CLASS Exercises

Julien Lesgourgues & Thomas Tram
julien.lesgourgues@cern.ch, thomas.tram@epfl.ch

April 3, 2014

Written solution will be made available progressively, after each session.

Exercise Ia: Comparison between lensed and unlensed temperature spectrum

Check the difference between the lensed and unlensed C_l^{TT} of scalars, to see the effect of smoothing of the maxima and minima of the spectrum, and the extra damping induced by lensing on small scales.

Exercise Ib: Comparison between lensed and unlensed BB spectrum

Check the difference between the lensed and unlensed C_l^{BB} in presence of tensor modes, to see that B modes are dominated by lensing on small scales. Use $r = 0.2$ like in BICEP results! To fix this value of the tensor to scalar ratio, just add this line to your input file: `r = 0.2`

Exercise Ic: Comparison between adiabatic and isocurvature CMB spectra

Check the difference between the unlensed C_l^{TT} of scalar modes for adiabatic and CDM isocurvature (CDI) initial conditions (with index $n_{cdi} = 1$), to check that peaks are suppressed in amplitude and shifted in scale. In order to enhance the isocurvature spectrum, you may use the cdi isocurvature fraction `f_cdi = 2`, together with `n_cdi = 1`.

Do the same with NID isocurvature modes (with index $n_{nid} = 1$) to check that the suppression in amplitude is less pronouced and the phase of NID and CDI are different. To enhance the isocurvature spectrum, you may use `f_nid = 4`, together with `n_nid = 1`.

Exercise Id: Comparison between linear and non-linear matter spectrum

Check the difference between the linear and non-linear matter power spectrum at $z = 0$ and $z = 2$, to see that at low redshift non-linear corrections are present on larger scales.

Solution Ia:

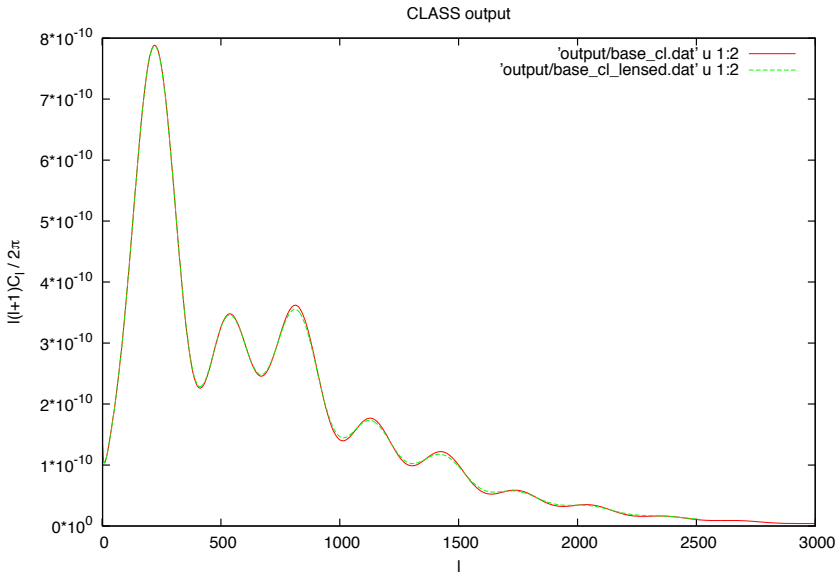
The solution of these exercises is of course not unique. You could e.g. run with the input file:

```
output = tCl,pCl,lCl
lensing = yes
root = output/base_
#
background_verbose = 1
thermodynamics_verbose = 1
perturbations_verbose = 1
transfer_verbose = 1
primordial_verbose = 1
spectra_verbose = 1
nonlinear_verbose = 1
lensing_verbose = 1
output_verbose = 1
```

Then you may plot with

```
python CPU -x output/base_cl.dat output/base_cl_lensed.dat
```

producing the figure



or in matlab:

```
plot_CLASS_output({'output/base_cl.dat','output/base_cl_lensed.dat'},'TT')
```

Solution Ib:

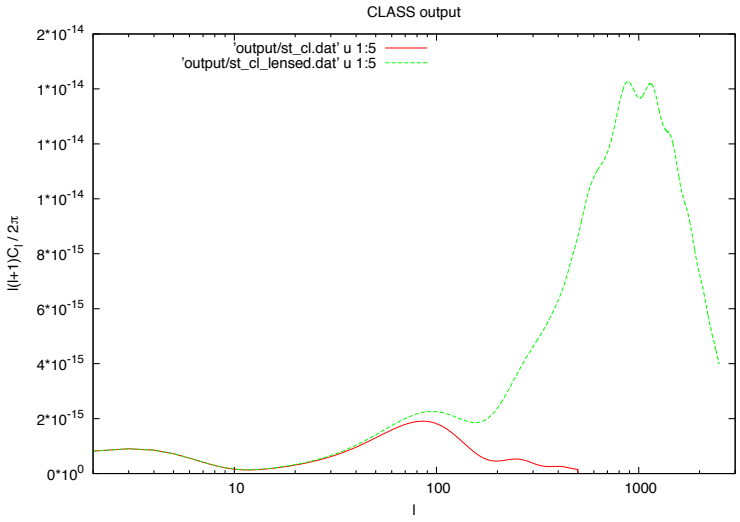
You could e.g. run with the input file:

```
modes = s,t
r = 0.2
output = tCl,pCl,lCl
lensing = yes
root = output/st_
#
background_verbose = 1
thermodynamics_verbose = 1
perturbations_verbose = 1
transfer_verbose = 1
primordial_verbose = 1
spectra_verbose = 1
nonlinear_verbose = 1
lensing_verbose = 1
output_verbose = 1
```

Then you may plot with

```
python CPU -colnum 5 -x -t cl_log output/st_cl.dat output/st_cl_lensed.dat
```

producing the figure



or in matlab:

```
plot_CLASS_output({'output/st_cl.dat','output/st_cl_lensed.dat'},'BB')
```

Solution Ic:

You could e.g. run with the two input files:

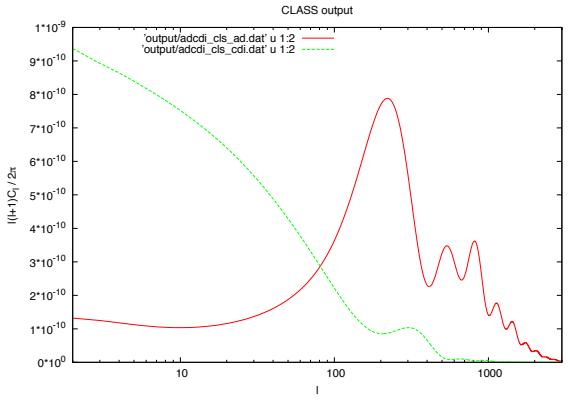
```
ic = ad cdi
f_cdi = 2.
n_cdi = 1.
output = tCl,pCl,lCl
lensing = yes
root = output/adcdi_
#
background_verbose = 1
thermodynamics_verbose = 1
perturbations_verbose = 1
transfer_verbose = 1
primordial_verbose = 1
spectra_verbose = 1
nonlinear_verbose = 1
lensing_verbose = 1
output_verbose = 1
```

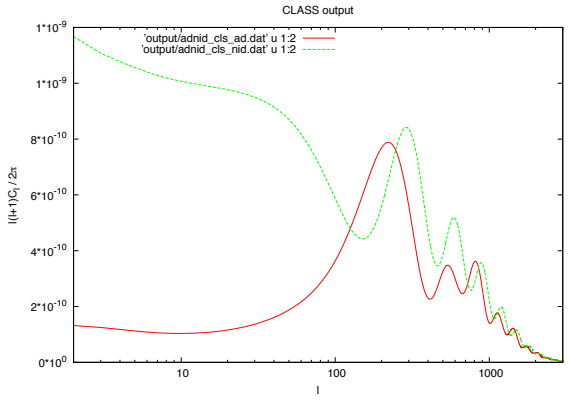
```
ic = ad nid
f_nid = 4.
n_nid = 1.
output = tCl,pCl,lCl
lensing = yes
root = output/adnid_
#
background_verbose = 1
thermodynamics_verbose = 1
perturbations_verbose = 1
transfer_verbose = 1
primordial_verbose = 1
spectra_verbose = 1
nonlinear_verbose = 1
lensing_verbose = 1
output_verbose = 1
```

Then you may plot with

```
python CPU -x -t cl_log output/adcdi_cls_ad.dat output/adcdi_cls_cdi.dat
python CPU -x -t cl_log output/adnid_cls_ad.dat output/adnid_cls_nid.dat
```

producing the figures





or in matlab:

```
plot_CLASS_output({'output/adcdi_cls_ad.dat','output/adcdi_cls_cdi.dat'},'TT')
plot_CLASS_output({'output/adnid_cls_ad.dat','output/adnid_cls_nid.dat'},'TT')
```

Solution Id:

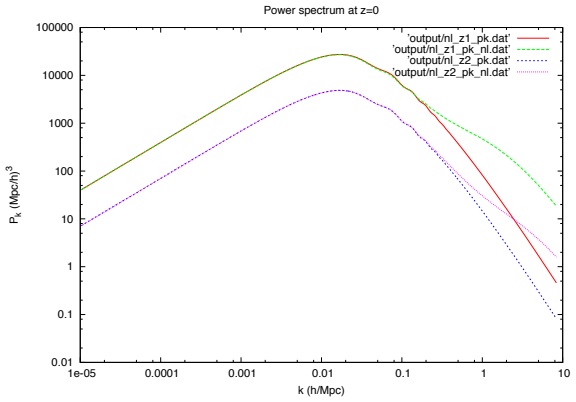
You could e.g. run with the input file:

```
output = tCl, mPk
non linear = halofit
z_pk = 0,2
P_k_max_h/Mpc = 10.
root = output/nl_
#
background_verbose = 1
thermodynamics_verbose = 1
perturbations_verbose = 1
transfer_verbose = 1
primordial_verbose = 1
spectra_verbose = 1
nonlinear_verbose = 1
lensing_verbose = 1
output_verbose = 1
```

Then you may plot with

```
python CPU -x output/nl_z1_pk.dat output/nl_z1_pk_nl.dat output/nl_z2_pk.dat
 output/nl_z2_pk_nl.dat
```

producing the figure



or in matlab:

```
plot_CLASS_output({'output/nl_z1_pk.dat', 'output/nl_z1_pk_nl.dat', 'output/nl_z2_pk.dat',
 'output/nl_z2_pk_nl.dat'},'P')
```

Exercise IIa: Printing and plotting background quantities

Reproduce this plot from the Dodelson book on *Modern Cosmology*, using the plotting software of your choice (gnuplot, IDL, matlab, python, etc...). Note that Dodelson plots the three cosmological distances d_X in units of $[1/H_0]$, which is equivalent to saying that he plots the dimensionless products $d_X H_0$. We recall that $H_0 = h/3000$ Mpc^{-1}.

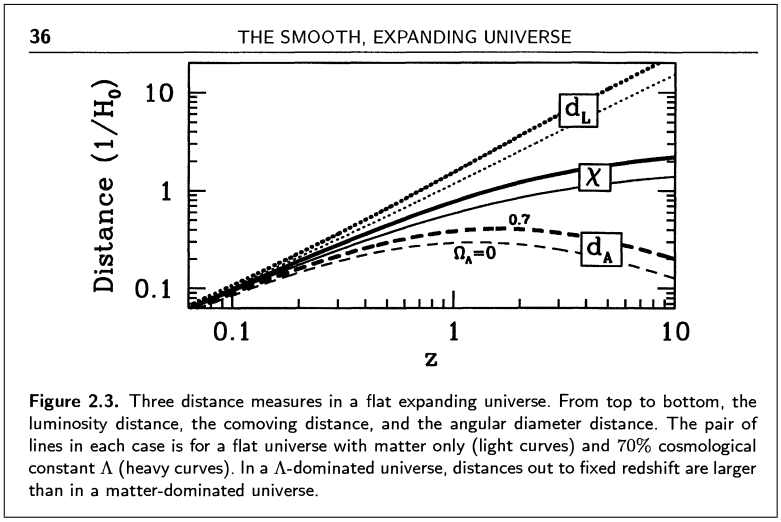


Figure 2.3. Three distance measures in a flat expanding universe. From top to bottom, the luminosity distance, the comoving distance, and the angular diameter distance. The pair of lines in each case is for a flat universe with matter only (light curves) and 70% cosmological constant Λ (heavy curves). In a Λ-dominated universe, distances out to fixed redshift are larger than in a matter-dominated universe.

Exercise IIb: Adding a species in the background module

Introduction. Many types of cosmological species are already available in CLASS, but sometimes it will be necessary to add another species. In this exercise you will add a fluid with equation of state parameter w to CLASS. A general fluid has already been implemented in CLASS with equation of state parameter $w = w_0 + w_a(1 - a/a_0)$ and arbitrary sound speed c_s^2. Nevertheless the exercise is not entirely pointless, because there could be situations where the model contains (or can be modelled by) two uncoupled fluids with different equation of state parameters.

Reading and storing new parameters. We need CLASS to read two additional parameters from the `.ini` file: `Omega_efld` and `w_efld`. (`efld` $\equiv$ extra fluid). First add the two new parameters (`Omega0_efld`, `w_efld`) to the background structure defined in `background.h`. Then open `input.c` and scroll down to `input_init()` which begins at around line 139. We need to add a couple of new lines to this function, and in principle they could be added almost anywhere inside this function. However, all the species are written in the same order all over the code, so it is nice to add the new species to the "order of species" and then stick to this convention. The order is {photons, baryons, ultrarelativistic species, cold dark matter, non-cold dark matter, curvature, lambda, fluid}. Since we are adding another fluid, it would fit nicely on either side of the existing fluid. However in this particular function, the extra fluid part must go before the lambda and fluid part, since the values assigned here will depend on `Omega_tot`.

Since everything related to the fluid species has `_fld` at the end, you can just search for `_fld` in your editor. Just before the line

```
/* Omega_0_lambda (cosmological constant), (...) */
```

add a comment about the species you are adding. Now we must read the two input values of (`Omega0_efld`, `w_efld`): to this end we will utilise the macro[1] `class_read_double(name,destination)`. `name` should be a string, e.g. `"Omega_efld"` and `destination` should then be `pba->Omega0_efld`. We must also add `Omega0_efld` to the total density:

```
Omega_tot += pba->Omega0_efld;
```

We must now set default values for the new parameters. Continue searching for instances of `_fld` until you find

```
pba->Omega0_fld = 0.;
```

Do the same for `pba->Omega0_efld` and put some default value of the equations of state parameter as well.

Modifying the background evolution. Many small changes need to be made to `background.c` and `background.h`. The strategy is to search the file for a similar species, in this case `_fld` and replicate (read: copy-paste-modify) the lines. Note that the evolution of the `_efld` species is uniquely determined by the scale factor, so we do not need to evolve the energy density and pressure in time. In CLASS-language, `rho_efld` and `p_efld` are known as {`A`}-variables: they are analytic functions of {`B`}-variables. (In the vanilla vase, the only `B`-variable is the scale factor.) An upcoming version of CLASS will contain a few examples of species requiring extra `B`-variables.
Intermezzo: The unit of energy densities and pressure in CLASS is defined in the following way:

$$\rho^{\rm CLASS} \equiv \frac{8\pi G}{3c^2}\rho^{\rm physical}. \tag{1}$$

$$p^{\rm CLASS} \equiv \frac{8\pi G}{3c^2}p^{\rm physical}. \tag{2}$$

What is the critical density today in CLASS units? $\rho^{\rm CLASS}_{\rm crit,0} =$
Going through `background.c`, you will realise that you also need to define a number of additional parameters to `background.h`. Be careful to change all instances of `_fld` to `_efld` if you copy-paste!

Modifying the output. The last thing we need is to modify `output.c` to output the new fluid density in the background data file. To this end we need to add 2 lines of code, utilising the macros
`class_fprintf_columntitle(file,title,condition)`
`class_fprintf_double(file,value,condition)`
Just search for these macros or `_fld` as before. You can now create a `.ini` files where one uses e.g. the parameters (`Omega_efld = 0.1`, `w_efld = 0.1667` and `root = efluid_`), to simulate the case $w = 1/6$. Run the code and check that the output `efluid_background.dat` is correctly created, with one columns for the extra fluid.

[1] A macro in C is a few lines of code which will be pasted directly into the source code by the preprocessor before compilation. The macros for reading parameters are defined in `input.h`.

Checking the result by comparing with the in-built fluid component. You can duplicate the previous `.ini` to a new one, where the same role should be played by the in-built fluid component (`Omega_fld = 0.1`, `w0_fld = 0.1667`, `wa_fld = 0.` and `root = fluid_`, while the parameter `Omega_efld` should either be set to zero, or left blank, or commented out).

Check that in this new input file, when you leave the `output` field set like in `explanatory.ini` (starting with `output = tCl,...`), your run returns an error. This is normal, and a good way to learn on the error management system in CLASS. Have a look at the error message. In which module did the error occur? You can have a look also at the exact line producing the error.

The reason for the error is that for `w0_fld>0`, the code cannot find initial conditions for the fluid perturbations in the standard way: hence the perturbation module contains a protection against this case. Since we are only interested in the background evolution, we can avoid this error by not computing output spectra. This is easily done by commenting the output line `output = tCl, ...` in the `.ini`-files. Check that the code runs correctly in that case.

By playing with other values of `w0_fld`, you can also check that values are now allowed up to 1/3. Above this value, check that there is another error message, now coming from another module. This comes from a second protection: for values `w0_fld >= 1/3`, the fluid would dominate over ordinary radiation at early times, and the initial conditions for the background evolution cannot be set in the standard way.

You should now finally compare the energy densities `rho_fld` and `rho_efld` which should be identical, for the same values of Ω and $w < 1/3$. Plot this energy density against `rho_cdm` and `rho_ur`. Your plot should be similar to figure 1.

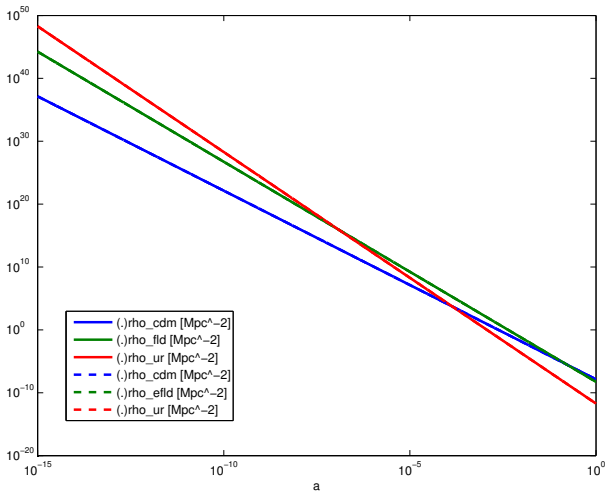


Figure 1: Energy densities of CDM, massless neutrinos and a fluid with equation of state parameter $w = 0.1667 \simeq 1/6$.

Solution of exercise IIa.

Dodelson compares two ΛCDM models with $\Omega_\Lambda = 0$ and $\Omega_\Lambda = 0.7$. Hence we can create two input files, leaving most parameters at their default values, but writing at least:

in `omega0.ini`:

```
H0 = 70.  # for this exercise, this does not matter, since in the final plot
          # the distances are in units of 1/H_0
Omega_b = 0.05
Omega_cdm = 0.95 # splitting between Omega_b and Omega_cdm
                 # does not matter provided that the sum is 1
write background = yes
root = output/omega0_
```

in `omega7.ini`:

```
H0 = 70.  # for this exercise, this does not matter, since in the final plot
          # the distances are in units of 1/H_0
Omega_b = 0.05
Omega_cdm = 0.25 # splitting between Omega_b and Omega_cdm
                 # does not matter provided that the sum is 0.3 (= 1 - Omega_Lambda)
write background = yes
root = output/omega7_
```

(Exemples of such files are available on-line in the exercise directory). After running
```
> ./class omega0.ini
> ./class omega7.ini
```
we can reproduce the Dodelson plot using our preferred plotting software. Using the MATLAB function `plot_CLASS_output()` it is enough to type

```
plot_CLASS_output({'output/omega0_background.dat','output/omega7_background.dat'},...
                  {'lum','comov. dist','ang'},...
                  'xvariable','z',...
                  'xlim',[0.08,10],...
                  'EpsFilename','Dodelson')
```

Given that all three titles have the word `dist` in common, we could also have typed `'dist'` instead of `{'lum','comov. dist','ang'}`. Using gnuplot, first go to the `output/` directory and open gnuplot by typing
```
> gnuplot
```
we can now enter:

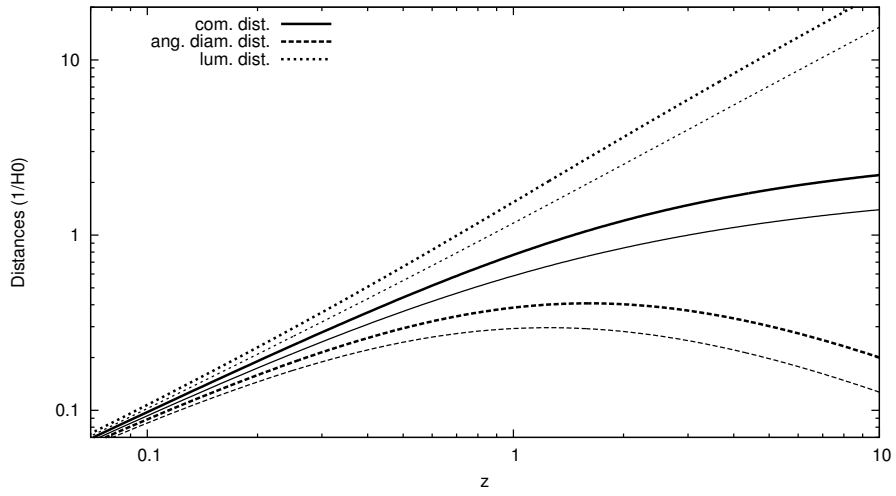
```
set logscale
x=0.7/3000
plot [0.07:10][0.07:20] 'omega7_background.dat' u 1:($5*x) w l\
'omega7_background.dat' u 1:($6*x) w l,\
'omega7_background.dat' u 1:($7*x) w l,\
```

```
'omega0_background.dat' u 1:($5*x) w l,\
'omega0_background.dat' u 1:($6*x) w l,\
'omega0_background.dat' u 1:($7*x) w l
```

or equivalently, for a better presentation, we can write a gnuplot script `dodelson.gnu` containing:

```
set term po eps
set output "dodelson.eps"
set logscale
x=0.7/3000
set size 1,0.8
set xlabel "z"
set ylabel "Distances (1/H0)"
set key top left
plot [0.07:10][0.07:20] 'omega7_background.dat' u 1:($5*x) t "com. dist." w l lt 1 lw 4,\
'omega7_background.dat' u 1:($6*x) t "ang. diam. dist." w l lt 2 lw 4,\
'omega7_background.dat' u 1:($7*x) t "lum. dist." w l lt 3 lw 4,\
'omega0_background.dat' u 1:($5*x) notitle w l lt 1 lw 2,\
'omega0_background.dat' u 1:($6*x) notitle w l lt 2 lw 2,\
'omega0_background.dat' u 1:($7*x) notitle w l lt 3 lw 2
```

(This files is available on-line in the exercise directory). After
`> gnuplot dodelson.gnu`
we get the figure `dodelson.eps` that looks like:



Solution of exercise IIb.

Here is the full list of lines that need to be added to the code in order to solve the exercise. We don't give exact line numbers: this is somewhat arbitrary and should be clear from the text of the exercise.

In `source/input.c`:

```
...
/* The extra fluid */
class_read_double("Omega_efld",pba->Omega0_efld);
class_read_double("w_efld",pba->w_efld);
Omega_tot += pba->Omega0_efld;
...
pba->Omega0_efld = 0.;
pba->w_efld = 1./3.;
...
```

In `include/background.h`:

```
...
double Omega0_efld;
double w_efld;
...
int index_bg_rho_efld;
...
short has_efld;
...
```

In `source/background.c`:

```
...
/* extra fluid with w */
if (pba->has_efld == _TRUE_) {
  pvecback[pba->index_bg_rho_efld] = pba->Omega0_efld * pow(pba->H0,2)
    / pow(a_rel,3.*(1.+pba->w_efld));
  rho_tot += pvecback[pba->index_bg_rho_efld];
  p_tot += (pba->w_efld) * pvecback[pba->index_bg_rho_efld];
}
...
if (pba->has_efld == _TRUE_) {
  Omega0_tot += pba->Omega0_efld;
}
...
```

```
pba->has_efld = _FALSE_;
...
if (pba->Omega0_efld != 0.)
  pba->has_efld = _TRUE_;
...
/* - index for extra fluid */
class_define_index(pba->index_bg_rho_efld,pba->has_efld,index_bg,1);
...
```

In `source/output.c`:

```
...
class_fprintf_columntitle(*backfile,"(.)rho_efld [Mpc^-2]",pba->has_efld);
...
class_fprintf_double(backfile,pvecback[pba->index_bg_rho_efld],pba->has_efld);
...
```

If you want to plot quantities using the matlab script `plot_CLASS_output.m`, the plot can easily be computed by `plot_CLASS_output` using the command

```
plot_CLASS_output({'output/fluid_background.dat','output/efluid_background.dat'},{'cdm','fld','ur'})
```

The output file will be `myplot.eps`, but it can be specified by adding `'EpsFilename','yourplot'` to the output command.

Exercise IVa: Comparing the two metric fluctuations $\phi(k,\tau)$ and $\psi(k,\tau)$

This exercise can be done in any of the two gauges (newtonian, synchronous), because the part of the perturbation module which plots the evolution of perturbations always writes results in the newtonian gauge (if the code uses the synchronous gauge, variables are gauge-transformed just before being printed in the output file).

- Plot the evolution of $\phi(k,\tau)$ and $\psi(k,\tau)$ versus conformal time, for the two modes $k = 0.01$/Mpc and $k = 0.1$/Mpc. Check that the two metric fluctuations coincide after the time of Hubble crossing $k\tau \sim 1$, while before there is a constant offset.
- To understand why, plot the evolution of the part of the energy-momentum tensor of massless neutrinos accounting for anisotropic pressure, multiplied by the square of the scale factor: $a^2(\bar{\rho}_\nu + \bar{p}_\nu)\sigma_\nu$. For that, you will need to modify the default output of the perturbation module: localise the line where the quantity `shear_ur` is printed in a file, and modify this line to plot $(4/3)a^2\bar{\rho}_\nu\sigma_\nu$ instead (given that equation of state of massless neutrinos is $\bar{p}_\nu = \bar{\rho}_\nu/3$)). Indication: when the code reaches this line, the vector `pvecback` is known (because it is passed among the input arguments of the function `perturb_print_variables(...)`), and it does contain all background quantities at the right time. Finally, print $a^2(\bar{\rho}_\nu + \bar{p}_\nu)\sigma_\nu$ as a function of τ.
- Check that the result of the ($\phi(k,\tau)$, $\psi(k,\tau)$) plot and of the $a^2(\bar{\rho} + \bar{p})\sigma_\nu$ are consistent with each other, given the Einstein equation (Ma & Bertschinger eq. (23d)):

$$k^2(\phi - \psi) = 12\pi G a^2(\bar{\rho} + \bar{p})\sigma_{\rm tot} \ .$$

Exercise IVb: A very simple modification of gravity

There exist several ways to parametrise modifications of gravity. For instance, people often study the effect of a function $\mu(k,\tau)$ inserted in the Poisson equation, giving in the synchronous gauge:

$$k^2\eta - \frac{1}{2}\frac{a'}{a}h' = -\mu(k,\tau)\ 4\pi G a^2 \bar{\rho}_{\rm tot}\delta_{\rm tot} \ .$$

The perturbed Einstein equations are defined in a single place, in `perturb_einstein(...)`. Localise the above equation and implement, for instance, $\mu = 1 + a^3$. Print the evolution of ϕ and ψ in the standard and modified models, and conclude that the C_l^{TT}'s should be affected only through the late ISW effect. Get a confirmation by comparing directly the C_l's (printed in the files `<root>_cl.dat`).

Exercise IVc: is the tensor temperature spectrum generated mainly at recombination, or along the line-of-sight?

By doing small modifications of `perturb_sources()`, check whether the tensor temperature spectrum $C_l^{TT,\rm tens.}$ comes mainly from photon perturbations on the last scattering surface, or from an integrated Sachs-Wolfe effect.

For this exercise you can run the full `class` code with `modes = s,t`, `output = tCl`, `l_max_tensors = 500`, `lensing = no`. In each run you can change the name of `root = output/test_` to a more specific name (e.g. `root = output/ISW_`) to get the output C_l^{TT} in different files (e.g. `root = output/ISW_cl.dat`).