --> ur contribution to gravitational wave source:
- --> ncdm contribution to gravitational wave source:

4.19.2.24 perturbations_sources()

```
int perturbations_sources (
             double tau,
             double * y,
             double * dy,
             int index_tau,
             void * parameters_and_workspace,
             ErrorMsg error_message )
```

Compute the source functions (three terms for temperature, one for E or B modes, etc.)

This is one of the few functions in the code which is passed to the generic_integrator() routine. Since generic_integrator() should work with functions passed from various modules, the format of the arguments is a bit special:

- fixed parameters and workspaces are passed through a generic pointer. generic_integrator() doesn't know the content of this pointer.
- the error management is a bit special: errors are not written as usual to pth->error_message, but to a generic error_message passed in the list of arguments.

Parameters

tau	Input: conformal time
y	Input: vector of perturbations
dy	Input: vector of time derivative of perturbations
index_tau	Input: index in the array tau_sampling
parameters_and_workspace	Input/Output: in input, all parameters needed by perturbations_derivs, in output, source terms
error_message	Output: error message

Returns

the error status

Summary:

- define local variables
- rename structure fields (just to avoid heavy notations)
- get background/thermo quantities in this point
- for scalars
- --> compute metric perturbations
- --> compute quantities depending on approximation schemes
- --> for each type, compute source terms

gamma in N-body gauge. Eq. A.2 in 1811.00904 gives k2gamma = (a'/a)H_T' + k2(phi-psi) - H_T''. The last term is cubersome to calculate (one would need finite derivatives) but usually small. Here we only compute an approximate k2gamma without this last term. If needed, the term could be restored: you can see how T. Tram did it in a previous commit beec79548877e1e43403d1f4de5ddee6741a3c16 (28.02.2019) - then it had to go to harmonic.c, now it could stay in this module. Later this feature was removed for simplicity. Note that to compute the transfer functions in the N-body gauge we do not need k2gamma anyway.

We follow the (debatable) CMBFAST/CAMB convention of not including rho_lambda in rho_tot

- for tensors
- --> compute quantities depending on approximation schemes

4.19.2.25 perturbations_print_variables()

```
int perturbations_print_variables (
             double tau,
             double * y,
             double * dy,
             void * parameters_and_workspace,
             ErrorMsg error_message )
```

When testing the code or a cosmological model, it can be useful to output perturbations at each step of integration (and not just the delta's at each source sampling point, which is achieved simply by asking for matter transfer functions). Then this function can be passed to the generic_evolver routine.

By default, instead of passing this function to generic_evolver, one passes a null pointer. Then this function is just not used.

Parameters

tau	Input: conformal time
y	Input: vector of perturbations
dy	Input: vector of its derivatives (already allocated)
parameters_and_workspace	Input: fixed parameters (e.g. indices)
error_message	Output: error message

Summary:

- define local variables
- ncdm sector begins
- ncdm sector ends
- rename structure fields (just to avoid heavy notations)
- update background/thermo quantities in this point
- update metric perturbations in this point
- calculate perturbed recombination
- for scalar modes
- --> Get delta, deltaP/rho, theta, shear and store in array
- --> TODO: gauge transformation of delta, deltaP/rho (?) and theta using -= 3aH(1+w_ncdm) alpha for delta.
- --> Handle (re-)allocation

Fluid

- for tensor modes:
- --> Handle (re-)allocation

4.19.2.26 perturbations_derivs()

```
int perturbations_derivs (
             double tau,
             double * y,
             double * dy,
             void * parameters_and_workspace,
             ErrorMsg error_message )
```

Compute derivative of all perturbations to be integrated

For each mode (scalar/vector/tensor) and each wavenumber k, this function computes the derivative of all values in the vector of perturbed variables to be integrated.

This is one of the few functions in the code which is passed to the generic_integrator() routine. Since generic_integrator() should work with functions passed from various modules, the format of the arguments is a bit special:

- fixed parameters and workspaces are passed through a generic pointer. generic_integrator() doesn't know what the content of this pointer is.
- errors are not written as usual in pth->error_message, but in a generic error_message passed in the list of arguments.

Parameters

tau	Input: conformal time
y	Input: vector of perturbations
dy	Output: vector of its derivatives (already allocated)
parameters_and_workspace	Input/Output: in input, fixed parameters (e.g. indices); in output, background and thermo quantities evaluated at tau.
error_message	Output: error message

Summary:

- define local variables
- rename the fields of the input structure (just to avoid heavy notations)
- get background/thermo quantities in this point
- get metric perturbations with perturbations_einstein()
- compute related background quantities
- Compute 'generalised cotK function of argument $\sqrt{|K|} * \tau$, for closing hierarchy. (see equation 2.34 in arXiv:1305.3261):
- for scalar modes:
- --> (a) define short-cut notations for the scalar perturbations
- --> (b) perturbed recombination
- --> (c) compute metric-related quantities (depending on gauge; additional gauges can be coded below)
- Each continuity equation contains a term in (theta+metric_continuity) with metric_continuity = (h_prime/2) in synchronous gauge, (-3 phi_prime) in newtonian gauge
- Each Euler equation contains a source term metric_euler with metric_euler = 0 in synchronous gauge, (k2 psi) in newtonian gauge
- Each shear derivative equation contains a source term metric_shear equal to metric_shear = (h_↩ prime+6eta_prime)/2 in synchronous gauge, 0 in newtonian gauge
- metric_shear_prime is the derivative of metric_shear
- In the ufa_class approximation, the leading-order source term is (h_prime/2) in synchronous gauge, (-3 (phi↩ _prime+psi_prime)) in newtonian gauge: we approximate the later by (-6 phi_prime)
- --> (d) if some approximation schemes are turned on, enforce a few y[] values computed in perturbations_↩ einstein
- --> (e) BEGINNING OF ACTUAL SYSTEM OF EQUATIONS OF EVOLUTION
- —> photon temperature density
- —> baryon density
- —> baryon velocity (depends on tight-coupling approximation=tca)
- —-> perturbed recombination has an impact
- —> photon temperature higher momenta and photon polarization (depend on tight-coupling approximation)
- —-> if photon tight-coupling is off
- ---—> define $\Pi = G_{\gamma 0} + G_{\gamma 2} + F_{\gamma 2}$

- ---—> photon temperature velocity
- ---—> photon temperature shear
- ---—> photon temperature l=3
- ---—> photon temperature l>3
- ---—> photon temperature lmax
- ---—> photon polarization l=0
- ---—> photon polarization l=1
- ---—> photon polarization l=2
- ---—> photon polarization l>2
- ---—> photon polarization lmax_pol
- -—> if photon tight-coupling is on:
- ---—> in that case, only need photon velocity
- —> cdm
- -—> newtonian gauge: cdm density and velocity
- -—> synchronous gauge: cdm density only (velocity set to zero by definition of the gauge)
- —> idr
- —> idm_dr
- —> dcdm and dr
- -—> dcdm
- —> dr
- -—> dr F0
- -—> dr F1
- -—> exact dr F2
- -—> exact dr l=3
- -—> exact dr l>3
- -—> exact dr lmax_dr
- —> fluid (fld)
- -—> factors w, w_prime, adiabatic sound speed ca2 (all three background-related), plus actual sound speed in the fluid rest frame cs2
- -—> fluid density
- -—> fluid velocity
- —> scalar field (scf)
- -—> field value
- -—> Klein Gordon equation
- —> interacting dark radiation
- -—> idr velocity

- -—> exact idr shear
- -—> exact idr l=3
- -—> exact idr l>3
- -—> exact idr lmax_dr
- —> ultra-relativistic neutrino/relics (ur)
- -—> if radiation streaming approximation is off
- --—> ur density
- --—> ur velocity
- --—> exact ur shear
- --—> exact ur l=3
- --—> exact ur l>3
- --—> exact ur lmax_ur
- --—> in fluid approximation (ufa): only ur shear needed
- —> non-cold dark matter (ncdm): massive neutrinos, WDM, etc.
- -—> first case: use a fluid approximation (ncdmfa)
- --—> loop over species
- --—> define intermediate quantitites
- --—> exact continuity equation
- --—> exact euler equation
- --—> different ansatz for approximate shear derivative
- --—> jump to next species
- -—> second case: use exact equation (Boltzmann hierarchy on momentum grid)
- --—> loop over species
- --—> loop over momentum
- --—> define intermediate quantities
- --—> ncdm density for given momentum bin
- --—> ncdm velocity for given momentum bin
- --—> ncdm shear for given momentum bin
- --—> ncdm l>3 for given momentum bin
- --—> ncdm lmax for given momentum bin (truncation as in Ma and Bertschinger) but with curvature taken into account a la arXiv:1305.3261
- --—> jump to next momentum bin or species
- —> metric
- —> eta of synchronous gauge
- vector mode
- --> baryon velocity

- tensor modes:
- --> non-cold dark matter (ncdm): massive neutrinos, WDM, etc.
- —> loop over species
- -—> loop over momentum
- -—> define intermediate quantities
- -—> ncdm density for given momentum bin
- -—> ncdm l>0 for given momentum bin
- -—> ncdm lmax for given momentum bin (truncation as in Ma and Bertschinger) but with curvature taken into account a la arXiv:1305.3261
- -—> jump to next momentum bin or species
- --> tensor metric perturbation h (gravitational waves)
- --> its time-derivative

4.19.2.27 perturbations_tca_slip_and_shear()

```
int perturbations_tca_slip_and_shear (
             double * y,
             void * parameters_and_workspace,
             ErrorMsg error_message )
```

Compute the baryon-photon slip (theta_g - theta_b)' and the photon shear in the tight-coupling approximation

Parameters

y	Input: vector of perturbations
parameters_and_workspace	Input/Output: in input, fixed parameters (e.g. indices); in output, slip and shear
error_message	Output: error message

Summary:

- define local variables
- rename the fields of the input structure (just to avoid heavy notations)
- compute related background quantities
- --> (a) define short-cut notations for the scalar perturbations
- --> (b) define short-cut notations used only in tight-coupling approximation
- --> (c) compute metric-related quantities (depending on gauge; additional gauges can be coded below)
- Each continuity equation contains a term in (theta+metric_continuity) with metric_continuity = (h_prime/2) in synchronous gauge, (-3 phi_prime) in newtonian gauge
- Each Euler equation contains a source term metric_euler with metric_euler = 0 in synchronous gauge, (k2 psi) in newtonian gauge

- Each shear derivative equation contains a source term metric_shear equal to metric_shear = (h_↩ prime+6eta_prime)/2 in synchronous gauge, 0 in newtonian gauge
- metric_shear_prime is the derivative of metric_shear
- In the ufa_class approximation, the leading-order source term is (h_prime/2) in synchronous gauge, (-3 (phi↩ _prime+psi_prime)) in newtonian gauge: we approximate the later by (-6 phi_prime)
- --> (d) if some approximation schemes are turned on, enforce a few y[ ] values computed in perturbations↩ _einstein
- —> like Ma & Bertschinger
- —> relax assumption dkappa∼a $^{-2}$ (like in CAMB)
- —> also relax assumption cb2∼a $^{-1}$
- —> intermediate quantities for 2nd order tca: shear_g at first order in tight-coupling
- —> intermediate quantities for 2nd order tca: zero order for theta_b' = theta_g'
- —> intermediate quantities for 2nd order tca: shear_g_prime at first order in tight-coupling
- —> 2nd order as in CRS
- —> 2nd order like in CLASS paper
- —> add only the most important 2nd order terms
- —> store tight-coupling values of photon shear and its derivative

4.19.2.28 perturbations_rsa_delta_and_theta()

```
int perturbations_rsa_delta_and_theta (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             double k,
             double * y,
             double a_prime_over_a,
             double * pvecthermo,
             struct perturbations_workspace * ppw,
             ErrorMsg error_message )
```

Compute the density delta and velocity theta of photons and ultra-relativistic neutrinos in the radiation streaming approximation

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer to perturbation structure
k	Input: wavenumber
y	Input: vector of perturbations
a_prime_over↩ _a	Input: a'/a
pvecthermo	Input: vector of thermodynamics quantites
ppw	Input/Output: in input, fixed parameters (e.g. indices); in output, delta and theta
error_message	Output: error message

4.19.2.29 perturbations_rsa_idr_delta_and_theta()

```
int perturbations_rsa_idr_delta_and_theta (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             double k,
             double * y,
             double a_prime_over_a,
             double * pvecthermo,
             struct perturbations_workspace * ppw,
             ErrorMsg error_message )
```

Compute the density delta and velocity theta of interacting dark radiation in its streaming approximation

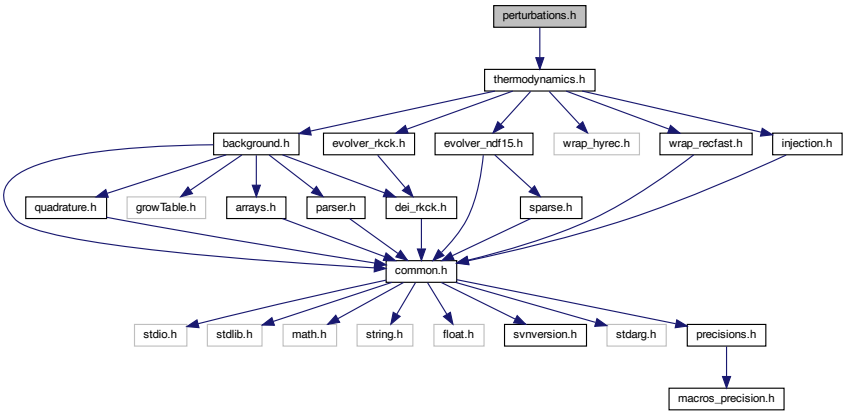
Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer to perturbation structure
k	Input: wavenumber
y	Input: vector of perturbations
a_prime_over_a	Input: a'/a
pvecthermo	Input: vector of thermodynamics quantites
ppw	Input/Output: in input, fixed parameters (e.g. indices); in output, delta and theta
error_message	Output: error message

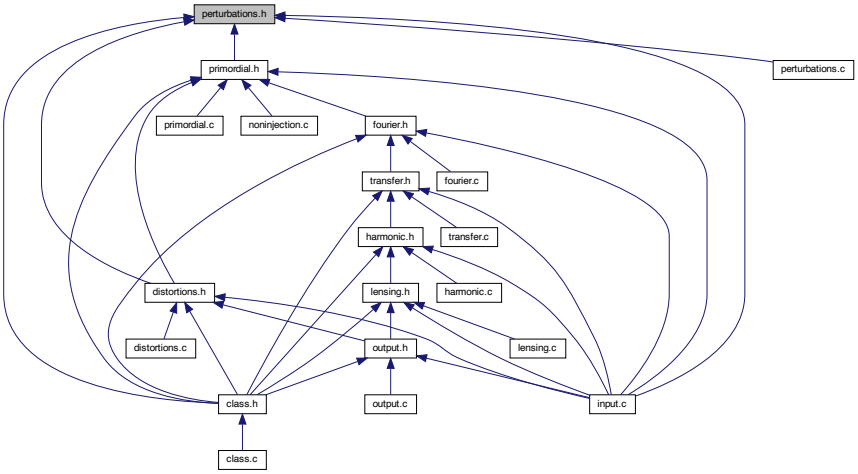
4.20 perturbations.h File Reference

```
#include "thermodynamics.h"
```

Include dependency graph for perturbations.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct perturbations
- struct perturbations_vector
- struct perturbations_workspace
- struct perturbations_parameters_and_workspace

Macros

- #define _SELECTION_NUM_MAX_ 100
- #define _MAX_NUMBER_OF_K_FILES_ 30

Enumerations

- enum tca_flags
- enum tca_method
- enum possible_gauges { newtonian , synchronous }

4.20.1 Detailed Description

Documented includes for perturbation module

4.20.2 Data Structure Documentation

4.20.2.1 struct perturbations

Structure containing everything about perturbations that other modules need to know, in particular tabled values of the source functions $S(k,\tau)$ for all requested modes (scalar/vector/tensor), initial conditions, types (temperature, E-polarization, B-polarization, lensing potential, etc), multipole l and wavenumber k.

Data Fields

short	has_perturbations	do we need to compute perturbations at all ?
short	has_cls	do we need any harmonic space spectrum C_l (and hence Bessel functions, transfer functions, ...)?
short	has_scalars	do we need scalars?
short	has_vectors	do we need vectors?
short	has_tensors	do we need tensors?
short	has_ad	do we need adiabatic mode?
short	has_bi	do we need isocurvature bi mode?
short	has_cdi	do we need isocurvature cdi mode?
short	has_nid	do we need isocurvature nid mode?
short	has_niv	do we need isocurvature niv mode?
short	has_perturbed_recombination	Do we want to consider perturbed temperature and ionization fraction?
enum tensor_methods	tensor_method	Neutrino contribution to tensors way to treat neutrinos in tensor perturbations(neglect, approximate as massless, take exact
short	evolve_tensor_ur	will we evolve ur tensor perturbations (either because we have ur species, or we have ncdm species with massless approximation) ?
short	evolve_tensor_ncdm	will we evolve ncdm tensor perturbations (if we have ncdm species and we use the exact method) ?
short	has_cl_cmb_temperature	do we need C_l's for CMB temperature?
short	has_cl_cmb_polarization	do we need C_l's for CMB polarization?
short	has_cl_cmb_lensing_potential	do we need C_l's for CMB lensing potential?
short	has_cl_lensing_potential	do we need C_l's for galaxy lensing potential?

Data Fields

short	has_cl_number_count	do we need C_l's for density number count?
short	has_pk_matter	do we need matter Fourier spectrum?
short	has_density_transfers	do we need to output individual matter density transfer functions?
short	has_velocity_transfers	do we need to output individual matter velocity transfer functions?
short	has_metricpotential_transfers	do we need to output individual transfer functions for scalar metric perturbations?
short	has_Nbody_gauge_transfers	should we convert density and velocity transfer functions to Nbody gauge?
short	has_nl_corrections_based_on_delta_m	do we want to compute non-linear corrections with an algorithm relying on delta_m (like halofit)?
short	has_nc_density	in dCl, do we want density terms ?
short	has_nc_rsd	in dCl, do we want redshift space distortion terms ?
short	has_nc_lens	in dCl, do we want lensing terms ?
short	has_nc_gr	in dCl, do we want gravity terms ?
int	l_scalar_max	maximum l value for CMB scalars C_l's
int	l_vector_max	maximum l value for CMB vectors C_l's
int	l_tensor_max	maximum l value for CMB tensors C_l's
int	l_lss_max	maximum l value for LSS C_l's (density and lensing potential in bins)
double	k_max_for_pk	maximum value of k in 1/Mpc in P(k) (if C_l's also requested, overseeded by value kmax inferred from l_scalar_max if it is bigger)
int	selection_num	number of selection functions (i.e. bins) for matter density C_l's
enum selection_type	selection	type of selection functions
double	selection_mean[_SELECTION_NUM_MAX_]	centers of selection functions
double	selection_width[_SELECTION_NUM_MAX_]	widths of selection functions
int	switch_sw	in temperature calculation, do we want to include the intrinsic temperature + Sachs Wolfe term?
int	switch_eisw	in temperature calculation, do we want to include the early integrated Sachs Wolfe term?
int	switch_lisw	in temperature calculation, do we want to include the late integrated Sachs Wolfe term?
int	switch_dop	in temperature calculation, do we want to include the Doppler term?
int	switch_pol	in temperature calculation, do we want to include the polarization-related term?
double	eisw_lisw_split_z	at which redshift do we define the cut between eisw and lisw ?
int	store_perturbations	Do we want to store perturbations?

Data Fields

int	k_output_values_num	Number of perturbation outputs (default=0)
double	k_output_values[_MAX_NUMBER_OF_K_FILES_]	List of k values where perturbation output is requested.
double	three_ceff2_ur	3 x effective squared sound speed for the ultrarelativistic perturbations
double	three_cvis2_ur	3 x effective viscosity parameter for the ultrarelativistic perturbations
double	z_max_pk	when we compute only the matter spectrum / transfer functions, but not the CMB, we are sometimes interested to sample source functions at very high redshift, way before recombination. This z_max_pk will then fix the initial sampling time of the sources.
double *	alpha_idm_dr	Angular contribution to collisional term at l>=2 for idm_fr-idr
double *	beta_idr	Angular contribution to collisional term at l>=2 for idr-idr
int	idr_nature	Nature of the interacting dark radiation (free streaming or fluid)
short	has_cmb	do we need CMB-related sources (temperature, polarization) ?
short	has_lss	do we need LSS-related sources (lensing potential, ...) ?
enum possible_gauges	gauge	gauge in which to perform this calculation
int	index_md_scalars	index value for scalars
int	index_md_tensors	index value for tensors
int	index_md_vectors	index value for vectors
int	md_size	number of modes included in computation
int	index_ic_ad	index value for adiabatic
int	index_ic_cdi	index value for CDM isocurvature
int	index_ic_bi	index value for baryon isocurvature
int	index_ic_nid	index value for neutrino density isocurvature
int	index_ic_niv	index value for neutrino velocity isocurvature
int	index_ic_ten	index value for unique possibility for tensors
int *	ic_size	for a given mode, ic_size[index_md] = number of initial conditions included in computation
short	has_source_t	do we need source for CMB temperature?
short	has_source_p	do we need source for CMB polarization?
short	has_source_delta_m	do we need source for delta of total matter?
short	has_source_delta_cb	do we ALSO need source for delta of ONLY cdm and baryon?

Data Fields

short	has_source_delta_tot	do we need source for delta total?
short	has_source_delta_g	do we need source for delta of gammas?
short	has_source_delta_b	do we need source for delta of baryons?
short	has_source_delta_cdm	do we need source for delta of cold dark matter?
short	has_source_delta_idr	do we need source for delta of interacting dark radiation?
short	has_source_delta_idm_dr	do we need source for delta of interacting dark matter (with dr)?
short	has_source_delta_dcdm	do we need source for delta of DCDM?
short	has_source_delta_fld	do we need source for delta of dark energy?
short	has_source_delta_scf	do we need source for delta from scalar field?
short	has_source_delta_dr	do we need source for delta of decay radiation?
short	has_source_delta_ur	do we need source for delta of ultra-relativistic neutrinos/relics?
short	has_source_delta_ncdm	do we need source for delta of all non-cold dark matter species (e.g. massive neutrinos)?
short	has_source_theta_m	do we need source for theta of total matter?
short	has_source_theta_cb	do we ALSO need source for theta of ONLY cdm and baryon?
short	has_source_theta_tot	do we need source for theta total?
short	has_source_theta_g	do we need source for theta of gammas?
short	has_source_theta_b	do we need source for theta of baryons?
short	has_source_theta_cdm	do we need source for theta of cold dark matter?
short	has_source_theta_idr	do we need source for theta of interacting dark radiation?
short	has_source_theta_idm_dr	do we need source for theta of interacting dark matter (with dr)?
short	has_source_theta_dcdm	do we need source for theta of DCDM?
short	has_source_theta_fld	do we need source for theta of dark energy?
short	has_source_theta_scf	do we need source for theta of scalar field?
short	has_source_theta_dr	do we need source for theta of ultra-relativistic neutrinos/relics?
short	has_source_theta_ur	do we need source for theta of ultra-relativistic neutrinos/relics?
short	has_source_theta_ncdm	do we need source for theta of all non-cold dark matter species (e.g. massive neutrinos)?
short	has_source_phi	do we need source for metric fluctuation phi?
short	has_source_phi_prime	do we need source for metric fluctuation phi'?

Data Fields

short	has_source_phi_plus_psi	do we need source for metric fluctuation (phi+psi)?
short	has_source_psi	do we need source for metric fluctuation psi?
short	has_source_h	do we need source for metric fluctuation h?
short	has_source_h_prime	do we need source for metric fluctuation h'?
short	has_source_eta	do we need source for metric fluctuation eta?
short	has_source_eta_prime	do we need source for metric fluctuation eta'?
short	has_source_H_T_Nb_prime	do we need source for metric fluctuation H_T_Nb'?
short	has_source_k2gamma_Nb	do we need source for metric fluctuation gamma in Nbody gauge?
int	index_tp_t0	index value for temperature (j=0 term)
int	index_tp_t1	index value for temperature (j=1 term)
int	index_tp_t2	index value for temperature (j=2 term)
int	index_tp_p	index value for polarization
int	index_tp_delta_m	index value for matter density fluctuation
int	index_tp_delta_cb	index value for delta cb
int	index_tp_delta_tot	index value for total density fluctuation
int	index_tp_delta_g	index value for delta of gammas
int	index_tp_delta_b	index value for delta of baryons
int	index_tp_delta_cdm	index value for delta of cold dark matter
int	index_tp_delta_dcdm	index value for delta of DCDM
int	index_tp_delta_fld	index value for delta of dark energy
int	index_tp_delta_scf	index value for delta of scalar field
int	index_tp_delta_dr	index value for delta of decay radiation
int	index_tp_delta_ur	index value for delta of ultra-relativistic neutrinos/relics
int	index_tp_delta_idr	index value for delta of interacting dark radiation
int	index_tp_delta_idm_dr	index value for delta of interacting dark matter (with dr)
int	index_tp_delta_ncdm1	index value for delta of first non-cold dark matter species (e.g. massive neutrinos)
int	index_tp_perturbed_recombination_delta_temp	Gas temperature perturbation
int	index_tp_perturbed_recombination_delta_chi	Ionization fraction perturbation
int	index_tp_theta_m	index value for matter velocity fluctuation
int	index_tp_theta_cb	index value for theta cb
int	index_tp_theta_tot	index value for total velocity fluctuation
int	index_tp_theta_g	index value for theta of gammas
int	index_tp_theta_b	index value for theta of baryons
int	index_tp_theta_cdm	index value for theta of cold dark matter
int	index_tp_theta_dcdm	index value for theta of DCDM
int	index_tp_theta_fld	index value for theta of dark energy

Data Fields

int	index_tp_theta_scf	index value for theta of scalar field
int	index_tp_theta_ur	index value for theta of ultra-relativistic neutrinos/relics
int	index_tp_theta_idr	index value for theta of interacting dark radiation
int	index_tp_theta_idm_dr	index value for theta of interacting dark matter (with dr)
int	index_tp_theta_dr	index value for F1 of decay radiation
int	index_tp_theta_ncdm1	index value for theta of first non-cold dark matter species (e.g. massive neutrinos)
int	index_tp_phi	index value for metric fluctuation phi
int	index_tp_phi_prime	index value for metric fluctuation phi'
int	index_tp_phi_plus_psi	index value for metric fluctuation phi+psi
int	index_tp_psi	index value for metric fluctuation psi
int	index_tp_h	index value for metric fluctuation h
int	index_tp_h_prime	index value for metric fluctuation h'
int	index_tp_eta	index value for metric fluctuation eta
int	index_tp_eta_prime	index value for metric fluctuation eta'
int	index_tp_H_T_Nb_prime	index value for metric fluctuation H_T_Nb'
int	index_tp_k2gamma_Nb	index value for metric fluctuation gamma times k^2 in Nbody gauge
int *	tp_size	number of types tp_size[index_md] included in computation for each mode
int *	k_size_cmb	k_size_cmb[index_md] number of k values used for CMB calculations, requiring a fine sampling in k-space
int *	k_size_cl	k_size_cl[index_md] number of k values used for non-CMB C_l calculations, requiring a coarse sampling in k-space.
int *	k_size	k_size[index_md] = total number of k values, including those needed for P(k) but not for C_l's
double **	k	k[index_md][index_k] = list of values
double	k_min	minimum value (over all modes)
double	k_max	maximum value (over all modes)
double *	tau_sampling	array of tau values
int	tau_size	number of values in this array
double	selection_min_of_tau_min	used in presence of selection functions (for matter density, cosmic shear...)
double	selection_max_of_tau_max	used in presence of selection functions (for matter density, cosmic shear...)
double	selection_delta_tau	used in presence of selection functions (for matter density, cosmic shear...)
double *	selection_tau_min	value of conformal time below which W(tau) is considered to vanish for each bin
double *	selection_tau_max	value of conformal time above which W(tau) is considered to vanish for each bin

Data Fields

double *	selection_tau	value of conformal time at the center of each bin
double *	selection_function	selection function W(tau), normalized to $\int W(tau)dtau = 1$, stored in selection_function[bin*ppt->tau_size+index_tau]
double ***	sources	Pointer towards the source interpolation table sources[index_md] [index_ic * ppt->tp_size[index_md] + index_tp] [index_tau * ppt->k_size + index_k]
double *	ln_tau	log of the arrau tau_sampling, covering only the final time range required for the output of Fourier transfer functions (used for interpolations)
int	ln_tau_size	number of values in this array
double ***	late_sources	Pointer towards the source interpolation table late_sources[index_md] [index_ic * ppt->tp_size[index_md] + index_tp] [index_tau * ppt->k_size + index_k] Note that this is not a replication of part of the sources table, it is just poiting towards the same memory zone, at the place where the late_sources actually start
double ***	ddlate_sources	Pointer towards the splined source interpolation table with second derivatives with respect to time ddlate_sources[index_md] [index_ic * ppt->tp_size[index_md] + index_tp] [index_tau * ppt->k_size + index_k]
int *	index_k_output_values	List of indices corresponding to k-values close to k_output_values for each mode. index_k_output_values[index_md*k_output_values_num+ik]
char	scalar_titles[_MAXTITLESTRINGLENGTH_]	*DELIMITER* separated string of titles for scalar perturbation output files.
char	vector_titles[_MAXTITLESTRINGLENGTH_]	*DELIMITER* separated string of titles for vector perturbation output files.
char	tensor_titles[_MAXTITLESTRINGLENGTH_]	*DELIMITER* separated string of titles for tensor perturbation output files.
int	number_of_scalar_titles	number of titles/columns in scalar perturbation output files
int	number_of_vector_titles	number of titles/columns in vector perturbation output files
int	number_of_tensor_titles	number of titles/columns in tensor perturbation output files
double *	scalar_perturbations_data[_MAX_NUMBER_OF_K_FILES_]	Array of double pointers to perturbation output for scalars
double *	vector_perturbations_data[_MAX_NUMBER_OF_K_FILES_]	Array of double pointers to perturbation output for vectors
double *	tensor_perturbations_data[_MAX_NUMBER_OF_K_FILES_]	Array of double pointers to perturbation output for tensors

Data Fields

int	size_scalar_perturbation_data[_MAX_NUMBER_OF_K_FILES_]	Array of sizes of scalar double pointers
int	size_vector_perturbation_data[_MAX_NUMBER_OF_K_FILES_]	Array of sizes of vector double pointers
int	size_tensor_perturbation_data[_MAX_NUMBER_OF_K_FILES_]	Array of sizes of tensor double pointers
short	perturbations_verbose	flag regulating the amount of information sent to standard output (none if set to zero)
ErrorMsg	error_message	zone for writing error messages

4.20.2.2 struct perturbations_vector

Structure containing the indices and the values of the perturbation variables which are integrated over time (as well as their time-derivatives). For a given wavenumber, the size of these vectors changes when the approximation scheme changes.

Data Fields

int	index_pt_delta_g	photon density
int	index_pt_theta_g	photon velocity
int	index_pt_shear_g	photon shear
int	index_pt_l3_g	photon l=3
int	l_max_g	max momentum in Boltzmann hierarchy (at least 3)
int	index_pt_pol0_g	photon polarization, l=0
int	index_pt_pol1_g	photon polarization, l=1
int	index_pt_pol2_g	photon polarization, l=2
int	index_pt_pol3_g	photon polarization, l=3
int	l_max_pol_g	max momentum in Boltzmann hierarchy (at least 3)
int	index_pt_delta_b	baryon density
int	index_pt_theta_b	baryon velocity
int	index_pt_delta_cdm	cdm density
int	index_pt_theta_cdm	cdm velocity
int	index_pt_delta_idm_dr	idm_dr density
int	index_pt_theta_idm_dr	idm_dr velocity
int	index_pt_delta_dcdm	dcdm density
int	index_pt_theta_dcdm	dcdm velocity
int	index_pt_delta_fld	dark energy density in true fluid case
int	index_pt_theta_fld	dark energy velocity in true fluid case
int	index_pt_Gamma_fld	unique dark energy dynamical variable in PPF case
int	index_pt_phi_scf	scalar field density
int	index_pt_phi_prime_scf	scalar field velocity
int	index_pt_delta_ur	density of ultra-relativistic neutrinos/relics
int	index_pt_theta_ur	velocity of ultra-relativistic neutrinos/relics
int	index_pt_shear_ur	shear of ultra-relativistic neutrinos/relics

Data Fields

int	index_pt_l3_ur	l=3 of ultra-relativistic neutrinos/relics
int	l_max_ur	max momentum in Boltzmann hierarchy (at least 3)
int	index_pt_delta_idr	density of interacting dark radiation
int	index_pt_theta_idr	velocity of interacting dark radiation
int	index_pt_shear_idr	shear of interacting dark radiation
int	index_pt_l3_idr	l=3 of interacting dark radiation
int	l_max_idr	max momentum in Boltzmann hierarchy (at least 3) for interacting dark radiation
int	index_pt_perturbed_recombination_delta_temp	Gas temperature perturbation
int	index_pt_perturbed_recombination_delta_chi	Inionization fraction perturbation
int	index_pt_F0_dr	The index to the first Legendre multipole of the DR expansion. Not that this is not exactly the usual delta, see Kaplinghat et al., astro-ph/9907388.
int	l_max_dr	max momentum in Boltzmann hierarchy for dr)
int	index_pt_psi0_ncdm1	first multipole of perturbation of first ncdm species, Psi_0
int	N_ncdm	number of distinct non-cold-dark-matter (ncdm) species
int *	l_max_ncdm	mutipole l at which Boltzmann hierarchy is truncated (for each ncdm species)
int *	q_size_ncdm	number of discrete momenta (for each ncdm species)
int	index_pt_eta	synchronous gauge metric perturbation eta
int	index_pt_phi	newtonian gauge metric perturbation phi
int	index_pt_hv_prime	vector metric perturbation h_v' in synchronous gauge
int	index_pt_V	vector metric perturbation V in Newtonian gauge
int	index_pt_gw	tensor metric perturbation h (gravitational waves)
int	index_pt_gwdot	its time-derivative
int	pt_size	size of perturbation vector
double *	y	vector of perturbations to be integrated
double *	dy	time-derivative of the same vector
int *	used_in_sources	boolean array specifying which perturbations enter in the calculation of source functions

4.20.2.3 struct perturbations_workspace

Workspace containing, among other things, the value at a given time of all background/perturbed quantities, as well as their indices. There will be one such structure created for each mode (scalar/.../tensor) and each thread (in case of parallel computing)

Data Fields

int	index_mt_psi	psi in longitudinal gauge

Data Fields

int	index_mt_phi_prime	(d phi/d conf.time) in longitudinal gauge
int	index_mt_h_prime	h' (wrt conf. time) in synchronous gauge
int	index_mt_h_prime_prime	h'' (wrt conf. time) in synchronous gauge
int	index_mt_eta_prime	eta' (wrt conf. time) in synchronous gauge
int	index_mt_alpha	$\alpha = (h' + 6\eta')/(2k^2)$ in synchronous gauge
int	index_mt_alpha_prime	α' wrt conf. time) in synchronous gauge
int	index_mt_gw_prime_prime	second derivative wrt conformal time of gravitational wave field, often called h
int	index_mt_V_prime	derivative of Newtonian gauge vector metric perturbation V
int	index_mt_hv_prime_prime	Second derivative of Synchronous gauge vector metric perturbation h_v
int	mt_size	size of metric perturbation vector
double *	pvecback	background quantities
double *	pvecthermo	thermodynamics quantities
double *	pvecmetric	metric quantities
struct perturbations_vector *	pv	pointer to vector of integrated perturbations and their time-derivatives
double	delta_rho	total density perturbation (gives delta Too)
double	rho_plus_p_theta	total (rho+p)*theta perturbation (gives delta Toi)
double	rho_plus_p_shear	total (rho+p)*shear (gives delta Tij)
double	delta_p	total pressure perturbation (gives Tii)
double	rho_plus_p_tot	total (rho+p) (used to infer theta_tot from rho_plus_p_theta)
double	gw_source	stress-energy source term in Einstein's tensor equations (gives Tij[tensor])
double	vector_source_pi	first stress-energy source term in Einstein's vector equations
double	vector_source_v	second stress-energy source term in Einstein's vector equations
double	tca_shear_g	photon shear in tight-coupling approximation
double	tca_slip	photon-baryon slip in tight-coupling approximation
double	tca_shear_idm_dr	interacting dark radiation shear in tight coupling appproximation
double	rsa_delta_g	photon density in radiation streaming approximation
double	rsa_theta_g	photon velocity in radiation streaming approximation
double	rsa_delta_ur	photon density in radiation streaming approximation
double	rsa_theta_ur	photon velocity in radiation streaming approximation
double	rsa_delta_idr	interacting dark radiation density in dark radiation streaming approximation
double	rsa_theta_idr	interacting dark radiation velocity in dark radiation streaming approximation
double *	delta_ncdm	relative density perturbation of each ncdm species

Data Fields

double *	theta_ncdm	velocity divergence theta of each ncdm species
double *	shear_ncdm	shear for each ncdm species
double	delta_m	relative density perturbation of all non-relativistic species
double	theta_m	velocity divergence theta of all non-relativistic species
double	delta_cb	relative density perturbation of only cdm and baryon
double	theta_cb	velocity divergence theta of only cdm and baryon
double	delta_rho_fld	density perturbation of fluid, not so trivial in PPF scheme
double	delta_p_fld	pressure perturbation of fluid, very non-trivial in PPF scheme
double	rho_plus_p_theta_fld	velocity divergence of fluid, not so trivial in PPF scheme
double	S_fld	S quantity sourcing Gamma_prime evolution in PPF scheme (equivalent to eq. 15 in 0808.3125)
double	Gamma_prime_fld	Gamma_prime in PPF scheme (equivalent to eq. 14 in 0808.3125)
FILE *	perturbations_output_file	filepointer to output file
int	index_ikout	index for output k value (when k_output_values is set)
short	inter_mode	flag defining the method used for interpolation background/thermo quantities tables
int	last_index_back	the background interpolation function background_at_tau() keeps memory of the last point called through this index
int	last_index_thermo	the thermodynamics interpolation function thermodynamics_at_z() keeps memory of the last point called through this index
int	index_ap_tca	index for tight-coupling approximation
int	index_ap_rsa	index for radiation streaming approximation
int	index_ap_tca_idm_dr	index for dark tight-coupling approximation (idm-idr)
int	index_ap_rsa_idr	index for dark radiation streaming approximation
int	index_ap_ufa	index for ur fluid approximation
int	index_ap_ncdmfa	index for ncdm fluid approximation
int	ap_size	number of relevant approximations for a given mode
int *	approx	array of approximation flags holding at a given time: approx[index_ap]
int	max_l_max	maximum l_max for any multipole
double *	s_l	array of freestreaming coefficients $s_l = \sqrt{1 - K * (l^2 - 1)/k^2}$

4.20.2.4 struct perturbations_parameters_and_workspace

Structure pointing towards all what the function that perturbations_derivs needs to know: fixed input parameters and indices contained in the various structures, workspace, etc.

Data Fields

struct precision *	ppr	pointer to the precision structure
struct background *	pba	pointer to the background structure
struct thermodynamics *	pth	pointer to the thermodynamics structure
struct perturbations *	ppt	pointer to the precision structure
int	index_md	index of mode (scalar/.../vector/tensor)
int	index_ic	index of initial condition (adiabatic/isocurvature(s)/...)
int	index_k	index of wavenumber
double	k	current value of wavenumber in 1/Mpc
struct perturbations_workspace *	ppw	workspace defined above

4.20.3 Macro Definition Documentation

4.20.3.1 _SELECTION_NUM_MAX_

```
#define _SELECTION_NUM_MAX_ 100
```

maximum number and types of selection function (for bins of matter density or cosmic shear)

4.20.3.2 _MAX_NUMBER_OF_K_FILES_

```
#define _MAX_NUMBER_OF_K_FILES_ 30
```

maximum number of k-values for perturbation output

4.20.4 Enumeration Type Documentation

4.20.4.1 tca_flags

```
enum tca_flags
```

flags for various approximation schemes (tca = tight-coupling approximation, rsa = radiation streaming approximation, ufa = massless neutrinos / ultra-relativistic relics fluid approximation)

CAUTION: must be listed below in chronological order, and cannot be reversible. When integrating equations for a given mode, it is only possible to switch from left to right in the lists below.

4.20.4.2 tca_method

```
enum tca_method
```

labels for the way in which each approximation scheme is implemented

4.20.4.3 possible_gauges

```
enum possible_gauges
```

List of coded gauges. More gauges can in principle be defined.

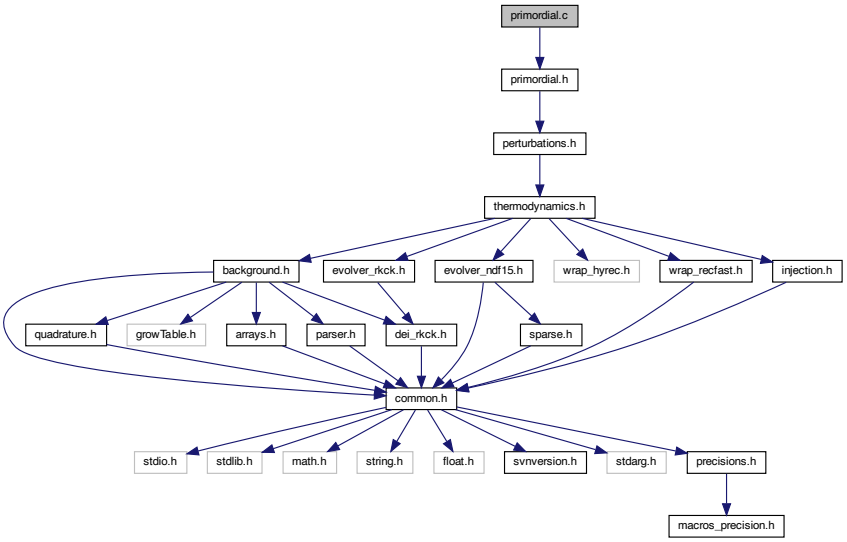
Enumerator

newtonian	newtonian (or longitudinal) gauge
synchronous	synchronous gauge with $\theta_{cdm} = 0$ by convention

4.21 primordial.c File Reference

```
#include "primordial.h"
```

Include dependency graph for primordial.c:



Functions

- int primordial_spectrum_at_k (struct primordial *ppm, int index_md, enum linear_or_logarithmic mode, double input, double *output)
- int primordial_init (struct precision *ppr, struct perturbations *ppt, struct primordial *ppm)
- int primordial_free (struct primordial *ppm)
- int primordial_indices (struct perturbations *ppt, struct primordial *ppm)
- int primordial_get_lnk_list (struct primordial *ppm, double kmin, double kmax, double k_per_decade)
- int primordial_analytic_spectrum_init (struct perturbations *ppt, struct primordial *ppm)
- int primordial_analytic_spectrum (struct primordial *ppm, int index_md, int index_ic1_ic2, double k, double *pk)
- int primordial_inflation_potential (struct primordial *ppm, double phi, double *V, double *dV, double *ddV)
- int primordial_inflation_hubble (struct primordial *ppm, double phi, double *H, double *dH, double *ddH, double *dddH)
- int primordial_inflation_indices (struct primordial *ppm)
- int primordial_inflation_solve_inflation (struct perturbations *ppt, struct primordial *ppm, struct precision *ppr)
- int primordial_inflation_analytic_spectra (struct perturbations *ppt, struct primordial *ppm, struct precision *ppr, double *y_ini)
- int primordial_inflation_spectra (struct perturbations *ppt, struct primordial *ppm, struct precision *ppr, double *y_ini)

- int primordial_inflation_one_wavenumber (struct perturbations *ppt, struct primordial *ppm, struct precision *ppr, double *y_ini, int index_k)
- int primordial_inflation_one_k (struct primordial *ppm, struct precision *ppr, double k, double *y, double *dy, double *curvature, double *tensor)
- int primordial_inflation_find_attractor (struct primordial *ppm, struct precision *ppr, double phi_0, double precision, double *y, double *dy, double *H_0, double *dphidt_0)
- int primordial_inflation_evolve_background (struct primordial *ppm, struct precision *ppr, double *y, double *dy, enum target_quantity target, double stop, short check_epsilon, enum integration_direction direction, enum time_definition time)
- int primordial_inflation_check_potential (struct primordial *ppm, double phi, double *V, double *dV, double *ddV)
- int primordial_inflation_check_hubble (struct primordial *ppm, double phi, double *H, double *dH, double *ddH, double *dddH)
- int primordial_inflation_get_epsilon (struct primordial *ppm, double phi, double *epsilon)
- int primordial_inflation_find_phi_pivot (struct primordial *ppm, struct precision *ppr, double *y, double *dy)
- int primordial_inflation_derivs (double tau, double *y, double *dy, void *parameters_and_workspace, Error↩ Msg error_message)
- int primordial_external_spectrum_init (struct perturbations *ppt, struct primordial *ppm)

4.21.1 Detailed Description

Documented primordial module.

Julien Lesgourgues, 24.08.2010

This module computes the primordial spectra. It can be used in different modes: simple parametric form, evolving inflaton perturbations, etc. So far only the mode corresponding to a simple analytic form in terms of amplitudes, tilts and runnings has been developed.

The following functions can be called from other modules:

1. primordial_init() at the beginning (anytime after perturbations_init() and before harmonic_init())
2. primordial_spectrum_at_k() at any time for computing P(k) at any k
3. primordial_free() at the end

4.21.2 Function Documentation

4.21.2.1 primordial_spectrum_at_k()

```
int primordial_spectrum_at_k (
             struct primordial * ppm,
             int index_md,
             enum linear_or_logarithmic mode,
             double input,
             double * output )
```

Primordial spectra for arbitrary argument and for all initial conditions.

This routine evaluates the primordial spectrum at a given value of k by interpolating in the pre-computed table.

When k is not in the pre-computed range but the spectrum can be found analytically, it finds it. Otherwise returns an error.

Can be called in two modes; linear or logarithmic:

- linear: takes k, returns P(k)
- logarithmic: takes ln(k), return ln(P(k))

One little subtlety: in case of several correlated initial conditions, the cross-correlation spectrum can be negative. Then, in logarithmic mode, the non-diagonal elements contain the cross-correlation angle $P_{12}/\sqrt{P_{11}P_{22}}$ (from -1 to 1) instead of $\ln P_{12}$

This function can be called from whatever module at whatever time, provided that primordial_init() has been called before, and primordial_free() has not been called yet.

Parameters

ppm	Input: pointer to primordial structure containing tabulated primordial spectrum
index_md	Input: index of mode (scalar, tensor, ...)
mode	Input: linear or logarithmic
input	Input: wavenumber in 1/Mpc (linear mode) or its logarithm (logarithmic mode)
output	Output: for each pair of initial conditions, primordial spectra P(k) in Mpc^3 (linear mode), or their logarithms and cross-correlation angles (logarithmic mode)

Returns

the error status

Summary:

- define local variables
- infer ln(k) from input. In linear mode, reject negative value of input k value.
- if ln(k) is not in the interpolation range, return an error, unless we are in the case of a analytic spectrum, for which a direct computation is possible
- otherwise, interpolate in the pre-computed table

4.21.2.2 primordial_init()

```
int primordial_init (
             struct precision * ppr,
             struct perturbations * ppt,
             struct primordial * ppm )
```

This routine initializes the primordial structure (in particular, it computes table of primordial spectrum values)

Parameters

ppr	Input: pointer to precision structure (defines method and precision for all computations)
ppt	Input: pointer to perturbation structure (useful for knowing k_min, k_max, etc.)
ppm	Output: pointer to initialized primordial structure

Returns

the error status

Summary:

- define local variables
- check that we really need to compute the primordial spectra
- get kmin and kmax from perturbation structure. Test that they make sense.
- allocate and fill values of $\ln k$'s
- define indices and allocate tables in primordial structure
- deal with case of analytic primordial spectra (with amplitudes, tilts, runnings, etc.)
- deal with case of inflation with given $V(\phi)$ or $H(\phi)$
- deal with the case of external calculation of P_k
- compute second derivative of each $\ln P_k$ versus lnk with spline, in view of interpolation
- derive spectral parameters from numerically computed spectra (not used by the rest of the code, but useful to keep in memory for several types of investigation)
- expression for alpha_s comes from:

```
ns_2 = (lnpk_plus-lnpk_pivot)/(dlnk)+1

ns_1 = (lnpk_pivot-lnpk_minus)/(dlnk)+1

alpha_s = dns/dlnk = (ns_2-ns_1)/dlnk = (lnpk_plus-lnpk_pivot-lnpk_pivot+lnpk↩
_minus)/(dlnk)/(dlnk)
```

- expression for beta_s:

```
ppm->beta_s = (alpha_plus-alpha_minus)/dlnk = (lnpk_plusplus-2.*lnpk_plus+lnpk↩
_pivot - (lnpk_pivot-2.*lnpk_minus+lnpk_minusminus)/pow(dlnk,3)
```

4.21.2.3 primordial_free()

```
int primordial_free (
            struct primordial * ppm )
```

This routine frees all the memory space allocated by primordial_init().

To be called at the end of each run.

Parameters

ppm	Input: pointer to primordial structure (which fields must be freed)

Returns

the error status

4.21.2.4 primordial_indices()

```
int primordial_indices (
             struct perturbations * ppt,
             struct primordial * ppm )
```

This routine defines indices and allocates tables in the primordial structure

Parameters

ppt	Input: pointer to perturbation structure
ppm	Input/output: pointer to primordial structure

Returns

the error status

4.21.2.5 primordial_get_lnk_list()

```
int primordial_get_lnk_list (
             struct primordial * ppm,
             double kmin,
             double kmax,
             double k_per_decade )
```

This routine allocates and fills the list of wavenumbers k

Parameters

ppm	Input/output: pointer to primordial structure
kmin	Input: first value
kmax	Input: last value that we should encompass
k_per_decade	Input: number of k per decade

Returns

the error status

4.21.2.6 primordial_analytic_spectrum_init()

```
int primordial_analytic_spectrum_init (
             struct perturbations * ppt,
             struct primordial * ppm )
```

This routine interprets and stores in a condensed form the input parameters in the case of a simple analytic spectra with amplitudes, tilts, runnings, in such way that later on, the spectrum can be obtained by a quick call to the routine primordial_analytic_spectrum(()

Parameters

ppt	Input: pointer to perturbation structure
ppm	Input/output: pointer to primordial structure

Returns

the error status

4.21.2.7 primordial_analytic_spectrum()

```
int primordial_analytic_spectrum (
             struct primordial * ppm,
             int index_md,
             int index_ic1_ic2,
             double k,
             double * pk )
```

This routine returns the primordial spectrum in the simple analytic case with amplitudes, tilts, runnings, for each mode (scalar/tensor...), pair of initial conditions, and wavenumber.

Parameters

ppm	Input/output: pointer to primordial structure
index_md	Input: index of mode (scalar, tensor, ...)
index_ic1_ic2	Input: pair of initial conditions (ic1, ic2)
k	Input: wavenumber in same units as pivot scale, i.e. in 1/Mpc
pk	Output: primordial power spectrum A (k/k_pivot)^(n+...)

Returns

the error status

4.21.2.8 primordial_inflation_potential()

```
int primordial_inflation_potential (
             struct primordial * ppm,
```

```
              double phi,
              double * V,
              double * dV,
              double * ddV )
```

This routine encodes the inflaton scalar potential

Parameters

ppm	Input: pointer to primordial structure
phi	Input: background inflaton field value in units of Mp
V	Output: inflaton potential in units of Mp^4
dV	Output: first derivative of inflaton potential wrt the field
ddV	Output: second derivative of inflaton potential wrt the field

Returns

the error status

4.21.2.9 primordial_inflation_hubble()

```
int primordial_inflation_hubble (
              struct primordial * ppm,
              double phi,
              double * H,
              double * dH,
              double * ddH,
              double * dddH )
```

This routine encodes the function $H(\phi)$

Parameters

ppm	Input: pointer to primordial structure
phi	Input: background inflaton field value in units of Mp
H	Output: Hubble parameters in units of Mp
dH	Output: $dH/d\phi$
ddH	Output: $d^2H/d\phi^2$
dddH	Output: $d^3H/d\phi^3$

Returns

the error status

4.21.2.10 primordial_inflation_indices()

```
int primordial_inflation_indices (
             struct primordial * ppm )
```

This routine defines indices used by the inflation simulator

Parameters

ppm	Input/output: pointer to primordial structure

Returns

the error status

4.21.2.11 primordial_inflation_solve_inflation()

```
int primordial_inflation_solve_inflation (
             struct perturbations * ppt,
             struct primordial * ppm,
             struct precision * ppr )
```

Main routine of inflation simulator. Its goal is to check the background evolution before and after the pivot value phi=phi_pivot, and then, if this evolution is suitable, to call the routine primordial_inflation_spectra().

Parameters

ppt	Input: pointer to perturbation structure
ppm	Input/output: pointer to primordial structure
ppr	Input: pointer to precision structure

Returns

the error status

Summary:

- define local variables
- allocate vectors for background/perturbed quantities
- eventually, needs first to find phi_pivot
- compute H_pivot at phi_pivot
- check positivity and negative slope of potential in field pivot value, and find value of phi_dot and H for field's pivot value, assuming slow-roll attractor solution has been reached. If no solution, code will stop there.
- check positivity and negative slope of $H(\phi)$ in field pivot value, and get H_pivot

- find a_pivot, value of scale factor when k_pivot crosses horizon while phi=phi_pivot
- integrate background solution starting from phi_pivot and until k_max>>aH. This ensures that the inflationary model considered here is valid and that the primordial spectrum can be computed. Otherwise, if slow-roll brakes too early, model is not suitable and run stops.
- starting from this time, i.e. from y_ini[ ], we run the routine which takes care of computing the primordial spectrum.
- before ending, we want to compute and store the values of ϕ corresponding to k=aH for k_min and k_max
- finally, we can de-allocate

4.21.2.12 primordial_inflation_analytic_spectra()

```
int primordial_inflation_analytic_spectra (
             struct perturbations * ppt,
             struct primordial * ppm,
             struct precision * ppr,
             double * y_ini )
```

Routine for the computation of an analytic apporoximation to the the primordial spectrum. In general, should be used only for comparing with exact numerical computation performed by primordial_inflation_spectra().

Parameters

ppt	Input: pointer to perturbation structure
ppm	Input/output: pointer to primordial structure
ppr	Input: pointer to precision structure
y_ini	Input: initial conditions for the vector of background/perturbations, already allocated and filled

Returns

the error status

Summary

- allocate vectors for background/perturbed quantities
- initialize the background part of the running vector
- loop over Fourier wavenumbers
- read value of phi at time when k=aH
- get potential (and its derivatives) at this value
- calculate the analytic slow-roll formula for the spectra
- store the obtained result for curvature and tensor perturbations

4.21.2.13 primordial_inflation_spectra()

```
int primordial_inflation_spectra (
             struct perturbations * ppt,
             struct primordial * ppm,
             struct precision * ppr,
             double * y_ini )
```

Routine with a loop over wavenumbers for the computation of the primordial spectrum. For each wavenumber it calls primordial_inflation_one_wavenumber()

Parameters

ppt	Input: pointer to perturbation structure
ppm	Input/output: pointer to primordial structure
ppr	Input: pointer to precision structure
y_ini	Input: initial conditions for the vector of background/perturbations, already allocated and filled

Returns

the error status

4.21.2.14 primordial_inflation_one_wavenumber()

```
int primordial_inflation_one_wavenumber (
             struct perturbations * ppt,
             struct primordial * ppm,
             struct precision * ppr,
             double * y_ini,
             int index_k )
```

Routine coordinating the computation of the primordial spectrum for one wavenumber. It calls primordial_inflation_one_k() to integrate the perturbation equations, and then it stores the result for the scalar/tensor spectra.

Parameters

ppt	Input: pointer to perturbation structure
ppm	Input/output: pointer to primordial structure
ppr	Input: pointer to precision structure
y_ini	Input: initial conditions for the vector of background/perturbations, already allocated and filled
index_k	Input: index of wavenumber to be considered

Returns

the error status

Summary

- allocate vectors for background/perturbed quantities
- initialize the background part of the running vector
- evolve the background until the relevant initial time for integrating perturbations
- evolve the background/perturbation equations from this time and until some time after Horizon crossing
- store the obtained result for curvature and tensor perturbations

4.21.2.15 primordial_inflation_one_k()

```
int primordial_inflation_one_k (
             struct primordial * ppm,
             struct precision * ppr,
             double k,
             double * y,
             double * dy,
             double * curvature,
             double * tensor )
```

Routine integrating the background plus perturbation equations for each wavenumber, and returning the scalar and tensor spectrum.

Parameters

ppm	Input: pointer to primordial structure
ppr	Input: pointer to precision structure
k	Input: Fourier wavenumber
y	Input: running vector of background/perturbations, already allocated and initialized
dy	Input: running vector of background/perturbation derivatives, already allocated
curvature	Output: curvature perturbation
tensor	Output: tensor perturbation

Returns

the error status

Summary:

- define local variables
- initialize the generic integrator (same integrator already used in background, thermodynamics and perturbation modules)
- initialize variable used for deciding when to stop the calculation (= when the curvature remains stable)
- initialize conformal time to arbitrary value (here, only variations of tau matter: the equations that we integrate do not depend explicitly on time)
- compute derivative of initial vector and infer first value of adaptive time-step
- loop over time

- clean the generic integrator
- store final value of curvature for this wavenumber
- store final value of tensor perturbation for this wavenumber

4.21.2.16 primordial_inflation_find_attractor()

```
int primordial_inflation_find_attractor (
             struct primordial * ppm,
             struct precision * ppr,
             double phi_0,
             double precision,
             double * y,
             double * dy,
             double * H_0,
             double * dphidt_0 )
```

Routine searching for the inflationary attractor solution at a given phi_0, by iterations, with a given tolerance. If no solution found within tolerance, returns error message. The principle is the following. The code starts integrating the background equations from various values of phi, corresponding to earlier and earlier value before phi_0, and separated by a small arbitrary step size, corresponding roughly to 1 e-fold of inflation. Each time, the integration starts with the initial condition $\phi = -V'/3H$ (slow-roll prediction). If the found value of ϕ' in phi_0 is stable (up to the parameter "precision"), the code considers that there is an attractor, and stops iterating. If this process does not converge, it returns an error message.

Parameters

ppm	Input: pointer to primordial structure
ppr	Input: pointer to precision structure
phi_0	Input: field value at which we wish to find the solution
precision	Input: tolerance on output values (if too large, an attractor will always considered to be found)
y	Input: running vector of background variables, already allocated and initialized
dy	Input: running vector of background derivatives, already allocated
H_0	Output: Hubble value at phi_0 for attractor solution
dphidt_0	Output: field derivative value at phi_0 for attractor solution

Returns

the error status

4.21.2.17 primordial_inflation_evolve_background()

```
int primordial_inflation_evolve_background (
             struct primordial * ppm,
             struct precision * ppr,
```

```
                double * y,
                double * dy,
                enum target_quantity target,
                double stop,
                short check_epsilon,
                enum integration_direction direction,
                enum time_definition time )
```

Routine integrating background equations only, from initial values stored in y, to a final value (if target = *aH*, until aH = aH_stop; if target = *phi*, till phi = phi_stop; if target = *end_inflation*, until $d^2a/dt^2 = 0$ (here t = proper time)). In output, y contains the final background values. In addition, if check_epsilon is true, the routine controls at each step that the expansion is accelerated and that inflation holds (wepsilon$>$1), otherwise it returns an error. Thanks to the last argument, it is also possible to specify whether the integration should be carried forward or backward in time. For the inflation_H case, only a 1st order differential equation is involved, so the forward and backward case can be done exactly without problems. For the inflation_V case, the equation of motion is 2nd order. What the module will do in the backward case is to search for an approximate solution, corresponding to the (first-order) attractor inflationary solution. This approximate backward solution is used in order to estimate some initial times, but the approximation made here will never impact the final result: the module is written in such a way that after using this approximation, the code always computes (and relies on) the exact forward solution.

Parameters

ppm	Input: pointer to primordial structure
ppr	Input: pointer to precision structure
y	Input/output: running vector of background variables, already allocated and initialized
dy	Input: running vector of background derivatives, already allocated
target	Input: whether the goal is to reach a given aH or ϕ
stop	Input: the target value of either aH or ϕ
check_epsilon	Input: whether we should impose inflation (epsilon$>$1) at each step
direction	Input: whether we should integrate forward or backward in time
time	Input: definition of time (proper or conformal)

Returns

the error status

4.21.2.18 primordial_inflation_check_potential()

```
int primordial_inflation_check_potential (
                struct primordial * ppm,
                double phi,
                double * V,
                double * dV,
                double * ddV )
```

Routine checking positivity and negative slope of potential. The negative slope is an arbitrary choice. Currently the code can only deal with monotonic variations of the inflaton during inflation. So the slope had to be always negative or always positive... we took the first option.

Parameters

ppm	Input: pointer to primordial structure
phi	Input: field value where to perform the check
V	Output: inflaton potential in units of Mp^4
dV	Output: first derivative of inflaton potential wrt the field
ddV	Output: second derivative of inflaton potential wrt the field

Returns

the error status

4.21.2.19 primordial_inflation_check_hubble()

```
int primordial_inflation_check_hubble (
             struct primordial * ppm,
             double phi,
             double * H,
             double * dH,
             double * ddH,
             double * dddH )
```

Routine checking positivity and negative slope of $H(\phi)$. The negative slope is an arbitrary choice. Currently the code can only deal with monotonic variations of the inflaton during inflation. And H can only decrease with time. So the slope $dH/d\phi$ has to be always negative or always positive... we took the first option: phi increases, H decreases.

Parameters

ppm	Input: pointer to primordial structure
phi	Input: field value where to perform the check
H	Output: Hubble parameters in units of Mp
dH	Output: $dH/d\phi$
ddH	Output: $d^2H/d\phi^2$
dddH	Output: $d^3H/d\phi^3$

Returns

the error status

4.21.2.20 primordial_inflation_get_epsilon()

```
int primordial_inflation_get_epsilon (
             struct primordial * ppm,
```

```
            double phi,
            double * epsilon )
```

Routine computing the first slow-roll parameter epsilon

Parameters

ppm	Input: pointer to primordial structure
phi	Input: field value where to compute epsilon
epsilon	Output: result

Returns

the error status

4.21.2.21 primordial_inflation_find_phi_pivot()

```
int primordial_inflation_find_phi_pivot (
              struct primordial * ppm,
              struct precision * ppr,
              double * y,
              double * dy )
```

Routine searching phi_pivot when a given amount of inflation is requested.

Parameters

ppm	Input/output: pointer to primordial structure
ppr	Input: pointer to precision structure
y	Input: running vector of background variables, already allocated and initialized
dy	Input: running vector of background derivatives, already allocated

Returns

the error status

Summary:

- define local variables
- check whether in vicinity of phi_end, inflation is still ongoing
- case in which epsilon>1: hence we must find the value phi_stop < phi_end where inflation ends up naturally
- --> find latest value of the field such that epsilon = primordial_inflation_small_epsilon (default: 0.1)
- --> bracketing right-hand value is phi_end (but the potential will not be evaluated exactly there, only closeby
- --> bracketing left-hand value is found by iterating with logarithmic step until epsilon < primordial_inflation↩ _small_epsilon
- --> find value such that epsilon = primordial_inflation_small_epsilon by bisection
- --> value found and stored as phi_small_epsilon
- --> find inflationary attractor in phi_small_epsilon (should exist since epsilon<<1 there)

- --> compute amount of inflation between this phi_small_epsilon and the end of inflation
- --> by starting from phi_small_epsilon and integrating an approximate solution backward in time, try to estimate roughly a value close to phi_pivot but a bit smaller. This is done by trying to reach an amount of inflation equal to the requested one, minus the amount after phi_small_epsilon, and plus primordial_inflation_extra↩ _efolds efolds (default: two). Note that it is not aggressive to require two extra e-folds of inflation before the pivot, since the calculation of the spectrum in the observable range will require even more.
- --> find attractor in phi_try
- --> check the total amount of inflation between phi_try and the end of inflation
- --> go back to phi_try, and now find phi_pivot such that the amount of inflation between phi_pivot and the end of inflation is exactly the one requested.
- case in which epsilon<1:
- --> find inflationary attractor in phi_small_epsilon (should exist since epsilon<1 there)
- --> by starting from phi_end and integrating an approximate solution backward in time, try to estimate roughly a value close to phi_pivot but a bit smaller. This is done by trying to reach an amount of inflation equal to the requested one, minus the amount after phi_small_epsilon, and plus primordial_inflation_extra_efolds efolds (default: two). Note that it is not aggressive to require two extra e-folds of inflation before the pivot, since the calculation of the spectrum in the observable range will require even more.
- --> we now have a value phi_try believed to be close to and slightly smaller than phi_pivot
- --> find attractor in phi_try
- --> check the total amount of inflation between phi_try and the end of inflation
- --> go back to phi_try, and now find phi_pivot such that the amount of inflation between phi_pivot and the end of inflation is exactly the one requested.
- --> In verbose mode, check that phi_pivot is correct. Done by restarting from phi_pivot and going again till the end of inflation.

4.21.2.22 primordial_inflation_derivs()

```
int primordial_inflation_derivs (
             double tau,
             double * y,
             double * dy,
             void * parameters_and_workspace,
             ErrorMsg error_message )
```

Routine returning derivative of system of background/perturbation variables. Like other routines used by the generic integrator (background_derivs, thermodynamics_derivs, perturbations_derivs), this routine has a generic list of arguments, and a slightly different error management, with the error message returned directly in an ErrMsg field.

Parameters

tau	Input: time (not used explicitly inside the routine, but requested by the generic integrator)
y	Input/output: running vector of background variables, already allocated and initialized
dy	Input: running vector of background derivatives, already allocated
parameters_and_workspace	Input: all necessary input variables apart from y
error_message	Output: error message

Returns

the error status

4.21.2.23 primordial_external_spectrum_init()

```
int primordial_external_spectrum_init (
             struct perturbations * ppt,
             struct primordial * ppm )
```

This routine reads the primordial spectrum from an external command, and stores the tabulated values. The sampling of the k's given by the external command is preserved.

Author: Jesus Torrado (torradocacho@lorentz.leidenuniv.nl) Date: 2013-12-20

Parameters

ppt	Input/output: pointer to perturbation structure
ppm	Input/output: pointer to primordial structure

Returns

the error status

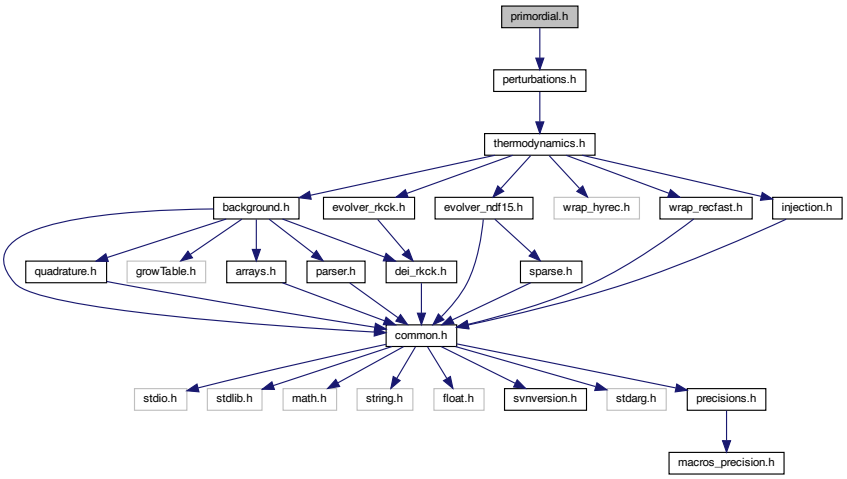
Summary:

- Initialization
- Launch the command and retrieve the output
- Store the read results into CLASS structures
- Make room
- Store values
- Release the memory used locally
- Tell CLASS that there are scalar (and tensor) modes

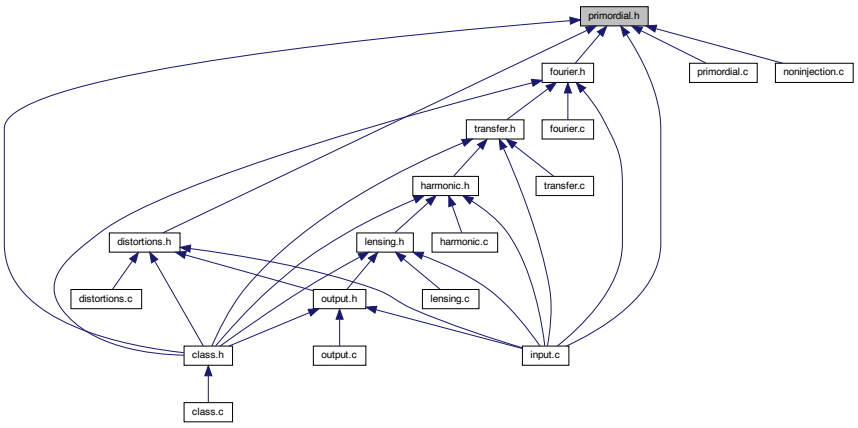
4.22 primordial.h File Reference

```
#include "perturbations.h"
```

Include dependency graph for primordial.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct primordial

Enumerations

- enum primordial_spectrum_type
- enum linear_or_logarithmic
- enum potential_shape
- enum target_quantity
- enum integration_direction
- enum time_definition
- enum phi_pivot_methods
- enum inflation_module_behavior

4.22.1 Detailed Description

Documented includes for primordial module.

4.22.2 Data Structure Documentation

4.22.2.1 struct primordial

Structure containing everything about primordial spectra that other modules need to know.

Once initialized by primordial_init(), contains a table of all primordial spectra as a function of wavenumber, mode, and pair of initial conditions.

Data Fields

double	k_pivot	pivot scale in Mpc^{-1}
int	has_k_max_for_primordial_pk	
double	k_max_for_primordial_pk	maximum value of k in 1/Mpc in P(k)
enum primordial_spectrum_type	primordial_spec_type	type of primordial spectrum (simple analytic from, integration of inflationary perturbations, etc.)
double	A_s	usual scalar amplitude = curvature power spectrum at pivot scale
double	n_s	usual scalar tilt = [curvature power spectrum tilt at pivot scale -1]
double	alpha_s	usual scalar running
double	beta_s	running of running
double	r	usual tensor to scalar ratio of power spectra, $r = A_T/A_S = P_h/P_R$
double	n_t	usual tensor tilt = [GW power spectrum tilt at pivot scale]
double	alpha_t	usual tensor running
double	f_bi	baryon isocurvature (BI) entropy-to-curvature ratio S_{bi}/R
double	n_bi	BI tilt
double	alpha_bi	BI running
double	f_cdi	CDM isocurvature (CDI) entropy-to-curvature ratio S_{cdi}/R
double	n_cdi	CDI tilt
double	alpha_cdi	CDI running
double	f_nid	neutrino density isocurvature (NID) entropy-to-curvature ratio S_{nid}/R
double	n_nid	NID tilt
double	alpha_nid	NID running
double	f_niv	neutrino velocity isocurvature (NIV) entropy-to-curvature ratio S_{niv}/R
double	n_niv	NIV tilt
double	alpha_niv	NIV running
double	c_ad_bi	ADxBI cross-correlation at pivot scale, from -1 to 1

Data Fields

double	n_ad_bi	ADxBI cross-correlation tilt
double	alpha_ad_bi	ADxBI cross-correlation running
double	c_ad_cdi	ADxCDI cross-correlation at pivot scale, from -1 to 1
double	n_ad_cdi	ADxCDI cross-correlation tilt
double	alpha_ad_cdi	ADxCDI cross-correlation running
double	c_ad_nid	ADxNID cross-correlation at pivot scale, from -1 to 1
double	n_ad_nid	ADxNID cross-correlation tilt
double	alpha_ad_nid	ADxNID cross-correlation running
double	c_ad_niv	ADxNIV cross-correlation at pivot scale, from -1 to 1
double	n_ad_niv	ADxNIV cross-correlation tilt
double	alpha_ad_niv	ADxNIV cross-correlation running
double	c_bi_cdi	BIxCDI cross-correlation at pivot scale, from -1 to 1
double	n_bi_cdi	BIxCDI cross-correlation tilt
double	alpha_bi_cdi	BIxCDI cross-correlation running
double	c_bi_nid	BIxNIV cross-correlation at pivot scale, from -1 to 1
double	n_bi_nid	BIxNIV cross-correlation tilt
double	alpha_bi_nid	BIxNIV cross-correlation running
double	c_bi_niv	BIxNIV cross-correlation at pivot scale, from -1 to 1
double	n_bi_niv	BIxNIV cross-correlation tilt
double	alpha_bi_niv	BIxNIV cross-correlation running
double	c_cdi_nid	CDIxNID cross-correlation at pivot scale, from -1 to 1
double	n_cdi_nid	CDIxNID cross-correlation tilt
double	alpha_cdi_nid	CDIxNID cross-correlation running
double	c_cdi_niv	CDIxNIV cross-correlation at pivot scale, from -1 to 1
double	n_cdi_niv	CDIxNIV cross-correlation tilt
double	alpha_cdi_niv	CDIxNIV cross-correlation running
double	c_nid_niv	NIDxNIV cross-correlation at pivot scale, from -1 to 1
double	n_nid_niv	NIDxNIV cross-correlation tilt
double	alpha_nid_niv	NIDxNIV cross-correlation running
enum potential_shape	potential	parameters describing the case primordial_spec_type = inflation_V
double	V0	one parameter of the function V(phi)
double	V1	one parameter of the function V(phi)
double	V2	one parameter of the function V(phi)
double	V3	one parameter of the function V(phi)
double	V4	one parameter of the function V(phi)
double	H0	one parameter of the function H(phi)
double	H1	one parameter of the function H(phi)
double	H2	one parameter of the function H(phi)
double	H3	one parameter of the function H(phi)
double	H4	one parameter of the function H(phi)

Data Fields

double	phi_end	value of inflaton at the end of inflation
enum phi_pivot_methods	phi_pivot_method	flag for method used to define and find the pivot scale
double	phi_pivot_target	For each of the above methods, critical value to be reached between pivot and end of inflation (N_star, [aH]ratio, etc.)
enum inflation_module_behavior	behavior	Specifies if the inflation module computes the primordial spectrum numerically (default) or analytically
char *	command	'external_Pk' mode: command generating the table of Pk and custom parameters to be passed to it string with the command for calling 'external_Pk'
double	custom1	one parameter of the primordial computed in 'external_Pk'
double	custom2	one parameter of the primordial computed in 'external_Pk'
double	custom3	one parameter of the primordial computed in 'external_Pk'
double	custom4	one parameter of the primordial computed in 'external_Pk'
double	custom5	one parameter of the primordial computed in 'external_Pk'
double	custom6	one parameter of the primordial computed in 'external_Pk'
double	custom7	one parameter of the primordial computed in 'external_Pk'
double	custom8	one parameter of the primordial computed in 'external_Pk'
double	custom9	one parameter of the primordial computed in 'external_Pk'
double	custom10	one parameter of the primordial computed in 'external_Pk'
int	md_size	number of modes included in computation
int *	ic_size	for a given mode, ic_size[index_md] = number of initial conditions included in computation
int *	ic_ic_size	number of ordered pairs of (index_ic1, index_ic2); this number is just N(N+1)/2 where N = ic_size[index_md]
int	lnk_size	number of ln(k) values
double *	lnk	list of ln(k) values lnk[index_k]

Data Fields

double **	lnpk	depends on indices index_md, index_ic1, index_ic2, index_k as: lnpk[index_md][index_k*ppm->ic_ic_↩ size[index_md]+index_ic1_ic2] where index_ic1_ic2 labels ordered pairs (index_ic1, index_ic2) (since the primordial spectrum is symmetric in (index_ic1, index_ic2)). • for diagonal elements (index_ic1 = index_ic2) this arrays contains ln[P(k)] where P(k) is positive by construction. • for non-diagonal elements this arrays contains the k-dependent cosine of the correlation angle, namely P(k)_(index_ic1, index_ic2)/sqrt[P(k)_index_ic1 P(k)_index_ic2] This choice is convenient since the sign of the non-diagonal cross-correlation is arbitrary. For fully correlated or anti-correlated initial conditions, this non -diagonal element is independent on k, and equal to +1 or -1.
double **	ddlnpk	second derivative of above array, for spline interpolation. So: • for index_ic1 = index_ic, we spline ln[P(k)] vs. ln(k), which is good since this function is usually smooth. • for non-diagonal coefficients, we spline P(k)_(index_ic1, index_ic2)/sqrt[P(k)_index_ic1 P(k)_index_ic2] vs. ln(k), which is fine since this quantity is often assumed to be constant (e.g for fully correlated/anticorrelated initial conditions) or nearly constant, and with arbitrary sign.
short **	is_non_zero	is_non_zero[index_md][index_ic1_ic2] set to false if pair (index_ic1, index_ic2) is uncorrelated (ensures more precision and saves time with respect to the option of simply setting P(k)_(index_ic1, index_ic2) to zero)
double **	amplitude	all amplitudes in matrix form: amplitude[index_md][index_ic1_ic2]
double **	tilt	all tilts in matrix form: tilt[index_md][index_ic1_ic2]

Data Fields

double **	running	all runnings in matrix form: running[index_md][index_ic1_ic2]
int	index_in_a	scale factor
int	index_in_phi	inflaton vev
int	index_in_dphi	its time derivative
int	index_in_ksi_re	Mukhanov variable (real part)
int	index_in_ksi_im	Mukhanov variable (imaginary part)
int	index_in_dksi_re	Mukhanov variable (real part, time derivative)
int	index_in_dksi_im	Mukhanov variable (imaginary part, time derivative)
int	index_in_ah_re	tensor perturbation (real part)
int	index_in_ah_im	tensor perturbation (imaginary part)
int	index_in_dah_re	tensor perturbation (real part, time derivative)
int	index_in_dah_im	tensor perturbation (imaginary part, time derivative)
int	in_bg_size	size of vector of background quantities only
int	in_size	full size of vector
double	phi_pivot	in inflationary module, value of phi_pivot (set to 0 for inflation_V, inflation_H; found by code for inflation_V_end)
double	phi_min	in inflationary module, value of phi when $k_{min} = aH$
double	phi_max	in inflationary module, value of phi when $k_{max} = aH$
double	phi_stop	in inflationary module, value of phi at the end of inflation
short	primordial_verbose	flag regulating the amount of information sent to standard output (none if set to zero)
ErrorMsg	error_message	zone for writing error messages

4.22.3 Enumeration Type Documentation

4.22.3.1 primordial_spectrum_type

`enum primordial_spectrum_type`

enum defining how the primordial spectrum should be computed

4.22.3.2 linear_or_logarithmic

`enum linear_or_logarithmic`

enum defining whether the spectrum routine works with linear or logarithmic input/output

4.22.3.3 potential_shape

`enum potential_shape`

enum defining the type of inflation potential function V(phi)

4.22.3.4 target_quantity

`enum target_quantity`

enum defining which quantity plays the role of a target for evolving inflationary equations

4.22.3.5 integration_direction

`enum integration_direction`

enum specifying if we want to integrate equations forward or backward in time

4.22.3.6 time_definition

`enum time_definition`

enum specifying if we want to evolve quantities with conformal or proper time

4.22.3.7 phi_pivot_methods

`enum phi_pivot_methods`

enum specifying how, in the inflation_V_end case, the value of phi_pivot should calculated

4.22.3.8 inflation_module_behavior

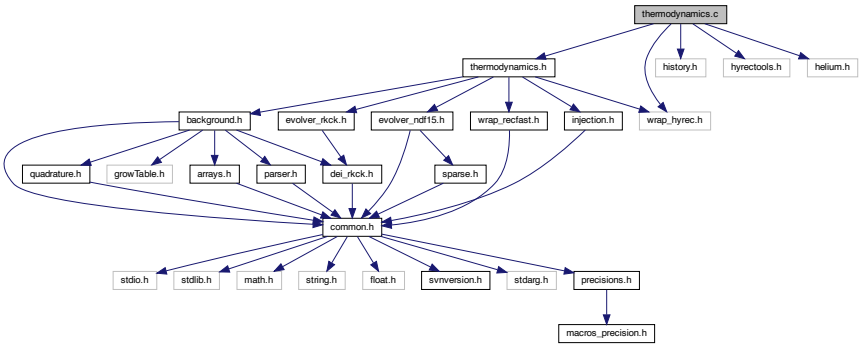
`enum inflation_module_behavior`

enum specifying how the inflation module computes the primordial spectrum (default: numerical)

4.23 thermodynamics.c File Reference

```
#include "thermodynamics.h"
#include "history.h"
#include "hyrectools.h"
#include "helium.h"
#include "wrap_hyrec.h"
```

Include dependency graph for thermodynamics.c:



Functions

- int thermodynamics_at_z (struct background *pba, struct thermodynamics *pth, double z, enum interpolation_method inter_mode, int *last_index, double *pvecback, double *pvecthermo)
- int thermodynamics_init (struct precision *ppr, struct background *pba, struct thermodynamics *pth)
- int thermodynamics_free (struct thermodynamics *pth)
- int thermodynamics_helium_from_bbn (struct precision *ppr, struct background *pba, struct thermodynamics *pth)
- int thermodynamics_checks (struct precision *ppr, struct background *pba, struct thermodynamics *pth)
- int thermodynamics_workspace_init (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct thermo_workspace *ptw)
- int thermodynamics_indices (struct background *pba, struct thermodynamics *pth, struct thermo_workspace *ptw)
- int thermodynamics_lists (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct thermo_workspace *ptw)
- int thermodynamics_set_parameters_reionization (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct thermo_reionization_parameters *preio)
- int thermodynamics_solve (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct thermo_workspace *ptw, double *pvecback)
- int thermodynamics_calculate_remaining_quantities (struct precision *ppr, struct background *pba, struct thermodynamics *pth, double *pvecback)
- int thermodynamics_output_summary (struct background *pba, struct thermodynamics *pth)
- int thermodynamics_workspace_free (struct thermodynamics *pth, struct thermo_workspace *ptw)
- int thermodynamics_vector_init (struct precision *ppr, struct background *pba, struct thermodynamics *pth, double mz, struct thermo_workspace *ptw)
- int thermodynamics_reionization_evolve_with_tau (struct thermodynamics_parameters_and_workspace *ptpaw, double mz_ini, double mz_end, double *mz_output, int mz_size)
- int thermodynamics_derivs (double mz, double *y, double *dy, void *parameters_and_workspace, ErrorMsg error_message)
- int thermodynamics_timescale (double mz, void *thermo_parameters_and_workspace, double *timescale, ErrorMsg error_message)

- int thermodynamics_sources (double mz, double *y, double *dy, int index_z, void *thermo_parameters_↩ and_workspace, ErrorMsg error_message)
- int thermodynamics_reionization_get_tau (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct thermo_workspace *ptw)
- int thermodynamics_vector_free (struct thermo_vector *tv)
- int thermodynamics_calculate_conformal_drag_time (struct background *pba, struct thermodynamics *pth, double *pvecback)
- int thermodynamics_calculate_damping_scale (struct background *pba, struct thermodynamics *pth, double *pvecback)
- int thermodynamics_calculate_opticals (struct precision *ppr, struct thermodynamics *pth)
- int thermodynamics_calculate_idm_dr_quantities (struct precision *ppr, struct background *pba, struct thermodynamics *pth, double *pvecback)
- int thermodynamics_calculate_recombination_quantities (struct precision *ppr, struct background *pba, struct thermodynamics *pth, double *pvecback)
- int thermodynamics_calculate_drag_quantities (struct precision *ppr, struct background *pba, struct thermodynamics *pth, double *pvecback)
- int thermodynamics_ionization_fractions (double z, double *y, struct background *pba, struct thermodynamics *pth, struct thermo_workspace *ptw, int current_ap)
- int thermodynamics_reionization_function (double z, struct thermodynamics *pth, struct thermo_reionization_parameters *preio, double *x)
- int thermodynamics_output_titles (struct background *pba, struct thermodynamics *pth, char titles[_↩ MAXTITLESTRINGLENGTH_])
- int thermodynamics_output_data (struct background *pba, struct thermodynamics *pth, int number_of_titles, double *data)

4.23.1 Detailed Description

Documented thermodynamics module

- Julien Lesgourgues, 6.09.2010
- Restructured by Nils Schoeneberg and Matteo Lucca, 27.02.2019
- Evolver implementation by Daniel Meinert, spring 2019

Deals with the thermodynamical evolution. This module has two purposes:

- at the beginning, to initialize the thermodynamics, i.e. to integrate the thermodynamical equations, and store all thermodynamical quantities as a function of redshift inside an interpolation table.
- to provide a routine which allow other modules to evaluate any thermodynamical quantities at a given redshift value (by interpolating within the interpolation table).

The most important differential equations to compute the free electron fraction x at each step are provided either by the HyRec 2020 or RecFastCLASS code, located in the external/ directory. The thermodynamics module integrates these equations using the generic integrator (which can be set to ndf15, rkck4, etc.) The HyRec and RecFastCLASS algorithms are used and called in the same way by this module.

In summary, the following functions can be called from other modules:

1. thermodynamics_init at the beginning (but after background_init)
2. thermodynamics_at_z at any later time
3. thermodynamics_free at the end, when no more calls to thermodynamics_at_z are needed

4.23.2 Function Documentation

4.23.2.1 thermodynamics_at_z()

```
int thermodynamics_at_z (
             struct background * pba,
             struct thermodynamics * pth,
             double z,
             enum interpolation_method inter_mode,
             int * last_index,
             double * pvecback,
             double * pvecthermo )
```

Thermodynamics quantities at given redshift z. Evaluates all thermodynamics quantities at a given value of the redshift by reading the pre-computed table and interpolating.

Parameters

pba	Input: pointer to background structure
pth	Input: pointer to the thermodynamics structure (containing pre-computed table)
z	Input: redshift
inter_mode	Input: interpolation mode (normal or growing_closeby)
last_index	Input/Output: index of the previous/current point in the interpolation array (input only for closeby mode, output for both)
pvecback	Input: vector of background quantities (used only in case z>z_initial for getting ddkappa and dddkappa; in that case, should be already allocated and filled, with format short_info or larger; in other cases, will be ignored)
pvecthermo	Output: vector of thermodynamics quantities (assumed to be already allocated)

Returns

the error status

Summary:

- define local variables
- interpolate in table with array_interpolate_spline (normal mode) or array_interpolate_spline_growing_closeby (closeby mode)

4.23.2.2 thermodynamics_init()

```
int thermodynamics_init (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth )
```

Initialize the thermodynamics structure, and in particular the thermodynamics interpolation table.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input/Output: pointer to initialized thermodynamics structure

Returns

the error status

Summary:

- define local variables
- set flag for varying constants
- compute and check primordial Helium mass fraction rho_He/(rho_H+rho_He)
- infer primordial helium-to-hydrogen nucleon ratio n_He/n_H It is calculated via n_He/n_H = rho_He/(m_↩ He/m_H $*$ rho_H) = YHe $*$ rho_b / (m_He/m_H $*$ (1-YHe) rho_b) = YHe / (m_He/m_H $*$ (1-YHe))
- infer number of hydrogen nuclei today in m$**$-3
- If there is idm-dr, we want the thermodynamics table to start at a much larger z, in order to capture the possible non-trivial behavior of the dark matter interaction rate at early times
- test whether all parameters are in the correct regime
- allocate and assign all temporary structures and indices
- initialize injection struct (not temporary)
- assign reionisation parameters
- solve recombination and reionization and store values of $z, x_e, d\kappa/d\tau, T_b, c_b^2$
- the differential equation system is now completely solved
- fill missing columns (quantities not computed during the differential evolution but related)
- write information on thermal history in standard output
- free workspace and local variables

4.23.2.3 thermodynamics_free()

```
int thermodynamics_free (
             struct thermodynamics * pth )
```

Free all memory space allocated by thermodynamics_init.

Parameters

pth	Input/Output: pointer to thermodynamics structure (to be freed)

Returns

the error status

4.23.2.4 thermodynamics_helium_from_bbn()

```
int thermodynamics_helium_from_bbn (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth )
```

Infer the primordial helium mass fraction from standard BBN calculations, as a function of the baryon density and expansion rate during BBN.

This module is simpler then the one used in arXiv:0712.2826 because it neglects the impact of a possible significant chemical potentials for electron neutrinos. The full code with xi_nu_e could be introduced here later.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input/Output: pointer to initialized thermodynamics structure

Returns

the error status

Summary:

Define local variables

- Infer effective number of neutrinos at the time of BBN
- We randomly choose 0.1 MeV to be the temperature of BBN
- compute Delta N_eff as defined in bbn file, i.e. $\Delta N_{eff} = 0$ means $N_{eff} = 3.046$. Note that even if 3.044 is a better default value, we must keep 3.046 here as long as the BBN file we are using has been computed assuming 3.046.
- spline in one dimension (along deltaN)
- interpolate in one dimension (along deltaN)
- spline in remaining dimension (along omegab)
- interpolate in remaining dimension (along omegab)
- Take into account impact of varying alpha on helium fraction
- deallocate arrays

4.23.2.5 thermodynamics_checks()

```
int thermodynamics_checks (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth )
```

Check the thermodynamics structure parameters for bounds and critical values.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to initialized thermodynamics structure

Returns

the error status

Summary:

- check BBN Y_He fracion
- tests in order to prevent divisions by zero
- test initial condition for recombination

4.23.2.6 thermodynamics_workspace_init()

```
int thermodynamics_workspace_init (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct thermo_workspace * ptw )
```

Initialize the thermodynamics workspace.

The workspace contains the arrays used for solving differential equations (dubbed thermo_diffeq_workspace), and storing all approximations, reionization parameters, heating parameters.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to the thermodynamics structure
ptw	Input/Output: pointer to thermodynamics workspace

Returns

the error status

Summary:

Define local variables

- number of z values
- relevant cosmological parameters
- relevant constants
- Allocate and initialize differential equation workspace
- define approximations
- store all ending redshifts for each approximation
- Rescale these redshifts in case of varying fundamental constants
- store smoothing deltas for transitions at the beginning of each aproximation
- Allocate reionisation parameter workspace

4.23.2.7 thermodynamics_indices()

```
int thermodynamics_indices (
             struct background * pba,
             struct thermodynamics * pth,
             struct thermo_workspace * ptw )
```

Assign value to each relevant index in vectors of thermodynamical quantities, and the reionization parameters

Parameters

pba	Input: pointer to background structure
pth	Input/Output: pointer to thermodynamics structure
ptw	Input/Output: pointer to thermo workspace

Returns

the error status

Summary:

- define local variables
- initialization of all indices and flags in thermodynamics structure
- initialization of all indices of parameters of reionization function

4.23.2.8 thermodynamics_lists()

```
int thermodynamics_lists (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct thermo_workspace * ptw )
```

Initialize the lists (of redshift, tau, etc.) of the thermodynamics struct

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input/Output: pointer to thermodynamics structure
ptw	Input: pointer to thermo workspace

Returns

the error status

Summary:

Define local variables

- allocate tables
- define time sampling
- store initial value of conformal time in the structure

4.23.2.9 thermodynamics_set_parameters_reionization()

```
int thermodynamics_set_parameters_reionization (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct thermo_reionization_parameters * preio )
```

This routine initializes reionization_parameters for the chosen scheme of reionization function.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to the thermodynamics structure
preio	Input/Output: pointer to the reionization parameters structure

Returns

the error status

Summary:

Define local variables

- allocate the vector of parameters defining the function $X_e(z)$
- (a) no reionization
- (b) if reionization implemented like in CAMB, or half tanh like in 1209.0247
- --> set values of these parameters, excepted those depending on the reionization redshift
- --> if reionization redshift given as an input, initialize the remaining values
- --> if reionization optical depth given as an input, find reionization redshift by bisection and initialize the remaining values
- (c) if reionization implemented with reio_bins_tanh scheme
- (d) if reionization implemented with reio_many_tanh scheme
- (e) if reionization implemented with reio_inter scheme

4.23.2.10 thermodynamics_solve()

```
int thermodynamics_solve (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct thermo_workspace * ptw,
             double * pvecback )
```

Integrate thermodynamics with your favorite recombination code. The default options are HyRec and RecFast↩ CLASS.

Integrate thermodynamics with HyRec or Recfast, allocate and fill part of the thermodynamics interpolation table (the rest is filled in thermodynamics_calculate_remaining_quantitie).

Version modified by Daniel Meinert and Nils Schoeneberg to use the ndf15 evolver or any other evolver inherent to CLASS, modified again by Nils Schoeneberg to use wrappers.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input/Output: pointer to thermodynamics structure where results are stored
ptw	Input: pointer to thermo_workspace structure used to communicate with generic evolver
pvecback	Input: pointer to an allocated (but empty) vector of background variables

Returns

the error status

Integrate thermodynamics with your favorite recombination code. The default options are HyRec and Recfast. Summary:

- define local variables
- choose evolver
- define the fields of the 'thermodynamics parameter and workspace' structure
- define time sampling: create a local array of minus z values called mz (from mz=-zinitial growing towards mz=0)
- define intervals for each approximation scheme
- loop over intervals over which approximation scheme is uniform. For each interval:
- --> (a) fix current approximation scheme.
- --> (b) define the vector of quantities to be integrated over. If the current interval starts from the initial time zinitial, fill the vector with initial conditions. If it starts from an approximation switching point, redistribute correctly the values from the previous to the new vector. For both RECFAST and HYREC, the vector consists of Tmat, x_H, x_He, + others for exotic models
- --> (c1) If we have the optical depth tau_reio as input the last evolver step (reionization approximation) is done separately in a loop, to find the approximate redshift of reionization given the input of tau_reio, using a bisection method. This is similar to the general CLASS shooting method, but doing this step here is more davantageous since we only need to do repeatedly the last approximation step of the integration, instead of the full background and thermodynamics module

--> (c2) otherwise, just integrate quantities over the current interval.

- Compute reionization optical depth, if not supplied as input parameter
- free quantities allocated at the beginning of the routine

4.23.2.11 thermodynamics_calculate_remaining_quantities()

```
int thermodynamics_calculate_remaining_quantities (
              struct precision * ppr,
              struct background * pba,
              struct thermodynamics * pth,
              double * pvecback )
```

Calculate those thermodynamics quantities which are not inside of the thermodynamics table already.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input/Output: pointer to initialized thermodynamics structure
pvecback	Input: pointer to some allocated pvecback

Returns

the error status

Summary:

- fill tables of second derivatives with respect to z (in view of spline interpolation)

4.23.2.12 thermodynamics_output_summary()

```
int thermodynamics_output_summary (
             struct background * pba,
             struct thermodynamics * pth )
```

In verbose mode, print basic information on the thermal history

Parameters

pba	Input: pointer to background structure
pth	Input/Output: pointer to initialized thermodynamics structure

Returns

the error status

Summary:

Define local variables

- print the main results

4.23.2.13 thermodynamics_workspace_free()

```
int thermodynamics_workspace_free (
             struct thermodynamics * pth,
             struct thermo_workspace * ptw )
```

Free the thermo_workspace structure (with the exception of the thermo_vector '->ptv' field, which is freed separately in thermo_vector_free).

Parameters

pth	Input: pointer to initialized thermodynamics structure
ptw	Input: pointer to perturbations_workspace structure to be freed

Returns

the error status

4.23.2.14 thermodynamics_vector_init()

```
int thermodynamics_vector_init (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             double mz,
             struct thermo_workspace * ptw )
```

Initialize the field '->ptv' of a thermo_diffeq_workspace structure, which is a thermo_vector structure. This structure contains indices and values of all quantities which need to be integrated with respect to time (and only them: quantities fixed analytically or obeying constraint equations are NOT included in this vector).

The routine sets and allocates the vector y, dy and used_in_output with the right size depending on the current approximation scheme stored in the workspace. Moreover the initial conditions for each approximation scheme are calculated and set correctly.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to the thermodynamics structure
mz	Input: negative redshift
ptw	Input/Output: pointer to thermodynamics workspace

Returns

the error status

Summary:

Define local variables

4.23.2.15 thermodynamics_reionization_evolve_with_tau()

```
int thermodynamics_reionization_evolve_with_tau (
             struct thermodynamics_parameters_and_workspace * ptpaw,
             double mz_ini,
             double mz_end,
             double * mz_output,
             int mz_size )
```

If the input for reionization is tau_reio, thermodynamics_solve() calls this function instead of the evolver for dealing with the last era (the reionization era).

Instead of computing the evolution of quantities during reionization for a fixed z_reio, as the evolver would do, this function finds z_reio by bisection. First we make an initial guess for z_reio with reionization_z_start_max and then find a z_reio which leads to the given tau_reio (in the range of tolerance reionization_optical_depth_tol).

Parameters

ptpaw	Input: pointer to parameters and workspace
mz_ini	Input: initial redshift
mz_end	Input: ending redshift
mz_output	Input: pointer to redshift array at which output should be written
mz_size	Input: number of redshift values in this array

Returns

the error status

Summary:

Define local variables

- Remame fields to avoid heavy notations
- Choose evolver
- ptvs will be a pointer towards the same thermo vector that was used in the previous approximation schemes; it contains values that will serve here to set initial conditions.
- ptv is a pointer towards a whole new thermo vector used for the calculations in the bisection, that we must allocate and initialize
- Initialize the values of the temporary vector
- Evolve quantities through reionization assuming upper value of z_reio
- Restore initial conditions
- Evolve quantities through reionization assuming lower value of z_reio
- Restore initial conditions
- Evolve quantities through reionization, trying intermediate values of z_reio by bisection
- Store the ionization redshift in the thermodynamics structure
- Free tempeoraty thermo vector

4.23.2.16 thermodynamics_derivs()

```
int thermodynamics_derivs (
             double mz,
             double * y,
             double * dy,
             void * parameters_and_workspace,
             ErrorMsg error_message )
```

Subroutine evaluating the derivative of thermodynamical quantities with respect to negative redshift mz=-z.

Automatically recognizes the current approximation interval and computes the derivatives for this interval of the vector y, which contains (Tmat, x_H, x_He) + others for exotic models.

Derivatives are obtained either by calling either HyRec 2020 (Lee and Ali-Haimoud 2020, 2007.14114) or Rec↩ FastCLASS (that is, RecFast version 1.5, modified by Daniel Meinert and Nils Schoeneberg for better precision and smoothness at early times). See credits and licences in the wrappers (in external/...)

This is one of the few functions in the code which are passed to the generic_evolver routine. Since generic_evolver should work with functions passed from various modules, the format of the arguments is a bit special:

- fixed parameters and workspaces are passed through a generic pointer. Here, this pointer contains the precision, background and thermodynamics structures, plus a background vector, but generic_evolver doesn't know its precise structure.
- the error management is a bit special: errors are not written as usual to pth->error_message, but to a generic error_message passed in the list of arguments.

Parameters

mz	Input: negative redshift mz = -z
y	Input: vector of variable to integrate
dy	Output: its derivative (already allocated)
parameters_and_workspace	Input: pointer to fixed parameters (e.g. indices) and workspace (already allocated)
error_message	Output: error message

Summary:

Define local variables

- Rename structure fields (just to avoid heavy notations)
- Get background/thermo quantities in this point

Set Tmat from the evolver (it is always evolved) and store it in the workspace.

- The input vector y contains thermodynamic variables like (Tmat, x_H,x_He). The goal of this function is: 1) Depending on the chosen code and current approximation, to use either analytical approximations or the vector y to calculate x_e; 2) To compute re-ionization effects on x_e; The output of this function is stored in the workspace ptdw
- If needed, calculate heating effects (i.e. any possible energy deposition rates affecting the evolution equations for x and Tmat)
- Derivative of the ionization fractions

--> use Recfast or HyRec to get the derivatives d(x_H)/dz and d(x_He)/dz, and store the result directly in the vector dy. This gives the derivative of the ionization fractions from recombination only (not from reionization). Of course, the full treatment would involve the actual evolution equations for x_H and x_He during reionization, but these are not yet fully implemented.

< Pass value of fsR = relative alpha (fine-structure)

< Pass value of meR = relative m_e (effective eletron mass)

- Derivative of the matter temperature (relevant for both Recfast and HyRec cases)
- If we have extreme heatings, recombination does not fully happen and/or re-ionization happens before a redshift of reionization_z_start_max (default = 50). We want to catch this unphysical regime, because it would lead to further errors (and/or unphysical calculations) within our recombination codes
- invert all derivatives (because the evolver evolves with -z, not with +z)

4.23.2.17 thermodynamics_timescale()

```
int thermodynamics_timescale (
              double mz,
              void * thermo_parameters_and_workspace,
              double * timescale,
              ErrorMsg error_message )
```

This function is relevant for the rk evolver, not ndf15. It estimates a timescale 'delta z' over which quantitites vary. The rk evolver divides large intervals in steps given by this timescale multiplied by ppr->thermo_integration_stepsize.

This is one of the few functions in the code which is passed to the generic_evolver routine. Since generic_evolver should work with functions passed from various modules, the format of the arguments is a bit special:

- fixed parameters and workspaces are passed through a generic pointer. generic_evolver doesn't know the content of this pointer.
- the error management is a bit special: errors are not written as usual to pth->error_message, but to a generic error_message passed in the list of arguments.

Parameters

mz	Input: minus the redshift
thermo_parameters_and_workspace	Input: pointer to parameters and workspace
timescale	Output: pointer to the timescale
error_message	Output: possible errors are written here

Returns

the error status

4.23.2.18 thermodynamics_sources()

```
int thermodynamics_sources (
              double mz,
              double * y,
              double * dy,
              int index_z,
              void * thermo_parameters_and_workspace,
              ErrorMsg error_message )
```

This function is passed to the generic evolver and is called whenever we want to store values for a given mz.

The ionization fraction is either computed within a call to thermodynamics_derivs(). Moreover there is an automatic smoothing enabled which smoothes out the the ionization_fraction after each approximation switch. This is also the place where HyRec is asked to evolve x(z) using its internal system of differential equations over the next range [z_i, z_i+1], and to store the result in a temporary table.

This is one of the few functions in the code which is passed to the generic_evolver routine. Since generic_evolver should work with functions passed from various modules, the format of the arguments is a bit special:

- fixed parameters and workspaces are passed through a generic pointer. generic_evolver doesn't know the content of this pointer.
- the error management is a bit special: errors are not written as usual to pth->error_message, but to a generic error_message passed in the list of arguments.

All quantities are computed by a simple call to thermodynamics_derivs, which computes all necessary quantities and stores them in the ptdw thermo_diffeq_workspace structure

Parameters

mz	Input: negative redshift, belonging to array mz_output
y	Input: vector of evolved thermodynamical quantities
dy	Input: derivatives of this vector w.r.t redshift
index_z	Input: index in the array mz_output
thermo_parameters_and_workspace	Input/Output: in input, all parameters needed by thermodynamics_derivs; in output, recombination table
error_message	Output: error message

Returns

the error status

Summary:

Define local variables

- Rename structure fields (just to avoid heavy notations)
- Recalculate all quantities at this current redshift: we need at least pvecback, ptdw->x_reio, dy[ptv->index_ti_D_Tmat]
- In the recfast case, we manually smooth the results a bit
- Store the results in the table. Results are obtained in order of decreasing z, and stored in order of growing z

4.23.2.19 thermodynamics_reionization_get_tau()

```
int thermodynamics_reionization_get_tau (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct thermo_workspace * ptw )
```

Get the optical depth of reionization tau_reio for a given thermodynamical history.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to the thermodynamics structure
ptw	Input: pointer to thermodynamics workspace

Returns

the error status

Summary:

Define local variables

We are searching now for the start of reionization. This will be the time at which the optical depth tau_reio will be computed.

Note that the value reionization_parameters[index_reio_start] is only the start of the reionization function added manually, but not necessarily the total start of reionization. Reionization could be longer/shifted by energy injection.

The actual the definition of tau_reio is not unique and unambiguous. We defined it here to be the optical depth up to the time at which there is a global minimum in the free electron fraction. We search for this time by iterating over the thermodynamics table, in order to find the corresponding index_reio_start.

- --> spline $d\tau/dz$ with respect to z in view of integrating for optical depth between 0 and the just found starting index
- --> integrate for optical depth

4.23.2.20 thermodynamics_vector_free()

```
int thermodynamics_vector_free (
             struct thermo_vector * tv )
```

Free the thermo_vector structure, which is the '->ptv' field of the thermodynamics_differential_workspace ptdw structure

Parameters

tv	Input: pointer to thermo_vector structure to be freed

Returns

the error status

4.23.2.21 thermodynamics_calculate_conformal_drag_time()

```
int thermodynamics_calculate_conformal_drag_time (
             struct background * pba,
             struct thermodynamics * pth,
             double * pvecback )
```

Compute the baryon drag conformal time tau_d = [int_{tau_today}^{tau} dtau -dkappa_d/dtau]

Parameters

pba	Input: pointer to background structure
pth	Input/Output: pointer to initialized thermodynamics structure
pvecback	Input: Initialized vector of background quantities

Returns

the error status

Summary:

Define local variables

- compute minus the baryon drag interaction rate time, -dkappa_d/dtau = -[1/R ∗ kappa'], with R = 3 rho_b / 4 rho_gamma, stored temporarily in column ddkappa
- compute second derivative of this rate, -[1/R ∗ kappa']'', stored temporarily in column dddkappa
- compute tau_d = [int_{tau_today}^{tau} dtau -dkappa_d/dtau]

4.23.2.22 thermodynamics_calculate_damping_scale()

```
int thermodynamics_calculate_damping_scale (
             struct background * pba,
             struct thermodynamics * pth,
             double * pvecback )
```

Compute the damping scale r_d = 2pi/k_d = 2pi ∗ [int_{tau_ini}^{tau} dtau (1/kappa') 1/6 (R^2+16/15(1+R))/(1+R)^2]^1/2 = 2pi ∗ [int_{tau_ini}^{tau} dtau (1/kappa') 1/6 (R^2/(1+R)+16/15)/(1+R)]^1/2

which is like in CosmoTherm (CT), but slightly different from Wayne Hu (WH)'s thesis eq. (5.59): The factor 16/15 in CT is 4/5 in WH, but 16/15 is taking more effects into account

Parameters

pba	Input: pointer to background structure
pth	Input/Output: pointer to initialized thermodynamics structure
pvecback	Input: Initialized vector of background quantities

Returns

the error status

Summary:

Define local variables

4.23.2.23 thermodynamics_calculate_opticals()

```
int thermodynamics_calculate_opticals (
             struct precision * ppr,
             struct thermodynamics * pth )
```

Calculate quantities relating to optical phenomena like kappa' and exp(-kappa) and the visibility function, optical depth, etc.

Parameters

ppr	Input: pointer to precision structure
pth	Input/Output: pointer to thermodynamics structure

Returns

the error status

Summary:

Define local quantities

- --> second derivative with respect to tau of dkappa (in view of spline interpolation)
- --> first derivative with respect to tau of dkappa (using spline interpolation)
- --> compute -kappa = [int_{tau_today}^{tau} dtau dkappa/dtau], store temporarily in column "g"
- --> compute visibility: $g = (d\kappa/d\tau)e^{-\kappa}$
- —> compute g
- —> compute exp(-kappa)
- —> compute g' (the plus sign of the second term is correct, see def of -kappa in thermodynamics module!)
- —> compute g"
- —> store g
- —> compute variation rate
- smooth the rate (details of smoothing unimportant: only the order of magnitude of the rate matters)
- --> derivatives of baryon sound speed (only computed if some non-minimal tight-coupling schemes is requested)
- —> second derivative with respect to tau of cb2
- —> first derivative with respect to tau of cb2 (using spline interpolation)

4.23.2.24 thermodynamics_calculate_idm_dr_quantities()

```
int thermodynamics_calculate_idm_dr_quantities (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             double * pvecback )
```

Compute the idm_dr quantities: idm-dr opacities, idr self-interaction rate, dark optical depths, etc.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input/Output: pointer to initialized thermodynamics structure
pvecback	Input: Initialized vector of background quantities

Returns

the error status

Summary:

- Define local variables
- second derivative of idm_dr interaction rate (with idr), [Sinv∗dmu_idm_dr]", stored temporarily in column dddmu
- compute optical depth of idm, tau_idm_dr = [int_{tau_today}^{tau} dtau [Sinv∗dmu_idm_dr]]. This step gives -tau_idm_dr. The resulty is mutiplied by -1 later on.
- second derivative of idr interaction rate (with idm_dr), [dmu_idm_idr]", stored temporarily in column dddmu
- compute optical depth of idr, tau_idr = [int_{tau_today}^{tau} dtau [dmu_idm_idr]]. This step gives -tau_idr. The resulty is mutiplied by -1 later on.
- --> second derivative with respect to tau of dmu_idm_dr (in view of spline interpolation)
- --> first derivative with respect to tau of dmu_idm_dr (using spline interpolation)
- --> now compute idm_dr temperature and sound speed in various regimes
- Find interacting dark radiation free-streaming time
- find idm_dr and idr drag times

4.23.2.25 thermodynamics_calculate_recombination_quantities()

```
int thermodynamics_calculate_recombination_quantities (
              struct precision * ppr,
              struct background * pba,
              struct thermodynamics * pth,
              double * pvecback )
```

Calculate various quantities at the time of recombination, as well as the time tau_cut at which visibility gets negligible and one can assume pure free-streaming.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input/Output: pointer to initialized thermodynamics structure
pvecback	Input: pointer to some allocated pvecback

Returns

the error status

Summary:

Define local variables

- find maximum of g
- find conformal recombination time using background_tau_of_z
- find damping scale at recombination (using linear interpolation)
- find time (always after recombination) at which tau_c/tau falls below some threshold, defining tau_free_↩ streaming
- find time above which visibility falls below a given fraction of its maximum
- find z_star (when optical depth kappa crosses one, using linear interpolation) and sound horizon at that time

4.23.2.26 thermodynamics_calculate_drag_quantities()

```
int thermodynamics_calculate_drag_quantities (
              struct precision * ppr,
              struct background * pba,
              struct thermodynamics * pth,
              double * pvecback )
```

Calculate various quantities at the time of ending of baryon drag (It is precisely where tau_d crosses one)

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input/Output: pointer to initialized thermodynamics structure
pvecback	Input: pointer to some allocated pvecback

Returns

the error status

Summary:

Define local variables

- find baryon drag time (when tau_d crosses one, using linear interpolation) and sound horizon at that time

4.23.2.27 thermodynamics_ionization_fractions()

```
int thermodynamics_ionization_fractions (
              double z,
              double * y,
              struct background * pba,
              struct thermodynamics * pth,
              struct thermo_workspace * ptw,
              int current_ap )
```

Compute ionization fractions with the RecFast of HyRec algorithm.

Compute ionization fractions using either the vector y or, in some approximation schemes, some analytic approximations. The output of this function is located in the ptw->ptdw workspace. We need to assign:

- in the RecFast scheme only: – ptdw->x_H, ptdw-> x_He (all neglecting reionisation, which is accounted for later on);
- in both schemes: – ptdw->x_noreio (neglecting reionisation); – ptdw->x_reio (if reionisation is going on; obtained by calling thermodynamics_reionization_function() and adding something to ptdw->x_noreio)

Parameters

z	Input: redshift
y	Input: vector of quantities to integrate with evolver
pth	Input: pointer to thermodynamics structure
pba	Input: pointer to background structure
ptw	Input/Output: pointer to thermo workspace. Contains output for x, ...
current_ap	Input: index of current approximation scheme

Returns

the error status

Summary:

Define local variables

- Calculate x_noreio from Recfast/Hyrec in each approximation regime. Store the results in the workspace.
- --> For credits, see external/wrap_recfast.c
- --> first regime: H and Helium fully ionized
- --> second regime: first Helium recombination (analytic approximation)
- --> third regime: first Helium recombination finished, H and Helium fully ionized
- --> fourth regime: second Helium recombination starts (analytic approximation)
- --> fifth regime: Hydrogen recombination starts (analytic approximation) while Helium recombination continues (full equation)
- --> sixth regime: full Hydrogen and Helium equations
- --> seventh regime: calculate x_noreio during reionization (i.e. x before taking reionisation into account)
- If z is during reionization, also calculate the reionized x

4.23.2.28 thermodynamics_reionization_function()

```
int thermodynamics_reionization_function (
             double z,
             struct thermodynamics * pth,
             struct thermo_reionization_parameters * preio,
             double * x )
```

This subroutine contains the reionization function $X_e(z)$ (one for each scheme) and gives x for a given z.

Parameters

z	Input: redshift
pth	Input: pointer to thermodynamics structure, to know which scheme is used
preio	Input: pointer to reionization parameters of the function $X_e(z)$
x	Output: $X_e(z)$

Summary:

- define local variables
- no reionization means nothing to be added to xe_before
- implementation of ionization function similar to the one in CAMB
- --> case z $>$ z_reio_start
- --> case z $<$ z_reio_start: hydrogen contribution (tanh of complicated argument)
- --> case z $<$ z_reio_start: helium contribution (tanh of simpler argument)
- implementation of half-tangent like in 1209.0247
- --> case z $>$ z_reio_start
- --> case z $<$ z_reio_start: hydrogen contribution (tanh of complicated argument)
- implementation of binned ionization function similar to astro-ph/0606552
- --> case z $>$ z_reio_start
- implementation of many tanh jumps
- --> case z $>$ z_reio_start
- implementation of reio_inter
- --> case z $>$ z_reio_start

4.23.2.29 thermodynamics_output_titles()

```
int thermodynamics_output_titles (
             struct background * pba,
             struct thermodynamics * pth,
             char titles[_MAXTITLESTRINGLENGTH_] )
```

Function for formatting the titles to be output

Parameters

pba	Input: pointer to background structure
pth	Input: pointer to the thermodynamics structure
titles	Input: titles string containing all titles

Returns

the error status

4.23.2.30 thermodynamics_output_data()

```
int thermodynamics_output_data (
             struct background * pba,
             struct thermodynamics * pth,
             int number_of_titles,
             double * data )
```

Output the data for the output into files

Parameters

pba	Input: pointer to background structure
pth	Input: pointer to the thermodynamics structure
number_of_titles	Input: number of titles
data	Input: pointer to data file

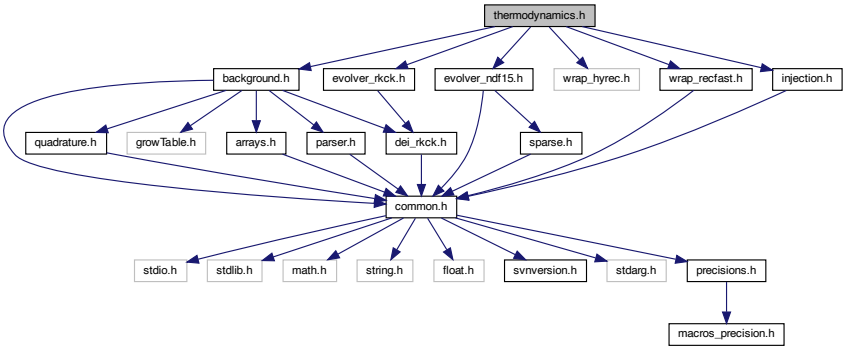
Returns

the error status

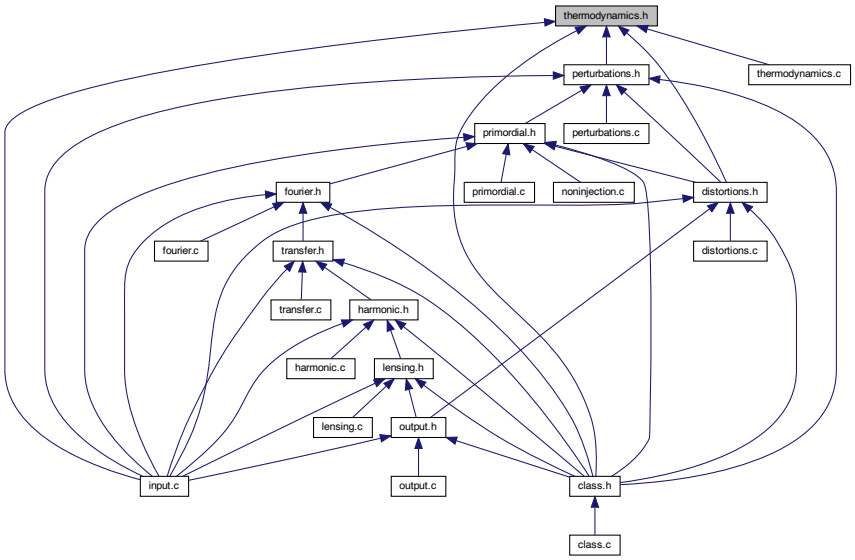
4.24 thermodynamics.h File Reference

```
#include "background.h"
#include "evolver_ndf15.h"
#include "evolver_rkck.h"
#include "wrap_hyrec.h"
#include "wrap_recfast.h"
#include "injection.h"
```

Include dependency graph for thermodynamics.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct thermodynamics
- struct thermo_vector
- struct thermo_diffeq_workspace
- struct thermo_reionization_parameters
- struct thermo_workspace
- struct thermodynamics_parameters_and_workspace

Macros

- #define f1(x) (-0.75∗x∗(x∗x/3.-1.)+0.5)
- #define f2(x) (x∗x∗(0.5-x/3.)∗6.)

Some limits imposed on cosmological parameter values:

- #define _YHE_BIG_ 0.5
- #define _YHE_SMALL_ 0.01
- #define **_Z_REC_MAX_** 2000.
- #define **_Z_REC_MIN_** 500.

Enumerations

- enum recombination_algorithm
- enum reionization_parametrization { reio_none , reio_camb , reio_bins_tanh , reio_half_tanh , reio_many_tanh , reio_inter }
- enum reionization_z_or_tau { reio_z , reio_tau }

4.24.1 Detailed Description

Documented includes for thermodynamics module

4.24.2 Data Structure Documentation

4.24.2.1 struct thermodynamics

All thermodynamics parameters and evolution that other modules need to know.

Once initialized by thermodynamics_init(), contains all the necessary information on the thermodynamics, and in particular, a table of thermodynamical quantities as a function of the redshift, used for interpolation in other modules.

Data Fields

double	YHe	Y_{He}: primordial helium mass fraction rho_He/(rho_H+rho_He), close but not exactly equal to the density fraction 4∗n_He/(n_H+4∗n_He)
double	bbn_alpha_sensitivity	Related to variation of fundamental constants (sensitivity of YHe to alpha)
enum recombination_algorithm	recombination	recombination code
enum recfast_photoion_modes	recfast_photoion_mode	photo-ionization coefficient mode of the recfast algorithm
enum reionization_parametrization	reio_parametrization	reionization scheme
enum reionization_z_or_tau	reio_z_or_tau	is the input parameter the reionization redshift or optical depth?
double	tau_reio	if above set to tau, input value of reionization optical depth
double	z_reio	if above set to z, input value of reionization redshift
short	compute_cb2_derivatives	do we want to include in computation derivatives of baryon sound speed?
short	compute_damping_scale	do we want to compute the simplest analytic approximation to the photon damping (or diffusion) scale?

Data Fields

double	reionization_width	parameters for reio_camb width of H reionization
double	reionization_exponent	shape of H reionization
double	helium_fullreio_redshift	redshift for of helium reionization
double	helium_fullreio_width	width of helium reionization
int	binned_reio_num	parameters for reio_bins_tanh with how many bins do we want to describe reionization?
double *	binned_reio_z	central z value for each bin
double *	binned_reio_xe	imposed $X_e(z)$ value at center of each bin
double	binned_reio_step_sharpness	sharpness of tanh() step interpolating between binned values
int	many_tanh_num	parameters for reio_many_tanh with how many jumps do we want to describe reionization?
double *	many_tanh_z	central z value for each tanh jump
double *	many_tanh_xe	imposed $X_e(z)$ value at the end of each jump (ie at later times)
double	many_tanh_width	sharpness of tanh() steps
int	reio_inter_num	parameters for reio_inter with how many jumps do we want to describe reionization?
double *	reio_inter_z	discrete z values
double *	reio_inter_xe	discrete $X_e(z)$ values
short	has_exotic_injection	parameters for energy injection true if some exotic mechanism injects energy and affects the evolution of ionization and/or temperature and/or other thermodynamics variables that are relevant for the calculation of CMB anisotropies (and spectral distorsions if requested).
struct injection	in	structure to store exotic energy injections and their energy deposition
double	annihilation	parameter describing CDM annihilation (f <sigma*v> / m_cdm, see e.g. 0905.0003)
short	has_on_the_spot	flag to specify if we want to use the on-the-spot approximation
double	decay	parameter describing CDM decay (f/tau, see e.g. 1109.6322)
double	annihilation_variation	if this parameter is non-zero, the function F(z)=(f <sigma*v> / m_cdm)(z) will be a parabola in log-log scale between zmin and zmax, with a curvature given by annihlation_variation (must be negative), and with a maximum in zmax; it will be constant outside this range

Data Fields

double	annihilation_z	if annihilation_variation is non-zero, this is the value of z at which the parameter annihilation is defined, i.e. F(annihilation_z)=annihilation
double	annihilation_zmax	if annihilation_variation is non-zero, redshift above which annihilation rate is maximal
double	annihilation_zmin	if annihilation_variation is non-zero, redshift below which annihilation rate is constant
double	annihilation_f_halo	takes the contribution of DM annihilation in halos into account
double	annihilation_z_halo	characteristic redshift for DM annihilation in halos
double	a_idm_dr	strength of the coupling between interacting dark matter and interacting dark radiation (idm-idr)
double	b_idr	strength of the self coupling for interacting dark radiation (idr-idr)
double	nindex_idm_dr	temperature dependence of the interaction between dark matter and dark radiation
double	m_idm_dr	dark matter mass for idm_dr
short	has_varconst	parameters for varying fundamental constants presence of varying fundamental constants?
int	index_th_xe	ionization fraction x_e
int	index_th_dkappa	Thomson scattering rate $d\kappa/d\tau$ (units 1/Mpc)
int	index_th_tau_d	Baryon drag optical depth
int	index_th_ddkappa	scattering rate derivative $d^2\kappa/d\tau^2$
int	index_th_dddkappa	scattering rate second derivative $d^3\kappa/d\tau^3$
int	index_th_exp_m_kappa	$exp^{-\kappa}$
int	index_th_g	visibility function $g = (d\kappa/d\tau) * exp^{-\kappa}$
int	index_th_dg	visibility function derivative $(dg/d\tau)$
int	index_th_ddg	visibility function second derivative $(d^2g/d\tau^2)$
int	index_th_dmu_idm_dr	scattering rate of idr with idm_dr (i.e. idr opacity to idm_dr scattering) (units 1/Mpc)
int	index_th_ddmu_idm_dr	derivative of this scattering rate
int	index_th_dddmu_idm_dr	second derivative of this scattering rate
int	index_th_dmu_idr	idr self-interaction rate
int	index_th_tau_idm_dr	optical depth of idm_dr (due to interactions with idr)
int	index_th_tau_idr	optical depth of idr (due to self-interactions)
int	index_th_g_idm_dr	visibility function of idm_idr

Data Fields

int	index_th_cidm_dr2	interacting dark matter squared sound speed c^2_{dm}
int	index_th_Tidm_dr	temperature of DM interacting with DR T_{idm_dr}
int	index_th_Tb	baryon temperature T_b
int	index_th_dTb	derivative of baryon temperature
int	index_th_wb	baryon equation of state parameter $w_b = k_B T_b/\mu$
int	index_th_cb2	squared baryon adiabatic sound speed c^2_b
int	index_th_dcb2	derivative wrt conformal time of squared baryon sound speed $d[c^2_b]/d\tau$ (only computed if some non-minimal tight-coupling schemes is requested)
int	index_th_ddcb2	second derivative wrt conformal time of squared baryon sound speed $d^2[c^2_b]/d\tau^2$ (only computed if some non0-minimal tight-coupling schemes is requested)
int	index_th_rate	maximum variation rate of $exp^{-\kappa}$, g and $(dg/d\tau)$, used for computing integration step in perturbation module
int	index_th_r_d	simple analytic approximation to the photon comoving damping scale
int	th_size	size of thermodynamics vector
int	tt_size	number of lines (redshift steps) in the tables
double *	z_table	vector z_table[index_z] with values of redshift (vector of size tt_size)
double *	tau_table	vector tau_table[index_tau] with values of conformal time (vector of size tt_size)
double *	thermodynamics_table	table thermodynamics_table[index↩ _z∗pth->tt_size+pba->index_th] with all other quantities (array of size th_size∗tt_size)
double *	d2thermodynamics_dz2_table	table d2thermodynamics_dz2_↩ table[index_z∗pth->tt_size+pba- >index_th] with values of d^2t_i/dz^2 (array of size th_size∗tt_size)
double	z_rec	z at which the visibility reaches its maximum (= recombination redshift)
double	tau_rec	conformal time at which the visibility reaches its maximum (= recombination time)
double	rs_rec	comoving sound horizon at recombination
double	ds_rec	physical sound horizon at recombination

Data Fields

double	ra_rec	conformal angular diameter distance to recombination
double	da_rec	physical angular diameter distance to recombination
double	rd_rec	comoving photon damping scale at recombination
double	z_star	redshift at which photon optical depth crosses one
double	tau_star	confirmal time at which photon optical depth crosses one
double	rs_star	comoving sound horizon at z_star
double	ds_star	physical sound horizon at z_star
double	ra_star	conformal angular diameter distance to z_star
double	da_star	physical angular diameter distance to z_star
double	rd_star	comoving photon damping scale at z_star
double	z_d	baryon drag redshift
double	tau_d	baryon drag time
double	ds_d	physical sound horizon at baryon drag
double	rs_d	comoving sound horizon at baryon drag
double	tau_cut	at at which the visibility goes below a fixed fraction of the maximum visibility, used for an approximation in perturbation module
double	angular_rescaling	[ratio ra_rec / (tau0-tau_rec)]: gives CMB rescaling in angular space relative to flat model (=1 for curvature K=0)
double	tau_free_streaming	minimum value of tau at which free-streaming approximation can be switched on
double	tau_idr_free_streaming	trigger for dark radiation free streaming approximation (idm-idr)
double	tau_idr	decoupling tau for idr
double	tau_idm_dr	decoupling tau for idm_dr
double	tau_ini	initial conformal time at which thermodynamical variables have been be integrated
double	fHe	f_{He}: primordial helium-to-hydrogen nucleon ratio 4∗n_He/n_H
double	n_e	total number density of electrons today (free or not)
short	inter_normal	flag for calling thermodynamics_at_z and find position in interpolation table normally

Data Fields

short	inter_closeby	flag for calling thermodynamics_at_z and find position in interpolation table starting from previous position in previous call
short	thermodynamics_verbose	flag regulating the amount of information sent to standard output (none if set to zero)
short	hyrec_verbose	flag regulating the amount of information sent to standard output from hyrec (none if set to zero)
ErrorMsg	error_message	zone for writing error messages

4.24.2.2 struct thermo_vector

Other structures that are used during the thermodynamics module execution (i.e. during thermodynamics_init()) but get erased later on: thus they cannot be accessed by other modules. Vector of thermodynamical quantities to integrate over, and indices of this vector

Data Fields

int	ti_size	size of thermo vector (ti stands for thermodynamical, integrated)
int	index_ti_x_H	index for hydrogen fraction in y
int	index_ti_x_He	index for helium fraction in y
int	index_ti_D_Tmat	index for T_mat - T_photon [Kelvin] in y
double *	y	vector of quantities to be integrated
double *	dy	time-derivative of the same vector
int *	used_in_output	boolean array specifying which quantities enter in the calculation of output functions

4.24.2.3 struct thermo_diffeq_workspace

Workspace for differential equation of thermodynamics

Data Fields

double	x_H	Hydrogen ionization fraction
double	x_He	Helium ionization fraction
double	x_noreio	Electron ionization fraction, not taking into account reionization
double	x_reio	Electron ionization fraction, taking into account reionization
double	x	total ionization fraction following usual CMB convention, n_free/n_H = x_H + fHe * x_He;
double	Tmat	matter temperature
int	index_ap_brec	before H- and He-recombination
int	index_ap_He1	during 1st He-recombination (HeIII)
int	index_ap_He1f	in between 1st and 2nd He recombination
int	index_ap_He2	beginning of 2nd He-recombination (HeII)

Data Fields

int	index_ap_H	beginning of H-recombination (HI)
int	index_ap_frec	during and after full H- and HeII-recombination
int	index_ap_reio	during reionization
int	ap_current	
int	ap_size	current approximation scheme index number of approximation intervals used during evolver loop
int	ap_size_loaded	number of all approximations
double *	ap_z_limits	vector storing ending limits of each approximation
double *	ap_z_limits_delta	vector storing smoothing deltas of each approximation
int	require_H	
int	require_He	in given approximation scheme, do we need to integrate hydrogen ionization fraction?
struct thermo_vector *	ptv	in given approximation scheme, do we need to integrate helium ionization fraction? pointer to vector of integrated quantities and their time-derivatives
struct thermohyrec *	phyrec	pointer to wrapper of HyRec structure
struct thermorecfast *	precfast	pointer to wrapper of RecFast structure

4.24.2.4 struct thermo_reionization_parameters

Workspace for reionization

Data Fields

int	index_re_reio_redshift	hydrogen reionization redshift
int	index_re_reio_exponent	an exponent used in the function x_e(z) in the reio_camb scheme
int	index_re_reio_width	a width defining the duration of hydrogen reionization in the reio_camb scheme
int	index_re_xe_before	ionization fraction at redshift 'reio_start'
int	index_re_xe_after	ionization fraction after full reionization
int	index_re_helium_fullreio_fraction	helium full reionization fraction inferred from primordial helium fraction
int	index_re_helium_fullreio_redshift	helium full reionization redshift
int	index_re_helium_fullreio_width	a width defining the duration of helium full reionization in the reio_camb scheme
int	re_z_size	number of reionization jumps
int	index_re_first_z	redshift at which we start to impose reionization function
int	index_re_first_xe	ionization fraction at redshift first_z (inferred from recombination code)
int	index_re_step_sharpness	sharpness of tanh jump
int	index_re_reio_start	redshift above which hydrogen reionization neglected
double *	reionization_parameters	vector containing all reionization parameters necessary to compute xe(z)
int	re_size	length of vector reionization_parameters

4.24.2.5 struct thermo_workspace

General parameters relevant to thermal history and pointers to few other more specialised worspaces

Data Fields

int	Nz_reco_lin	number of redshifts linearly sampled for recombination during the evolver loop
int	Nz_reco_log	number of redshifts logarithmically sampled for recombination during the evolver loop
int	Nz_reco	number of redshifts for recombination during the evolver loop
int	Nz_reio	number of redshift points of reionization during evolver loop
int	Nz_tot	total number of sampled redshifts
double	YHe	defined as in RECFAST : primordial helium mass fraction
double	fHe	defined as in RECFAST : primordial helium-to-hydrogen nucleon ratio
double	SIunit_H0	defined as in RECFAST : Hubble parameter today in SI units
double	SIunit_nH0	defined as in RECFAST : Hydrogen number density today in SI units
double	Tcmb	CMB temperature today in Kelvin
double	const_NR_numberdens	prefactor in number density of nonrelativistic species
double	const_Tion_H	ionization energy for HI as temperature
double	const_Tion_HeI	ionization energy for HeI as temperature
double	const_Tion_HeII	ionization energy for HeII as temperature
double	reionization_optical_depth	reionization optical depth inferred from reionization history
int	last_index_back	nearest location in background table
struct thermo_diffeq_workspace *	ptdw	pointer to workspace for differential equations
struct thermo_reionization_parameters *	ptrp	pointer to workspace for reionization

4.24.2.6 struct thermodynamics_parameters_and_workspace

temporary parameters and workspace passed to the thermodynamics_derivs function

4.24.3 Macro Definition Documentation

4.24.3.1 f1

```
#define f1(
             x ) (-0.75*x*(x*x/3.-1.)+0.5)
```

Two useful smooth step functions, for smoothing transitions in recfast. goes from 0 to 1 when x goes from -1 to 1

4.24.3.2 f2

```
#define f2(
             x ) (x*x*(0.5-x/3.)*6.)
```

goes from 0 to 1 when x goes from 0 to 1

4.24.3.3 _YHE_BIG_

```
#define _YHE_BIG_ 0.5
```

maximal Y_{He}

4.24.3.4 _YHE_SMALL_

```
#define _YHE_SMALL_ 0.01
```

minimal Y_{He}

4.24.4 Enumeration Type Documentation

4.24.4.1 recombination_algorithm

```
enum recombination_algorithm
```

List of possible recombination algorithms.

4.24.4.2 reionization_parametrization

```
enum reionization_parametrization
```

List of possible reionization schemes.

Enumerator

reio_none	no reionization
reio_camb	reionization parameterized like in CAMB
reio_bins_tanh	binned reionization history with tanh inteprolation between bins
reio_half_tanh	half a tanh, instead of the full tanh
reio_many_tanh	similar to reio_camb but with more than one tanh
reio_inter	linear interpolation between specified points

4.24.4.3 reionization_z_or_tau

```
enum reionization_z_or_tau
```

Is the input parameter the reionization redshift or optical depth?

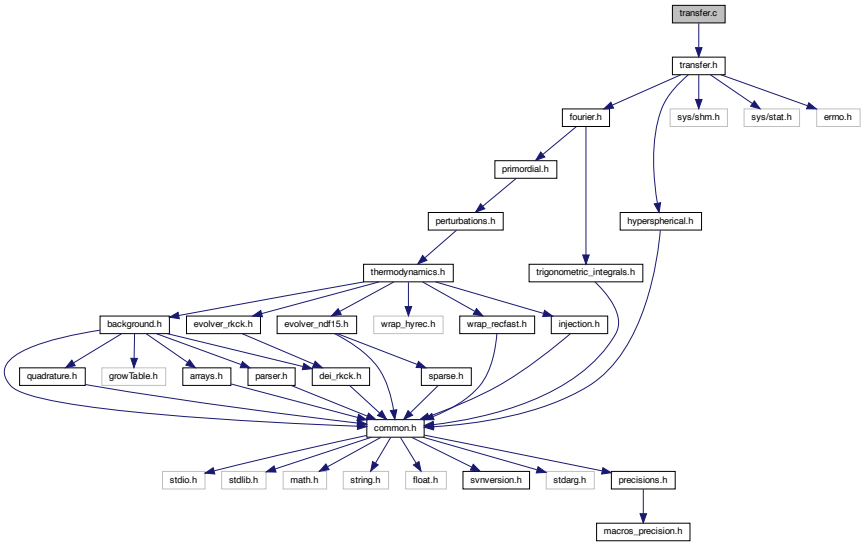
Enumerator

reio_z	input = redshift
reio_tau	input = tau

4.25 transfer.c File Reference

```
#include "transfer.h"
```

Include dependency graph for transfer.c:



Functions

- int transfer_functions_at_q (struct transfer *ptr, int index_md, int index_ic, int index_tt, int index_l, double q, double *transfer_function)
- int transfer_init (struct precision *ppr, struct background *pba, struct thermodynamics *pth, struct perturbations *ppt, struct fourier *pfo, struct transfer *ptr)
- int transfer_free (struct transfer *ptr)
- int transfer_indices (struct precision *ppr, struct perturbations *ppt, struct transfer *ptr, double q_period, double K, int sgnK)
- int transfer_get_l_list (struct precision *ppr, struct perturbations *ppt, struct transfer *ptr)

- int transfer_get_q_list (struct precision *ppr, struct perturbations *ppt, struct transfer *ptr, double q_period, double K, int sgnK)
- int transfer_get_k_list (struct perturbations *ppt, struct transfer *ptr, double K)
- int transfer_get_source_correspondence (struct perturbations *ppt, struct transfer *ptr, int **tp_of_tt)
- int transfer_source_tau_size (struct precision *ppr, struct background *pba, struct perturbations *ppt, struct transfer *ptr, double tau_rec, double tau0, int index_md, int index_tt, int *tau_size)
- int transfer_compute_for_each_q (struct precision *ppr, struct background *pba, struct perturbations *ppt, struct transfer *ptr, int **tp_of_tt, int index_q, int tau_size_max, double tau_rec, double ***pert_sources, double ***pert_sources_spline, double *window, struct transfer_workspace *ptw)
- int transfer_interpolate_sources (struct perturbations *ppt, struct transfer *ptr, int index_q, int index_md, int index_ic, int index_type, double *pert_source, double *pert_source_spline, double *interpolated_sources)
- int transfer_sources (struct precision *ppr, struct background *pba, struct perturbations *ppt, struct transfer *ptr, double *interpolated_sources, double tau_rec, int index_q, int index_md, int index_tt, double *sources, double *window, int tau_size_max, double *tau0_minus_tau, double *w_trapz, int *tau_size_out)
- int transfer_selection_function (struct precision *ppr, struct perturbations *ppt, struct transfer *ptr, int bin, double z, double *selection)
- int transfer_dNdz_analytic (struct transfer *ptr, double z, double *dNdz, double *dln_dNdz_dz)
- int transfer_selection_sampling (struct precision *ppr, struct background *pba, struct perturbations *ppt, struct transfer *ptr, int bin, double *tau0_minus_tau, int tau_size)
- int transfer_lensing_sampling (struct precision *ppr, struct background *pba, struct perturbations *ppt, struct transfer *ptr, int bin, double tau0, double *tau0_minus_tau, int tau_size)
- int transfer_source_resample (struct precision *ppr, struct background *pba, struct perturbations *ppt, struct transfer *ptr, int bin, double *tau0_minus_tau, int tau_size, int index_md, double tau0, double *interpolated↩ _sources, double *sources)
- int transfer_selection_times (struct precision *ppr, struct background *pba, struct perturbations *ppt, struct transfer *ptr, int bin, double *tau_min, double *tau_mean, double *tau_max)
- int transfer_selection_compute (struct precision *ppr, struct background *pba, struct perturbations *ppt, struct transfer *ptr, double *selection, double *tau0_minus_tau, double *w_trapz, int tau_size, double *pvecback, double tau0, int bin)
- int transfer_compute_for_each_l (struct transfer_workspace *ptw, struct precision *ppr, struct perturbations *ppt, struct transfer *ptr, int index_q, int index_md, int index_ic, int index_tt, int index_l, double l, double q_max_bessel, radial_function_type radial_type)
- int transfer_integrate (struct perturbations *ppt, struct transfer *ptr, struct transfer_workspace *ptw, int index_q, int index_md, int index_tt, double l, int index_l, double k, radial_function_type radial_type, double *trsf)
- int transfer_limber (struct transfer *ptr, struct transfer_workspace *ptw, int index_md, int index_q, double l, double q, radial_function_type radial_type, double *trsf)
- int transfer_limber_interpolate (struct transfer *ptr, double *tau0_minus_tau, double *sources, int tau_size, double tau0_minus_tau_limber, double *S)
- int transfer_limber2 (int tau_size, struct transfer *ptr, int index_md, int index_k, double l, double k, double *tau0_minus_tau, double *sources, radial_function_type radial_type, double *trsf)
- int transfer_precompute_selection (struct precision *ppr, struct background *pba, struct perturbations *ppt, struct transfer *ptr, double tau_rec, int tau_size_max, double **window)

4.25.1 Detailed Description

Documented transfer module.

Julien Lesgourgues, 28.07.2013

This module has two purposes:

- at the beginning, to compute the transfer functions $\Delta_l^X(q)$, and store them in tables used for interpolation in other modules.

- at any time in the code, to evaluate the transfer functions (for a given mode, initial condition, type and multipole l) at any wavenumber q (by interpolating within the interpolation table).

Hence the following functions can be called from other modules:

1. transfer_init() at the beginning (but after perturbations_init() and bessel_init())
2. transfer_functions_at_q() at any later time
3. transfer_free() at the end, when no more calls to transfer_functions_at_q() are needed

Note that in the standard implementation of CLASS, only the pre-computed values of the transfer functions are used, no interpolation is necessary; hence the routine transfer_functions_at_q() is actually never called.

4.25.2 Function Documentation

4.25.2.1 transfer_functions_at_q()

```
int transfer_functions_at_q (
             struct transfer * ptr,
             int index_md,
             int index_ic,
             int index_tt,
             int index_l,
             double q,
             double * transfer_function )
```

Transfer function $\Delta_l^X(q)$ at a given wavenumber q.

For a given mode (scalar, vector, tensor), initial condition, type (temperature, polarization, lensing, etc) and multipole, computes the transfer function for an arbitrary value of q by interpolating between pre-computed values of q. This function can be called from whatever module at whatever time, provided that transfer_init() has been called before, and transfer_free() has not been called yet.

Wavenumbers are called q in this module and k in the perturbation module. In flat universes k=q. In non-flat universes q and k differ through $q2 = k2 + K(1 + m)$, where m=0,1,2 for scalar, vector, tensor. q should be used throughout the transfer module, excepted when interpolating or manipulating the source functions S(k,tau) calculated in the perturbation module: for a given value of q, this should be done at the corresponding k(q).

Parameters

ptr	Input: pointer to transfer structure
index_md	Input: index of requested mode
index_ic	Input: index of requested initial condition
index_tt	Input: index of requested type
index_l	Input: index of requested multipole
q	Input: any wavenumber
transfer_function	Output: transfer function

Returns

the error status

Summary:

- interpolate in pre-computed table using array_interpolate_two()

4.25.2.2 transfer_init()

```
int transfer_init (
             struct precision * ppr,
             struct background * pba,
             struct thermodynamics * pth,
             struct perturbations * ppt,
             struct fourier * pfo,
             struct transfer * ptr )
```

This routine initializes the transfer structure, (in particular, computes table of transfer functions $\Delta_l^X(q)$)

Main steps:

- initialize all indices in the transfer structure and allocate all its arrays using transfer_indices().
- for each thread (in case of parallel run), initialize the fields of a memory zone called the transfer_workspace with transfer_workspace_init()
- loop over q values. For each q, compute the Bessel functions if needed with transfer_update_HIS(), and defer the calculation of all transfer functions to transfer_compute_for_each_q()
- for each thread, free the the workspace with transfer_workspace_free()

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
pth	Input: pointer to thermodynamics structure
ppt	Input: pointer to perturbation structure
pfo	Input: pointer to fourier structure
ptr	Output: pointer to initialized transfer structure

Returns

the error status

Summary:

- define local variables

- array with the correspondence between the index of sources in the perturbation module and in the transfer module, tp_of_tt[index_md][index_tt]
- check whether any spectrum in harmonic space (i.e., any C_l's) is actually requested
- get number of modes (scalars, tensors...)
- get conformal age / recombination time from background / thermodynamics structures (only place where these structures are used in this module)
- correspondence between k and l depend on angular diameter distance, i.e. on curvature.
- order of magnitude of the oscillation period of transfer functions
- initialize all indices in the transfer structure and allocate all its arrays using transfer_indices()
- copy sources to a local array sources (in fact, only the pointers are copied, not the data), and eventually apply non-linear corrections to the sources
- spline all the sources passed by the perturbation module with respect to k (in order to interpolate later at a given value of k)
- allocate and fill array describing the correspondence between perturbation types and transfer types
- evaluate maximum number of sampled times in the transfer sources: needs to be known here, in order to allocate a large enough workspace
- compute flat spherical bessel functions
- eventually read the selection and evolution functions
- precompute window function for integrated nCl/sCl quantities
- loop over all wavenumbers (parallelized).
- finally, free arrays allocated outside parallel zone

4.25.2.3 transfer_free()

```
int transfer_free (
             struct transfer * ptr )
```

This routine frees all the memory space allocated by transfer_init().

To be called at the end of each run, only when no further calls to transfer_functions_at_k() are needed.

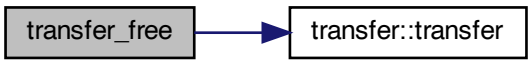
Parameters

ptr	Input: pointer to transfer structure (which fields must be freed)

Returns

the error status

Here is the call graph for this function:



4.25.2.4 transfer_indices()

```
int transfer_indices (
              struct precision * ppr,
              struct perturbations * ppt,
              struct transfer * ptr,
              double q_period,
              double K,
              int sgnK )
```

This routine defines all indices and allocates all tables in the transfer structure

Compute list of (k, l) values, allocate and fill corresponding arrays in the transfer structure. Allocate the array of transfer function tables.

Parameters

ppr	Input: pointer to precision structure
ppt	Input: pointer to perturbation structure
ptr	Input/Output: pointer to transfer structure
q_period	Input: order of magnitude of the oscillation period of transfer functions
K	Input: spatial curvature (in absolute value)
sgnK	Input: spatial curvature sign (open/closed/flat)

Returns

the error status

Summary:

- define local variables
- define indices for transfer types

- type indices common to scalars and tensors
- type indices for scalars
- type indices for vectors
- type indices for tensors
- allocate arrays of (k, l) values and transfer functions
- get q values using transfer_get_q_list()
- get k values using transfer_get_k_list()
- get l values using transfer_get_l_list()
- loop over modes (scalar, etc). For each mode:
- allocate arrays of transfer functions, (ptr->transfer[index_md])[index_ic][index_tt][index_l][index_k]

4.25.2.5 transfer_get_l_list()

```
int transfer_get_l_list (
             struct precision * ppr,
             struct perturbations * ppt,
             struct transfer * ptr )
```

This routine defines the number and values of multipoles l for all modes.

Parameters

ppr	Input: pointer to precision structure
ppt	Input: pointer to perturbation structure
ptr	Input/Output: pointer to transfer structure containing l's

Returns

the error status

Summary:

- allocate and fill l array
- start from l = 2 and increase with logarithmic step
- when the logarithmic step becomes larger than some linear step, stick to this linear step till l_max
- last value set to exactly l_max
- so far we just counted the number of values. Now repeat the whole thing but fill array with values.

4.25.2.6 transfer_get_q_list()

```
int transfer_get_q_list (
             struct precision * ppr,
             struct perturbations * ppt,
             struct transfer * ptr,
             double q_period,
             double K,
             int sgnK )
```

This routine defines the number and values of wavenumbers q for each mode (goes smoothly from logarithmic step for small q's to linear step for large q's).

Parameters

ppr	Input: pointer to precision structure
ppt	Input: pointer to perturbation structure
ptr	Input/Output: pointer to transfer structure containing q's
q_period	Input: order of magnitude of the oscillation period of transfer functions
K	Input: spatial curvature (in absolute value)
sgnK	Input: spatial curvature sign (open/closed/flat)

Returns

the error status

4.25.2.7 transfer_get_k_list()

```
int transfer_get_k_list (
             struct perturbations * ppt,
             struct transfer * ptr,
             double K )
```

This routine infers from the q values a list of corresponding k values for each mode.

Parameters

ppt	Input: pointer to perturbation structure
ptr	Input/Output: pointer to transfer structure containing q's
K	Input: spatial curvature

Returns

the error status

4.25.2.8 transfer_get_source_correspondence()

```
int transfer_get_source_correspondence (
             struct perturbations * ppt,
             struct transfer * ptr,
             int ** tp_of_tt )
```

This routine defines the correspondence between the sources in the perturbation and transfer module.

Parameters

ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure containing l's
tp_of_tt	Input/Output: array with the correspondence (allocated before, filled here)

Returns

the error status

Summary:

- running index on modes
- running index on transfer types
- which source are we considering? Define correspondence between transfer types and source types

4.25.2.9 transfer_source_tau_size()

```
int transfer_source_tau_size (
             struct precision * ppr,
             struct background * pba,
             struct perturbations * ppt,
             struct transfer * ptr,
             double tau_rec,
             double tau0,
             int index_md,
             int index_tt,
             int * tau_size )
```

the code makes a distinction between "perturbation sources" (e.g. gravitational potential) and "transfer sources" (e.g. total density fluctuations, obtained through the Poisson equation, and observed with a given selection function).

This routine computes the number of sampled time values for each type of transfer sources.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure

Parameters

ptr	Input: pointer to transfer structure
tau_rec	Input: recombination time
tau0	Input: time today
index_md	Input: index of the mode (scalar, tensor)
index_tt	Input: index of transfer type
tau_size	Output: pointer to number of sampled times

Returns

the error status

4.25.2.10 transfer_compute_for_each_q()

```
int transfer_compute_for_each_q (
             struct precision * ppr,
             struct background * pba,
             struct perturbations * ppt,
             struct transfer * ptr,
             int ** tp_of_tt,
             int index_q,
             int tau_size_max,
             double tau_rec,
             double *** pert_sources,
             double *** pert_sources_spline,
             double * window,
             struct transfer_workspace * ptw )
```

Summary:

- define local variables
- we deal with workspaces, i.e. with contiguous memory zones (one per thread) containing various fields used by the integration routine
- for a given l, maximum value of k such that we can convolve the source with Bessel functions j_l(x) without reaching x_max
- store the sources in the workspace and define all fields in this workspace
- loop over all modes. For each mode
- loop over initial conditions.
- check if we must now deal with a new source with a new index ppt->index_type. If yes, interpolate it at the right values of k.
- Select radial function type

4.25.2.11 transfer_interpolate_sources()

```
int transfer_interpolate_sources (
             struct perturbations * ppt,
             struct transfer * ptr,
             int index_q,
             int index_md,
             int index_ic,
             int index_type,
             double * pert_source,
             double * pert_source_spline,
             double * interpolated_sources )
```

This routine interpolates sources $S(k,\tau)$ for each mode, initial condition and type (of perturbation module), to get them at the right values of k, using the spline interpolation method.

Parameters

ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
index_q	Input: index of wavenumber
index_md	Input: index of mode
index_ic	Input: index of initial condition
index_type	Input: index of type of source (in perturbation module)
pert_source	Input: array of sources
pert_source_spline	Input: array of second derivative of sources
interpolated_sources	Output: array of interpolated sources (filled here but allocated in transfer_init() to avoid numerous reallocation)

Returns

the error status

Summary:

- define local variables
- interpolate at each k value using the usual spline interpolation algorithm.

4.25.2.12 transfer_sources()

```
int transfer_sources (
             struct precision * ppr,
             struct background * pba,
             struct perturbations * ppt,
             struct transfer * ptr,
             double * interpolated_sources,
             double tau_rec,
             int index_q,
```

```
int index_md,
int index_tt,
double * sources,
double * window,
int tau_size_max,
double * tau0_minus_tau,
double * w_trapz,
int * tau_size_out )
```

The code makes a distinction between "perturbation sources" (e.g. gravitational potential) and "transfer sources" (e.g. total density fluctuations, obtained through the Poisson equation, and observed with a given selection function).

This routine computes the transfer source given the interpolated perturbation source, and copies it in the workspace.

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
interpolated_sources	Input: interpolated perturbation source
tau_rec	Input: recombination time
index_q	Input: index of wavenumber
index_md	Input: index of mode
index_tt	Input: index of type of (transfer) source
sources	Output: transfer source
window	Input: window functions for each type and time
tau_size_max	Input: number of times at wich window fucntions are sampled
tau0_minus_tau	Output: values of (tau0-tau) at which source are sample
w_trapz	Output: trapezoidal weights for integration over tau
tau_size_out	Output: pointer to size of previous two arrays, converted to double

Returns

the error status

Summary:

- define local variables
- in which cases are perturbation and transfer sources are different? I.e., in which case do we need to multiply the sources by some background and/or window function, and eventually to resample it, or redefine its time limits?
- case where we need to redefine by a window function (or any function of the background and of k)
- case where we do not need to redefine
- return tau_size value that will be stored in the workspace (the workspace wants a double)

4.25.2.13 transfer_selection_function()

```
int transfer_selection_function (
             struct precision * ppr,
             struct perturbations * ppt,
             struct transfer * ptr,
             int bin,
             double z,
             double * selection )
```

Arbitrarily normalized selection function dN/dz(z,bin)

Parameters

ppr	Input: pointer to precision structure
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
bin	Input: redshift bin number
z	Input: one value of redshift
selection	Output: pointer to selection function

Returns

the error status

4.25.2.14 transfer_dNdz_analytic()

```
int transfer_dNdz_analytic (
             struct transfer * ptr,
             double z,
             double * dNdz,
             double * dln_dNdz_dz )
```

Analytic form for dNdz distribution, from arXiv:1004.4640

Parameters

ptr	Input: pointer to transfer structure
z	Input: redshift
dNdz	Output: density per redshift, dN/dZ
dln_dNdz_dz	Output: dln(dN/dz)/dz, used optionally for the source evolution

Returns

the error status

4.25.2.15 transfer_selection_sampling()

```
int transfer_selection_sampling (
             struct precision * ppr,
             struct background * pba,
             struct perturbations * ppt,
             struct transfer * ptr,
             int bin,
             double * tau0_minus_tau,
             int tau_size )
```

For sources that need to be multiplied by a selection function, redefine a finer time sampling in a small range

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
bin	Input: redshift bin number
tau0_minus_tau	Output: values of (tau0-tau) at which source are sample
tau_size	Output: pointer to size of previous array

Returns

the error status

4.25.2.16 transfer_lensing_sampling()

```
int transfer_lensing_sampling (
             struct precision * ppr,
             struct background * pba,
             struct perturbations * ppt,
             struct transfer * ptr,
             int bin,
             double tau0,
             double * tau0_minus_tau,
             int tau_size )
```

For lensing sources that need to be convolved with a selection function, redefine the sampling within the range extending from the tau_min of the selection function up to tau0

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
bin	Input: redshift bin number
tau0	Input: time today
tau0_minus_tau	Output: values of (tau0-tau) at which source are sample
tau_size	Output: pointer to size of previous array

Returns

the error status

4.25.2.17 transfer_source_resample()

```
int transfer_source_resample (
             struct precision * ppr,
             struct background * pba,
             struct perturbations * ppt,
             struct transfer * ptr,
             int bin,
             double * tau0_minus_tau,
             int tau_size,
             int index_md,
             double tau0,
             double * interpolated_sources,
             double * sources )
```

For sources that need to be multiplied by a selection function, redefine a finer time sampling in a small range, and resample the perturbation sources at the new value by linear interpolation

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
bin	Input: redshift bin number
tau0_minus_tau	Output: values of (tau0-tau) at which source are sample
tau_size	Output: pointer to size of previous array
index_md	Input: index of mode
tau0	Input: time today
interpolated_sources	Input: interpolated perturbation source
sources	Output: resampled transfer source

Returns

the error status

4.25.2.18 transfer_selection_times()

```
int transfer_selection_times (
             struct precision * ppr,
             struct background * pba,
             struct perturbations * ppt,
```

```
              struct transfer * ptr,
              int bin,
              double * tau_min,
              double * tau_mean,
              double * tau_max )
```

For each selection function, compute the min, mean and max values of conformal time (associated to the min, mean and max values of redshift specified by the user)

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
bin	Input: redshift bin number
tau_min	Output: smallest time in the selection interval
tau_mean	Output: time corresponding to z_mean
tau_max	Output: largest time in the selection interval

Returns

the error status

4.25.2.19 transfer_selection_compute()

```
int transfer_selection_compute (
              struct precision * ppr,
              struct background * pba,
              struct perturbations * ppt,
              struct transfer * ptr,
              double * selection,
              double * tau0_minus_tau,
              double * w_trapz,
              int tau_size,
              double * pvecback,
              double tau0,
              int bin )
```

Compute and normalize selection function for a set of time values

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
selection	Output: normalized selection function
tau0_minus_tau	Input: values of (tau0-tau) at which source are sample
w_trapz	Input: trapezoidal weights for integration over tau

Parameters

tau_size	Input: size of previous two arrays
pvecback	Input: allocated array of background values
tau0	Input: time today
bin	Input: redshift bin number

Returns

the error status

4.25.2.20 transfer_compute_for_each_l()

```
int transfer_compute_for_each_l (
              struct transfer_workspace * ptw,
              struct precision * ppr,
              struct perturbations * ppt,
              struct transfer * ptr,
              int index_q,
              int index_md,
              int index_ic,
              int index_tt,
              int index_l,
              double l,
              double q_max_bessel,
              radial_function_type radial_type )
```

This routine computes the transfer functions $\Delta_l^X(k))$ as a function of wavenumber k for a given mode, initial condition, type and multipole l passed in input.

For a given value of k, the transfer function is inferred from the source function (passed in input in the array interpolated_sources) and from Bessel functions (passed in input in the bessels structure), either by convolving them along tau, or by a Limber approximation. This elementary task is distributed either to transfer_integrate() or to transfer_limber(). The task of this routine is mainly to loop over k values, and to decide at which k_max the calculation can be stopped, according to some approximation scheme designed to find a compromise between execution time and precision. The approximation scheme is defined by parameters in the precision structure.

Parameters

ptw	Input: pointer to transfer_workspace structure (allocated in transfer_init() to avoid numerous reallocation)
ppr	Input: pointer to precision structure
ppt	Input: pointer to perturbation structure
ptr	Input/output: pointer to transfer structure (result stored there)
index_q	Input: index of wavenumber
index_md	Input: index of mode
index_ic	Input: index of initial condition
index_tt	Input: index of type of transfer
index_l	Input: index of multipole
l	Input: multipole
q_max_bessel	Input: maximum value of argument q at which Bessel functions are computed
radial_type	Input: type of radial (Bessel) functions to convolve with

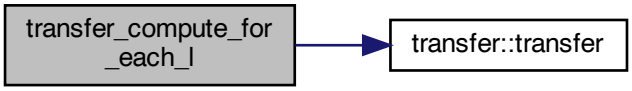
Returns

the error status

Summary:

- define local variables
- return zero transfer function if l is above l_max
- store transfer function in transfer structure

Here is the call graph for this function:



4.25.2.21 transfer_integrate()

```
int transfer_integrate (
             struct perturbations * ppt,
             struct transfer * ptr,
             struct transfer_workspace * ptw,
             int index_q,
             int index_md,
             int index_tt,
             double l,
             int index_l,
             double k,
             radial_function_type radial_type,
             double * trsf )
```

This routine computes the transfer functions $\Delta_l^X(k)$) for each mode, initial condition, type, multipole l and wavenumber k, by convolving the source function (passed in input in the array interpolated_sources) with Bessel functions (passed in input in the bessels structure).

Parameters

ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
ptw	Input: pointer to transfer_workspace structure (allocated in transfer_init() to avoid numerous reallocation)
index_q	Input: index of wavenumber
index_md	Input: index of mode

Parameters

index_tt	Input: index of type
l	Input: multipole
index_l	Input: index of multipole
k	Input: wavenumber
radial_type	Input: type of radial (Bessel) functions to convolve with
trsf	Output: transfer function $\Delta_l(k)$

Returns

the error status

Summary:

- define local variables
- find minimum value of (tau0-tau) at which $j_l(k[\tau_0 - \tau])$ is known, given that $j_l(x)$ is sampled above some finite value $x_{\rm min}$ (below which it can be approximated by zero)
- if there is no overlap between the region in which bessels and sources are non-zero, return zero
- if there is an overlap:
- --> trivial case: the source is a Dirac function and is sampled in only one point
- --> other cases
- —> (a) find index in the source's tau list corresponding to the last point in the overlapping region. After this step, index_tau_max can be as small as zero, but not negative.
- —> (b) the source function can vanish at large τ. Check if further points can be eliminated. After this step and if we did not return a null transfer function, index_tau_max can be as small as zero, but not negative.
- Compute the radial function:
- Now we do most of the convolution integral:
- This integral is correct for the case where no truncation has occurred. If it has been truncated at some index_tau_max because f[index_tau_max+1]==0, it is still correct. The 'mistake' in using the wrong weight w_trapz[index_tau_max] is exactly compensated by the triangle we miss. However, for the Bessel cut off, we must subtract the wrong triangle and add the correct triangle.

4.25.2.22 transfer_limber()

```
int transfer_limber (
             struct transfer * ptr,
             struct transfer_workspace * ptw,
             int index_md,
             int index_q,
             double l,
             double q,
             radial_function_type radial_type,
             double * trsf )
```

This routine computes the transfer functions $\Delta_l^X(k)$) for each mode, initial condition, type, multipole l and wavenumber k, by using the Limber approximation, i.e by evaluating the source function (passed in input in the array interpolated_sources) at a single value of tau (the Bessel function being approximated as a Dirac distribution).

Parameters

ptr	Input: pointer to transfer structure
ptw	Input: pointer to transfer workspace structure
index_md	Input: index of mode
index_q	Input: index of wavenumber
l	Input: multipole
q	Input: wavenumber
radial_type	Input: type of radial (Bessel) functions to convolve with
trsf	Output: transfer function $\Delta_l(k)$

Returns

the error status

Summary:

- define local variables
- get k, l and infer tau such that k(tau0-tau)=l+1/2; check that tau is in appropriate range
- get transfer = source $* \sqrt{\pi/(2l+1)}/q$ = source$*$[tau0-tau] $* \sqrt{\pi/(2l+1)}/(l+1/2)$

4.25.2.23 transfer_limber_interpolate()

```
int transfer_limber_interpolate (
             struct transfer * ptr,
             double * tau0_minus_tau,
             double * sources,
             int tau_size,
             double tau0_minus_tau_limber,
             double * S )
```

- find bracketing indices. index_tau must be at least 1 (so that index_tau-1 is at least 0) and at most tau_size-2 (so that index_tau+1 is at most tau_size-1).
- interpolate by fitting a polynomial of order two; get source and its first two derivatives. Note that we are not interpolating S, but the product S$*$(tau0-tau). Indeed this product is regular in tau=tau0, while S alone diverges for lensing.

4.25.2.24 transfer_limber2()

```
int transfer_limber2 (
             int tau_size,
             struct transfer * ptr,
             int index_md,
             int index_k,
             double l,
             double k,
             double * tau0_minus_tau,
             double * sources,
             radial_function_type radial_type,
             double * trsf )
```

This routine computes the transfer functions $\Delta_l^X(k)$) for each mode, initial condition, type, multipole l and wavenumber k, by using the Limber approximation at order two, i.e as a function of the source function and its first two derivatives at a single value of tau

Parameters

tau_size	Input: size of conformal time array
ptr	Input: pointer to transfer structure
index_md	Input: index of mode
index_k	Input: index of wavenumber
l	Input: multipole
k	Input: wavenumber
tau0_minus_tau	Input: array of values of (tau_today - tau)
sources	Input: source functions
radial_type	Input: type of radial (Bessel) functions to convolve with
trsf	Output: transfer function $\Delta_l(k)$

Returns

the error status

Summary:

- define local variables
- get k, l and infer tau such that k(tau0-tau)=l+1/2; check that tau is in appropriate range
- find bracketing indices
- interpolate by fitting a polynomial of order two; get source and its first two derivatives
- get transfer from 2nd order Limber approx (inferred from 0809.5112 [astro-ph])

4.25.2.25 transfer_precompute_selection()

```
int transfer_precompute_selection (
             struct precision * ppr,
             struct background * pba,
             struct perturbations * ppt,
             struct transfer * ptr,
             double tau_rec,
             int tau_size_max,
             double ** window )
```

Here we can precompute the window functions for the final integration For each type of nCl/dCl/sCl we combine the selection function with the corresponding prefactor (e.g. 1/aH), and, if required, we also integrate for integrated (lensed) contributions (In the original ClassGAL paper these would be labeled g4,g5, and lens)

All factors of k have to be added later (at least in the current version)

Parameters

ppr	Input: pointer to precision structure
pba	Input: pointer to background structure
ppt	Input: pointer to perturbation structure
ptr	Input: pointer to transfer structure
tau_rec	Input: recombination time
tau_size_max	Input: maximum size that tau array can have
window	Output: pointer to array of selection functions

Returns

the error status

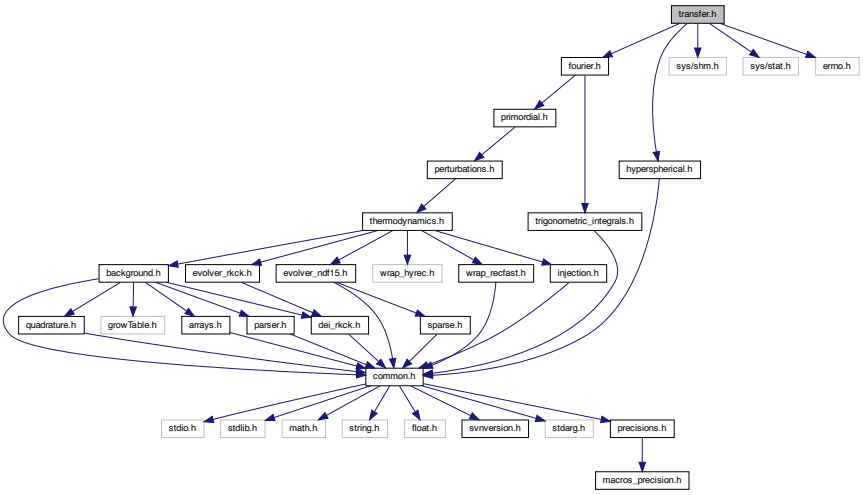
Summary:

- define local variables

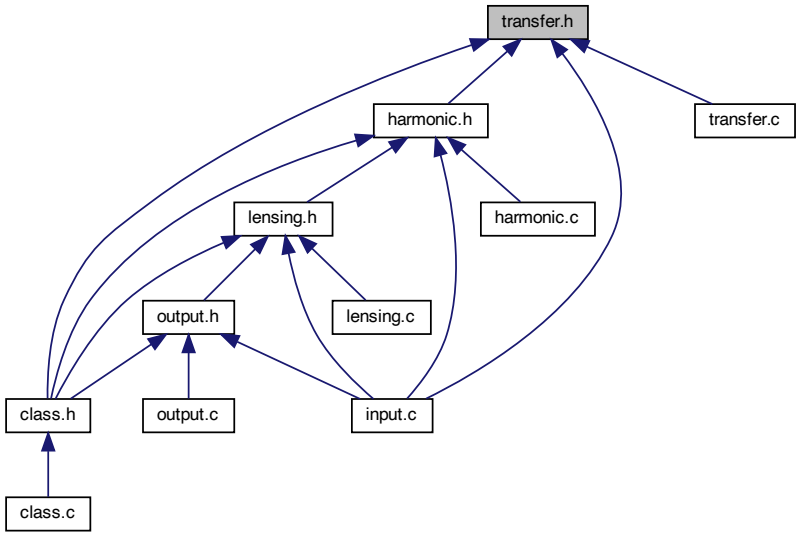
4.26 transfer.h File Reference

```
#include "fourier.h"
#include "hyperspherical.h"
#include <sys/shm.h>
#include <sys/stat.h>
#include "errno.h"
```

Include dependency graph for transfer.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct transfer
- struct transfer_workspace

Enumerations

- enum radial_function_type

4.26.1 Detailed Description

Documented includes for transfer module.

4.26.2 Data Structure Documentation

4.26.2.1 struct transfer

Structure containing everything about transfer functions in harmonic space $\Delta_l^X(q)$ that other modules need to know.

Once initialized by transfer_init(), contains all tables of transfer functions used for interpolation in other modules, for all requested modes (scalar/vector/tensor), initial conditions, types (temperature, polarization, etc), multipoles l, and wavenumbers q.

Wavenumbers are called q in this module and k in the perturbation module. In flat universes k=q. In non-flat universes q and k differ through q2 = k2 + K(1+m), where m=0,1,2 for scalar, vector, tensor. q should be used throughout the transfer module, except when interpolating or manipulating the source functions S(k,tau) calculated in the perturbation module: for a given value of q, this should be done at the corresponding k(q).

The content of this structure is entirely computed in this module, given the content of the 'precision', 'bessels', 'background', 'thermodynamics' and 'perturbation' structures.

Data Fields

double	lcmb_rescale	normally set to one, can be used exceptionally to rescale by hand the CMB lensing potential
double	lcmb_tilt	normally set to zero, can be used exceptionally to tilt by hand the CMB lensing potential
double	lcmb_pivot	if lcmb_tilt non-zero, corresponding pivot scale
double	selection_bias[_SELECTION_NUM_MAX_]	light-to-mass bias in the transfer function of density number count
double	selection_magnification_bias[_SELECTION_NUM_MAX_]	magnification bias in the transfer function of density number count
short	has_nz_file	Has dN/dz (selection function) input file?
short	has_nz_analytic	Use analytic form for dN/dz (selection function) distribution?
FileName	nz_file_name	dN/dz (selection function) input file name
int	nz_size	number of redshift values in input tabulated selection function
double *	nz_z	redshift values in input tabulated selection function
double *	nz_nz	input tabulated values of selection function
double *	nz_ddnz	second derivatives in splined selection function
short	has_nz_evo_file	Has dN/dz (evolution function) input file?
short	has_nz_evo_analytic	Use analytic form for dN/dz (evolution function) distribution?
FileName	nz_evo_file_name	dN/dz (evolution function) input file name
int	nz_evo_size	number of redshift values in input tabulated evolution function
double *	nz_evo_z	redshift values in input tabulated evolution function
double *	nz_evo_nz	input tabulated values of evolution function
double *	nz_evo_dlog_nz	log of tabulated values of evolution function

Data Fields

double *	nz_evo_dd_dlog_nz	second derivatives in splined log of evolution function
short	has_cls	copy of same flag in perturbation structure
int	md_size	number of modes included in computation
int	index_tt_t0	index for transfer type = temperature (j=0 term)
int	index_tt_t1	index for transfer type = temperature (j=1 term)
int	index_tt_t2	index for transfer type = temperature (j=2 term)
int	index_tt_e	index for transfer type = E-polarization
int	index_tt_b	index for transfer type = B-polarization
int	index_tt_lcmb	index for transfer type = CMB lensing
int	index_tt_density	index for first bin of transfer type = matter density
int	index_tt_lensing	index for first bin of transfer type = galaxy lensing
int	index_tt_rsd	index for first bin of transfer type = redshift space distortion of number count
int	index_tt_d0	index for first bin of transfer type = doppler effect for of number count (j=0 term)
int	index_tt_d1	index for first bin of transfer type = doppler effect for of number count (j=1 term)
int	index_tt_nc_lens	index for first bin of transfer type = lensing for of number count
int	index_tt_nc_g1	index for first bin of transfer type = gravity term G1 for of number count
int	index_tt_nc_g2	index for first bin of transfer type = gravity term G2 for of number count
int	index_tt_nc_g3	index for first bin of transfer type = gravity term G3 for of number count
int	index_tt_nc_g4	index for first bin of transfer type = gravity term G3 for of number count
int	index_tt_nc_g5	index for first bin of transfer type = gravity term G3 for of number count
int *	tt_size	number of requested transfer types tt_size[index_md] for each mode
int **	l_size_tt	number of multipole values for which we effectively compute the transfer function,l_size_tt[index_md][index_tt]
int *	l_size	number of multipole values for each requested mode, l_size[index_md]
int	l_size_max	greatest of all l_size[index_md]
int *	l	list of multipole values l[index_l]
double	angular_rescaling	correction between l and k space due to curvature (= comoving angular diameter distance to recombination / comoving radius to recombination)
size_t	q_size	number of wavenumber values
double *	q	list of wavenumber values, q[index_q]

Data Fields

double **	k	list of wavenumber values for each requested mode, k[index_md][index_q]. In flat universes k=q. In non-flat universes q and k differ through q2 = k2 + K(1+m), where m=0,1,2 for scalar, vector, tensor. q should be used throughout the transfer module, excepted when interpolating or manipulating the source functions S(k,tau): for a given value of q this should be done in k(q).
int	index_q_flat_approximation	index of the first q value using the flat rescaling approximation
double **	transfer	table of transfer functions for each mode, initial condition, type, multipole and wavenumber, with argument transfer[index_md][((index_ic * ptr->tt_size[index_md] + index_tt) * ptr->l_size[index_md] + index_l) * ptr->q_size + index_q]
short	transfer_verbose	flag regulating the amount of information sent to standard output (none if set to zero)
ErrorMsg	error_message	zone for writing error messages

4.26.2.2 struct transfer_workspace

Structure containing all the quantities that each thread needs to know for computing transfer functions (but that can be forgotten once the transfer functions are known, otherwise they would be stored in the transfer module)

Data Fields

HyperInterpStruct	HIS	structure containing all hyperspherical bessel functions (flat case) or all hyperspherical bessel functions for a given value of beta=q/sqrt(|K|) (non-flat case). HIS = Hyperspherical Interpolation Structure.
int	HIS_allocated	flag specifying whether the previous structure has been allocated
HyperInterpStruct *	pBIS	pointer to structure containing all the spherical bessel functions of the flat case (used even in the non-flat case, for approximation schemes). pBIS = pointer to Bessel Interpolation Structure.
int	l_size	number of l values
int	tau_size	number of discrete time values for a given type
int	tau_size_max	maximum number of discrete time values for all types
double *	interpolated_sources	interpolated_sources[index_tau]: sources interpolated from the perturbation module at the right value of k
double *	sources	sources[index_tau]: sources used in transfer module, possibly differing from those in the perturbation module by some resampling or rescaling
double *	tau0_minus_tau	tau0_minus_tau[index_tau]: values of (tau0 - tau)
double *	w_trapz	w_trapz[index_tau]: values of weights in trapezoidal integration (related to time steps)
double *	chi	chi[index_tau]: value of argument of bessel function: k(tau0-tau) (flat case) or sqrt(|K|)(tau0-tau) (non-flat case)

Data Fields

double *	cscKgen	cscKgen[index_tau]: useful trigonometric function
double *	cotKgen	cotKgen[index_tau]: useful trigonometric function
double	K	curvature parameter (see background module for details)
int	sgnK	0 (flat), 1 (positive curvature, spherical, closed), -1 (negative curvature, hyperbolic, open)
double	tau0_minus_tau_cut	critical value of (tau0-tau) in time cut approximation for the wavenumber at hand
short	neglect_late_source	flag stating whether we use the time cut approximation for the wavenumber at hand

4.26.3 Enumeration Type Documentation

4.26.3.1 radial_function_type

```
enum radial_function_type
```

enumeration of possible source types. This looks redundant with respect to the definition of indices index_tt_... This definition is however convenient and time-saving: it allows to use a "case" statement in transfer_radial_function()

Chapter 5

The <tt>external_Pk</tt> mode

- Author: Jesus Torrado (torradocacho [@] lorentz.leidenuniv.nl)
- Date: 2013-12-20

5.1 Introduction

This mode allows for an arbitrary primordial spectrum `P(k)` to be calculated by an external command and passed to CLASS. That external command may be anything that can be run in the shell: a python script, some compiled C or Fortran code... This command is executed from within CLASS, and CLASS is able to pass it a number of parameters defining the spectrum (an amplitude, a tilt...). Those parameters can be used in a Markov chain search performed by MontePython.

This mode includes the simple case of a precomputed primordial spectrum stored in a text file. In that case, the `cat` shell command will do the trick (see below).

Currently, scalar and tensor spectra of perturbations of adiabatic modes are supported.

5.2 Use case #1: reading the spectrum from a table

In this case, say the file with the table is called `spectrum.txt`, located under `/path/to`, simply include in the `.ini` file

```
command = cat path/to/spectrum.txt
```

It is necessary that 1st 4 characters are exactly `cat`.

5.3 Use case #2: getting the spectrum from an external command

Here an external command is called to generate the spectrum; it may be some compiled C or Fortran code, a python script... This command may be passed up to 10 floating point arguments, named `custom1` to `custom10`, which are assigned values inside the `.ini` file of CLASS. The `command` parameter would look like

```
command = /path/to/example.py
```

if it starts with `#/usr/bin/python`, otherwise

```
command = python /path/to/example.py
```

As an example of the 1st use case, one may use the included script `generate_Pk_example.py`, which implements a single-field slow-roll spectrum without running, and takes 3 arguments:

- `custom1` – the pivot scale (`k_0 = 0.05 1/Mpc` for Planck).
- `custom2` – the amplitude of the scalar power spectrum.
- `custom3` – the scalar spectral index.

In order to use it, the following lines must be present in the parameter file:

```
P_k_ini type = external_Pk
command = /path/to/CLASS/external_Pk/generate_Pk_example.py
custom1 = 0.05
custom2 = 2.2e-9
custom3 = 1.
```

Defined or not (in that case, 0-valued), parameters from `custom4` to `custom10` will be passed to the example script, which should ignore them. In this case, CLASS will run in the shell the command

```
/path/to/CLASS/external_Pk/generate_Pk_example.py 0.05 2.2e-9 1. 0 0 0 0 0 0 0
```

If CLASS fails to run the command, try to do it directly yourself by hand, using exactly the same string that was given in `command`.

5.4 Output of the command / format of the table

The command must generate an output separated into lines, each containing a tuple (`k`, `P(k)`). The following requirements must be fulfilled:

- Each line must contain 2 (3, if tensors) floating point numbers: `k` (in `1/Mpc` units) and `P_s(k)` (and `P_t(k)`, if tensors), separated by any number of spaces or tabs. The numbers can be in scientific notation, e.g. `1.4e-3`.
- The lines must be sorted in increasing values of `k`.
- There must be at least two points `(k, P(k))` before and after the interval of `k` requested by CLASS, in order not to introduce unnecessary interpolation error. Otherwise, an error will be raised. In most of the cases, generating the spectrum between `1e-6` and `1 1/Mpc` should be more than enough.

5.5 Precision

This implementation properly handles double-precision floating point numbers (i.e. about 17 significant figures), both for the input parameters of the command and for the output of the command (or the table).

The sampling of `k` given by the command (or table) is preserved to be used internally by CLASS. It must be fine enough a sampling to clearly show the features of the spectrum. The best way to test this is to plot the output/table and check it with the naked eye.

Another thing to have in mind arises at the time of convolving with the transfer functions. Two precision parameters are implied: the sampling of `k` in the integral, given by `k_step_trans`, and the sampling of the transfer functions in `l`, given by `l_logstep` and `l_linstep`. In general, it will be enough to reduce the values of the first and the third parameters. A good start is to give them rather small values, say `k_step_trans=0.01` and `l_linstep=1`, and to increase them slowly until the point at which the effect of increasing them gets noticeable.

5.6 Parameter fit with MontePython

(MontePython)[http://montepython.net/] is able to interact with the `external_Pk` mode transparently, using the `custom` parameters in an MCMC fit. One must just add the appropriate lines to the input file of MontePython. For our example, if we wanted to fit the amplitude and spectral index of the primordial spectrum, it would be:

```
data.cosmo_arguments['P_k_ini type'] = 'external_Pk'
data.cosmo_arguments['command'] = '/path/to/CLASS/external_Pk/generate_Pk_example.py'
data.cosmo_arguments['custom1'] = 0.05                                  # k_pivot
data.parameters['custom2']      = [ 2.2,  0, -1,  0.055, 1.e-9, 'cosmo'] # A_s
data.parameters['custom3']      = [  1.,  0, -1, 0.0074,     1, 'cosmo'] # n_s
```

Notice that since in our case `custom1` represents the pivot scale, it is passed as a (non-varying) argument, instead of as a (varying) parameter.

In this case, one would not include the corresponding lines for the primordial parameters of CLASS: `k_pivot`, `A_s`, `n_s`, `alpha_s`, etc. They would simply be ignored.

5.7 Limitations

- So far, this mode cannot handle vector perturbations, nor isocurvature initial conditions.
- The external script knows nothing about the rest of the CLASS parameters, so if it needs, e.g., `k_pivot`, it should be either hard coded, or its value passed as one of the `custom` parameters.

Chapter 6

Updating the manual

Author: D. C. Hooper (hooper@physik.rwth-aachen.de)

This pdf manual and accompanying web version have been generated using the doxygen software (http://www.doxygen.org). This software directly reads the code and extracts the necessary comments to form the manual, meaning it is very easy to generate newer versions of the manual as desired.

6.0.1 For CLASS developpers:

To maintain the usefulness of the manual, a new version should be generated after any major upgrade to CLASS. To keep track of how up-to-date the manual is the title page also displays the last modification date. The manual is generated automatically from the code, excepted a few chapters written manually in the files

```
README.md
doc/input/chap2.md
doc/input/chap3.md
doc/input/mod.md
external_Pk/README.md
```

You can update these files, or add new ones that should be declared in the INPUT= field of doc/input/doxyconf.

Generating a new version of this manual is straightforward. First, you need to install the doxygen software, which can be done by following the instructions on the software's webpage. The location where you install this software is irrelevant; it doesn't need to be in the same folder as CLASS. For Mac OSX, homebrew users can install the software with brew install doxygen --with-graphviz.

Once installed, navigate to the class/doc/input directory and run the first script

```
.  make1.sh
```

This will generate a new version of the html manual and the necessary files to make the pdf version. Unfortunately, doxygen does not yet offer the option to automatically order the output chapters in the pdf version of the manual. Hence, before compiling the pdf, this must be done manually. To do this you need to find the refman.tex file in class/doc/manual/latex. With this file you can modify the title page, headers, footers, and chapter ordering for the final pdf. Usually we just make two things: add manually the line

```
\vspace*{1cm}
{\large Last updated \today}\\
```

after

```
{\Large C\+L\+A\+SS M\+A\+N\+U\+AL }\\
```

and move manually the chapters "`The external Pk mode`" and "`Updating the manual`" to the end, after the automatically generated part. Once you have this file with your desired configuration, navigate back to the class/doc/input directory, and run the second script

```
. make2.sh
```

You should now be able to find the finished pdf in `class/doc/manual/CLASS_MANUAL.pdf`. Finally you can commit the changes to git, but not all the content of `doc/` is necessary: only `doc/README`, `doc/input/` and `doc/manual/CLASS_MANUAL.pdf`. Since version 2.8, we are not committing anymore `doc/manual/html/` because it was too big (and complicating the version history): users only get the PDF manual from git.

As a final comment, doxygen uses two main configuration files: `doxyconf` and `doxygen.sty`, both located in class/doc/input. Changes to these files can dramatically impact the outcome, so any modifications to these files should be done with great care.

Index