Restructuring of the nonlinear and spectra modules in v2.8

J. Lesgourgues

November 22, 2019

Between v2.7 and v2.8, there has been a complete restructuring of internal functions in the `spectra.c` and `nonlinear.c` modules. This will generate punctually a bit of pain for some of the `class` developers, because the next merging of their own branch with `class` v2.8 will generate several conflicts, that this document helps resolving. However this restructuring - which was planned since a long time - was really necessary in order to go on with the development of `class` on a healthier basis, with a more concise, readable and efficient code.

Goals of the restructuring.

1. To move several functions that had been placed in `spectra.c` since the first release in 2011, without a clear logical reason, towards other modules to which they logically belong, namely `perturbations.c` and `nonlinear.c`. This leads to a drastic reduction of redundancy throughout the code.

2. To rewrite a large part of `nonlinear.c` in a clearer, more concise and readable style. Essentially, everything in `nonlinear.c` has been rewritten excepted some functions belonging to the `halofit` and `HMcode` algorithms.

Modules affected by the restructuring.
Are affected: a small part of `perturbations.c`, most of `nonlinear.c`, a large fraction of `spectra.c`, a small part of `output.c`, and just a few lines in the python wrapper files `python/cclassy.pxd` and `python/classy.pyx`. Note that v2.8 contains lots of other new features compared to v2.7: `HMcode`, the N-body gauge, a new management of precision parameters, etc. Thus the conflicts that will appear when merging various branches with v2.8 are not all related to the restructuring discussed here. In particular, the N-body gauge leads to changes in `background.c` (e.g. new calculation and storing of `rho_tot`, `p_tot`) and `perturbations.c` (e.g. new source functions `H_T_Nb_prime`, and new correction terms for all transfer functions of the type `delta_i` and `theta_i`).

Future renaming of two modules.
The new version defines better the true physical role of four central modules in `class`. In the future, this role will be better described with new names for `nonlinear.c` and `spectra.c`. We did not change the module names in v2.8 in order to avoid imposing too many changes at the same time. The renaming will be done softly in a future version, once everybody had time to update their own branches by merging with of v2.8. While the current restructuring may generate a bit of work on the developer side, the renaming will be a trivial and painless thing, because we will circulate a script able to convert any branch from the old to the new naming scheme and vice-versa. Although this is not implemented yet, we summarize the expected renaming scheme in the following table, while clarifying the role that each of the four modules is now playing.

Old name	New name	Role & Target
perturbations.c	unchanged	Computing all sources and transfer functions in Fourier space, $S(k,\tau)$
nonlinear.c	fourier.c	Computing two-point statistics in Fourier space: $P_{\rm L}(k,z)$, $P_{\rm NL}(k,z)$, $\sigma(R,z)$, etc.
transfer.c	unchanged	Computing all transfer functions in harmonic space $\Delta_l(k)$
spectra.c	harmonic.c	Computing two-point statistics in harmonic space, C_ℓ^{XY}

Keeping full compatibility with previous versions
There are two types of `class` developers: those who only call external functions and functions in the python wrapper from their own codes/scripts, and those who do modifications to the internal structure of the `class` modules. The first category will not be impacted by the change. Indeed:

1. we have not removed, changed the name or changed the argument list of any external functions (the functions at the beginning of each module, like for instance `spectra_pk_at_z()`). Simply, some of the functions used to contain some actual lines of codes, while they are now just pointing to a new function in another modules. This is the case for all the functions in the `spectra.c` module dealing with some transfer functions $T_X(k,z)$, power spectra $P_X(k,z)$ or integrated quantities like $\sigma(R,z)$. They are now pointing to new functions in the `perturbations.c` or `nonlinear.c` module. They are still working as before, but they write a warning in the standard output, that says that they are deprecated and that gives the name of the new functions that one should now try to use. If you find these warnings irritating, you can de-activate them by just commenting the corresponding lines in `spectra.c` and recompiling.

2. we have not changed the name nor the argument list of any functions in the python wrapper. Simply, some functions in `python/classy.pyx` (like for instance `get_pk`, `get_pk_array`, etc.) now contain a call to the new functions instead of the deprecated functions.

If you are in this first category of developers, you may stop reading this document, because you will be blind to the changes. The second category of users, and in particular those who had modified `nonlinear.c` or `spectra.c`, will have to solve a few issues when they will merge with v2.8. The rest of this document is for them and will help to solve conflicts easily.

Tips for merging various older branches into v2.8
If you want to do your merging quickly and without a full understanding of all the restructuring, just read this section, and ignore the rest of this document. This section should be self-contained. But if you want to do your merging while having full control and deep undertsanding of what is going on, you may wish to read first the sections after this one, and come back here later.

Your next merge is likely to generate a moderate amount of conflicts in `background.c` (related to other features like the N-body gauge) and in `perturbations.c` (related both to the restructuring and other things). If you had not modified `nonlinear.c`, `output.c` and `python/classy.pyx`, you will get no or very few conflicts there, because the new files will overwrite your old ones. The main trouble should be with `spectra.c` and `spectra.h`. The module has changed so much that `git` will get lost and will produce conflicting files for these two that are a total mess, impossible to fix region by region. This section suggests to avoid this by not merging the new `spectra.c` and `spectra.h` in your own ones, but instead, by overwriting them, while possibly propagating your changes afterwards. We suggest to follow these steps:

1. Clone a fresh version of your branch in a new directory where you will do the merge. During or

after the merge, avoid pushing your commits too quickly to your repository. Then, if things go wrong, you can always erase this directory and redo everything from scratch.

2. If you have not merged any recent released version in your branch, first merge 2.7.2 in your branch, in order to have less things to do when merging with 2.8.x.

3. Once this is done, check the differences between your own `spectra.c` and the one of 2.7.2, with a `diff` or `git diff`. Store the output in a file, e.g. `spectra.c.diff`. Do the same with `spectra.h` and store the change in a `spectra.h.diff` You could be in one of these cases:

   (A) no differences or a few very trivial ones. Then the next steps will go fast!

   (B) differences due to the fact that you have added some new specie `xx`. Then you probably have new lines that enable to output the transfer functions for these quantities, similar to:

   (a) in `spectra_indices()`:

   ```
   class_define_index(psp->index_tr_delta_xx,ppt->has_source_delta_xx,index_tr,1);
   ```

   (b) in `spectra_matter_transfers()`:

   ```
   if (ppt->has_source_delta_xx == _TRUE_) {
       delta_i = ppt->sources[index_md]
           [index_ic * ppt->tp_size[index_md] + ppt->index_tp_delta_xx]
           [(index_tau-psp->ln_tau_size+ppt->tau_size) * ppt->k_size[index_md] + index_k];
       psp->matter_transfer[((index_tau*psp->ln_k_size + index_k)
           * psp->ic_size[index_md] + index_ic) * psp->tr_size + psp->index_tr_delta_xx]
           = delta_i;
       delta_rho_tot += rho_i * delta_i;
   }
   ```

   (c) in `spectra_output_tk_data()`:

   ```
   class_store_double(dataptr,tk[psp->index_tr_delta_xx],ppt->has_source_delta_xx,storeidx);
   ```

   (d) in `spectra_output_tk_titles()`:

   ```
   class_store_columntitle(titles,"d_xx",pba->has_xx);
   ```

   or equivalently

   ```
   class_store_columntitle(titles,"d_xx",ppt->has_source_delta_xx);
   ```

   These changes do not need to stay in the new `spectra.c`, since the module does not deal with transfer functions anymore! Just keep this information in your log. We will see later what to do with it.

   (C) several other differences. Then you will have to read very carefully the following pages and think where these other changes should end up after the merge: `perturbations.c`, `nonlinear.c` or `spectra.c`?

4. Now, get a version of 2.8 cloned somewhere, then go to your own branch, and merge 2.8 inside your branch. The code will report some conflicts in a few files, and in particular in `spectra.c` and `spectra.h`.

5. Do not try to fix the conflicts in `spectra.c` and `spectra.h`, but overwrite these files brutally with their 2.8 version, using `git checkout --theirs source/spectra.c include/spectra.h`. No worries: you are keeping track of your changes in the file `spectra_diff.log`.

6. Fix the conflicts in other files. That should not be harder than with previous releases of new versions. (still it will probably be the longest step in this list).

7. Go back to `spectra.c.diff` and `spectra.h.diff`. If you were in case **(A)**, you have nothing to do to the new `spectra.c` and `spectra.h`, or maybe very trivial things. If you are in case **(B)**, you need to propagate the changes that you see in `spectra.c.diff`, but not to the new `spectra.c`. They should go instead to the new `perturbations.c`. Taking the same example as above:

(a) the change in spectra_indices() does not need not be propagated at all. Indeed, you have already defined a ppt->index_tp_delta_xx somewhere in perturbations.c, and there are no more indices psp->index_tr_...

(b) the change in spectra_matter_transfers() does not need to be propagated either. You should have already something equivalent in perturb_sources():

```
if (ppt->has_source_delta_xx == _TRUE_) {
    _set_source_(ppt->index_tp_delta_xx) = y[ppw->pv->index_pt_delta_xx];
}
```

However you need to propagate a change coming from the N-body gauge option. Now, when a flag is switched on, the transfer functions computed either in the newtonian or synchronous gauge are sent to the output after a conversion to the N-body gauge. In order to keep this option working, you must add a line like:

```
if (ppt->has_source_delta_xx == _TRUE_) {
    _set_source_(ppt->index_tp_delta_xx) = y[ppw->pv->index_pt_delta_xx]
        + (3.*a_prime_over_a)*theta_over_k2; // N-body gauge correction;
}
```

where the quantity to be put in the parenthesis should be $-\bar{\rho}'_X/\bar{\rho}_X$, that could be equal to 3.*a_prime_over_a (non-relativistic species) or 4.*a_prime_over_a (relativistic species) or more complicated. This is valid for density transfer functions. For velocity transfer functions, as you can see in the new code, you should just add theta_shift to your new source. When the N-body gauge option is switched off, both theta_over_k2 and theta_shift are null, and nothing changes.

(c) the change in spectra_output_tk_data() must be propagated to perturb_output_data():

```
class_store_double(dataptr,tk[ppt->index_tp_delta_xx],ppt->has_source_delta_xx,storeidx);
```

(note the difference with respect to the old line: it is not a psp->index_tr_... anymore, but a ppt->index_tp_...)

(d) the change in spectra_output_tk_titles() must be propagated to perturb_output_titles():

```
class_store_columntitle(titles,"d_xx",ppt->has_source_delta_xx);
```

You can also propagate some of the changes seen in spectra.h.diff into spectra.h, but only if they have to do with the calculation of the C_l's. If they have to do with the calculation of the $P(k,z)$ and $T(k,z)$ for new species, they don't need to be propagated at all, since the spectra.c module does not deal with these anymore, while perturbations.h should already contain the necessary indices.

8. At this point, you should essentially be done, apart from a bit of debugging...

The following is more technical and meant for advanced developpers.

New calculation of Fourier transfer functions

By Fourier transfer functions, we refer to transfer functions in Fourier space for density fluctuations $\delta_X(k,\tau)$, velocities $\theta_X(k,\tau)$, and metric fluctuations. They are computed when the output list contains mPk and/or vTk.

- *Before v2.8*: a grid for these quantities was computed and stored in the perturbation structure by perturb_sources(). Later, spectra_matter_transfers() had the task of duplicating this grid in the spectra structure. The function spectra_tk_at_z() was designed to interpolate the grid at the values of time τ requested for the output. The spectra.c module also

contained two functions spectra_output_tk_data() and spectra_output_tk_titles() that were called either from the output.c module or from the wrapper. spectra_output_tk_titles() would return headers for output files or output dictionaries. spectra_output_tk_data() would loop through the value of time requested for the output, get the transfer functions at these times with spectra_tk_at_z() and return them in a pointer. Finally, the function output_tk() or the python/classy.pyx function get_transfer() would call spectra_output_tk_titles() to get headers, spectra_output_tk_data() to get values, and write them either in output files or in output dictionaries.

- *In v2.8*: the useless duplication step in the spectra.c module is suppressed, and everything is done in perturbations.c, following the overall principle that *all the things that a module can do (without changing its overall purpose) should be done there*, rather than being deferred to another module. The grid of transfer functions is still computed in perturb_sources() and stored in the perturbation structure. The function perturb_sources_at_tau() is able to interpolate the grid at any time (this function existed before but was never used, now it is playing its normal role). The functions perturb_output_data() and perturb_output_titles() play the role previously attributed to spectra_output_tk_data() and spectra_output_tk_titles(). In particular, perturb_output_data() gets the transfer functions at the values of time requested in output using perturb_sources_at_tau(). The functions output_tk() or the python/classy.pyx function get_transfer() have the same names and arguments as before, but now they call perturb_output_titles() to get headers, perturb_output_data() to get values. Thus the whole process is more condensed and straightforward.

New calculation of linear $P_{\rm L}(k,z)$

- *Before v2.8*: the linear power spectrum was computed twice!

  1. The full calculation leading to the output was in spectra.c. The function spectra_k_and_tau() defined the grid of output (k,τ) values, while spectra_pk() did the actual calculation of $P_L(k,\tau)$ including all options and components (_m, _cb, total, and decomposed in adiabatic / isocurvature modes) and stored the result in the spectra structure. The functions spectra_pk_at_z() and spectra_pk_at_k_and_z() were able to interpolate in that grid, and to return $P_L(k,z)$ at arbitrary (k,z). The function spectra_fast_pk_at_kvec_and_zvec() returned $P_L(k,z)$ for an input grid of (k,z) values.

     The function output_pk() used the local functions output_open_pk_file() and output_one_line_of_pk() to handle the opening and the writing in output files, and called spectra_pk_at_z() to get the values to be written in the files.

     Finally, python/classy.pyx contained several functions pk(), pk_cb(), pk_lin(), pk_lin_cb(), get_pk(), get_pk_cb(), get_pk_lin(), get_pk_lin_cb(), get_pk_and_k_and_z() that all extracted the output using spectra_pk_at_k_and_z(). It also contained get_pk_array(), get_pk_cb_array(), that were wrappers for spectra_fast_pk_at_kvec_and_zvec().

  2. The linear power spectrum was actually needed by Halofit in nonlinear.c before reaching spectra.c. Thus there was a redundant calculation of $P_L(k,\tau)$ inside the function nonlinear_pk_l() (although a simplified one, without the decomposition into adiabatic / isocurvature modes). The results were stored in local variables passed to Halofit rather than in the structure, thus they were not reused afterwards. This redundancy was not elegant, but the impact on performances was minor (the function nonlinear_pk_l() was very fast to evaluate).

- *In v2.8*: the useless duplication step in the spectra.c module is suppressed, and everything is done in nonlinear.c, following the overall principle that *all the things that a module can do (without changing its overall purpose) should be done there*, rather than being deferred to another module.

Of course it sounds weird to compute the linear spectrum in a module called nonlinear.c, but this will sound better when the module will be renamed fourier.c. It is important to note that in the old nonlinear.c and spectra.c, there were lots of arrays and functions with a suffix _m for total matter perturbations and _cb for CDM+baryon perturbations. This is not the case anymore, because the code loops everywhere over an index index_pk that can refer to _m-type or _cb-type perturbations. Thus many arrays have one more dimension than before, e.g. pk[index_pk][...], and many functions have an extra input argument index_pk .

The grid of $P_L(k,z)$ values is now computed in nonlinear_pk_linear() and stored in the nonlinear structure. Inside nonlinear_pk_linear(), the step of extracting transfer functions for the matter or baryon+cdm fluctuations is deferred to a function nonlinear_get_source(). This function also knows how to extrapolate the results at higher k than pre-computed values: this functionality is needed for HMcode. Several functions can then read and interpolate the grid of pre-computed $P_L(k,z)$: nonlinear_pk_at_z() does the same as the old spectra_pk_at_z() but for only one input value of index_pk, while nonlinear_pks_at_z() gives the results for all the available index_pk types. The same holds for nonlinear_pk_at_k_and_z() and nonlinear_pks_at_k_and_z() that replace spectra_pk_at_k_and_z(). The function nonlinear_pks_at_kvec_and_zvec() replaces spectra_fast_pk_at_kvec_and_zvec().

In spectra.c, nothing else is done with the linear power spectrum, but we have kept the (now deprecated) functions spectra_pk_at_z(), spectra_pk_at_k_and_z() and spectra_fast_pk_at_kvec_and_zvec(), that just encapsulate the new functions nonlinear_pks_at_z(), nonlinear_pks_at_k_and_z() and nonlinear_pks_at_kvec_and_zvec().

Later, in the output.c module, the output_pk() function works as before, using the local functions output_open_pk_file() and output_one_line_of_pk() to handle the opening and the writing in output files, but now calling nonlinear_pk_at_z() to get the values to be written in the files.

Finally, in python/classy.pyx, the functions pk(), pk_cb(), pk_lin(), pk_lin_cb(), get_pk(), get_pk_cb(), get_pk_lin(), get_pk_lin_cb(), get_pk_and_k_and_z() are still there but now extracting data through nonlinear_pk_at_k_and_z(), while get_pk_array(), get_pk_cb_array(), are some wrappers for nonlinear_pks_at_kvec_and_zvec().

New calculation of non-linear $P_{\rm NL}(k,z)$

- *Before v2.8*: the goal of nonlinear.c was not to compute $P_{\rm NL}(k,z)$ but only the non-linear rescaling factors $R_{\rm NL}(k,z) = (P_{\rm NL}(k,z)/P_{\rm L}(k,z))^{1/2}$. These are useful not only for the final output of $P_{\rm NL}(k,z)$, but also for computing all the large scale structure C_l's (number count, galaxy shear, CMB lensing) including non-linear corrections. Thus $P_{\rm NL}(k,z)$ was computed for a first time in nonlinear.c, but only $R_{\rm NL}(k,z)$ was stored in the nonlinear structure. $P_{\rm NL}(k,z)$ was computed a second time in spectra.c, given $P_{\rm L}(k,z)$ and $R_{\rm NL}(k,z)$, such that there was some redundancy.

  More precisely, nonlinear_init() would call nonlinear_halofit() to get $P_{\rm NL}(k,z)$, and then would compute and store $R_{\rm NL}(k,z)$ in the nonlinear structure. Then, in spectra.c, the same spectra_pk() in charge of computing $P_{\rm L}(k,z)$ would also compute $P_{\rm NL}(k,z)$ if requested, and store it in a grid in the spectra structure. This grid could be read and interpolated by spectra_pk_nl_at_z() and spectra_pk_nl_at_k_and_z(). The function spectra_fast_pk_at_kvec_and_zvec(), already seen before, would also work for the nonlinear spectrum, thanks to an input flag set either to linear or non-linear.

  In output.c, the function output_pk_nl() handled nonlinear spectra and extracted the data using spectra_pk_nl_at_z().

  Finally, python/classy.pyx contained several functions pk(), pk_cb(), get_pk(), get_pk_cb(), get_pk_and_k_and_z() that would return the linear spectrum if non-linear corrections were not requested, and would otherwise return $P_{\rm NL}(k,z)$ using spectra_pk_nl_at_k_and_z(). It also contained get_pk_array(), get_pk_cb_array(), that were wrappers for spectra_fast_pk_at_kvec_and_zvec().

- *In v2.8*: the re-computing of $P_{\rm NL}(k,z)$ in the spectra.c module is suppressed, and everything is done in nonlinear.c, following the overall principle that *all the things that a module can do (without changing its overall purpose) should be done there*, rather than being deferred to another module.

  For the calculation of $P_{\rm NL}(k,z)$, nonlinear_init() calls either nonlinear_halofit() or nonlinear_hmcode(). At the end of this step, both $P_{\rm NL}(k,z)$ and $R_{\rm NL}(k,z)$ are stored in the nonlinear structure. During these steps, the code computes all the requested types of spectra, _m and _cb, by looping over index_pk. The table of $P_{\rm NL}(k,z)$ values can be read and interpolated by the very same functions as for the linear spectrum: nonlinear_pk_at_z(), nonlinear_pks_at_z(), nonlinear_pk_at_k_and_z(), nonlinear_pks_at_k_and_z(), nonlinear_pks_at_kvec_and_zvec(), because they all have an input flag pk_output set either to linear or non-linear.

  In spectra.c, nothing else is done with the non-linear power spectrum, but we have kept the (now deprecated) functions spectra_pk_nl_at_z(), spectra_pk_nl_at_k_and_z() and spectra_fast_pk_at_kvec_and_zvec() that just encapsulate the new functions nonlinear_pks_at_z(), nonlinear_pks_at_k_and_z() and nonlinear_pks_at_kvec_and_zvec() called with the flag pk_output set to non-linear.

  Later, in the output.c module, the function output_pk() now handles both linear and non-linear spectra, because it has the input flag pk_output. It extracts $P_{\rm NL}(k,z)$ using nonlinear_pk_at_z() with pk_output set to non-linear.

  Finally, in python/classy.pyx, the functions pk(), pk_cb(), pk_lin(), pk_lin_cb(), get_pk(), get_pk_cb(), get_pk_lin(), get_pk_lin_cb(), get_pk_and_k_and_z() are still there but now extracting data through nonlinear_pk_at_k_and_z(), while get_pk_array(), get_pk_cb_array(), are some wrappers for nonlinear_pks_at_kvec_and_zvec().

New calculation of $\sigma(R,z)$ and similar derived quantities

- *Before v2.8*: spectra.c contained two functions spectra_sigma() and spectra_sigma_cb(), that computed $\sigma(R,z)$ for all requested types (total matter, cdm+baryons) by integrating the $P_{\rm L}(k,z)$ stored in the spectra structure over k for a fixed z. These functions could be called from anywhere, but they were called at least inside spectra_init() to compute $\sigma_8 \equiv \sigma(8,0)$ and possibly $\sigma_{8\rm cb}$, and to store them in the spectra structure.

  python/classy.pyx contained some functions sigma() and sigma_cb() that were some wrappers for spectra_sigma() and spectra_sigma_cb(). It also contained two functions sigma8() and sigma8_cb() extracting the values stored in the spectra structure.

- *In v2.8*: everything related to σ's and similar quantities is now done in nonlinear.c, which allows to call the new functions from the Halofit or HMcode algorithm.

  The actual calculation is done in an internal function nonlinear_sigmas() that takes in input a value of R and a tabulated $P_{\rm L}(k)$. The redshift is not explicitly passed in input, because $P_{\rm L}(k)$ is supposed to be a linear spectrum at the redshift of interest. The input $P_{\rm L}(k)$ does not need to be tabulated on the same values of k used for storing quantities in the nonlinear structure: it can be better sampled, extrapolated to larger k's, etc. These features are very useful for HMcode, that needs some very precise σ's, and uses fine sampling and extrapolation for that. nonlinear_sigmas() also takes in input a variable sigma_output which belongs to a new enumeration: its value refers to different things to compute, either output_sigma for the plain $\sigma(R)$, output_sigma_prime for $\sigma'(R)$, or output_sigma_disp for the velocity dispersion used by HMcode. The user can easily enlarge this list to compute more derived quantities.

  However, the function that is the most exposed to the users is the external function nonlinear_sigmas_at_z(). It has a more standard argument list. Instead of a linear power spectrum, nonlinear_sigmas_at_z() takes a redshift in input. Then it gets $P_{\rm L}(k,z)$ from nonlinear_pk_at_z(), and it passes it to nonlinear_sigmas() to get the result (still with the same options sigma_output, and with an

input index index_pk to get either the _m or _cb quantities). In nonlinear_init() there is a call to nonlinear_sigmas_at_z() to compute σ_8 and possibly $\sigma_{8\mathrm{cb}}$, and store them in the nonlinear structure.

Finally, nonlinear_sigma_at_z() is another internal function meant just for compatibility with previous versions, but nonlinear_sigmas_at_z() should be preferred due to its better list of arguments. nonlinear_sigma_at_z() has roughly the same arguments as the deprecated spectra_sigma() and spectra_sigma_cb(). It also uses nonlinear_sigmas() for the calculation of $\sigma(R, z)$ (but without the option to get other quantitites like σ').

In spectra.c, nothing else is done with the σ quantities, but we have kept the (now deprecated) functions spectra_sigma() and spectra_sigma_cb(), now just encapsulating nonlinear_sigma_at_z().

The wrapper python/classy.pyx still contains the same functions sigma() and sigma_cb(), that are now wrappers for nonlinear_sigmas_at_z(). It also stills contain the functions sigma8() and sigma8_cb() extracting the values stored in the nonlinear structure.

What remains in spectra.c?
The new spectra.c only contains the calculations related to all C_l spectra, plus a list of deprecated functions that encapsulate the new improved functions located in nonlinear.c. The deprecated functions are:
spectra_pk_at_z()
spectra_pk_at_k_and_z()
spectra_pk_nl_at_z()
spectra_pk_nl_at_k_and_z()
spectra_fast_pk_at_kvec_and_zvec()
spectra_sigma()
spectra_sigma_cb()