CLASS Exercise Corrections

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

October 30, 2014

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

Solution 1a:

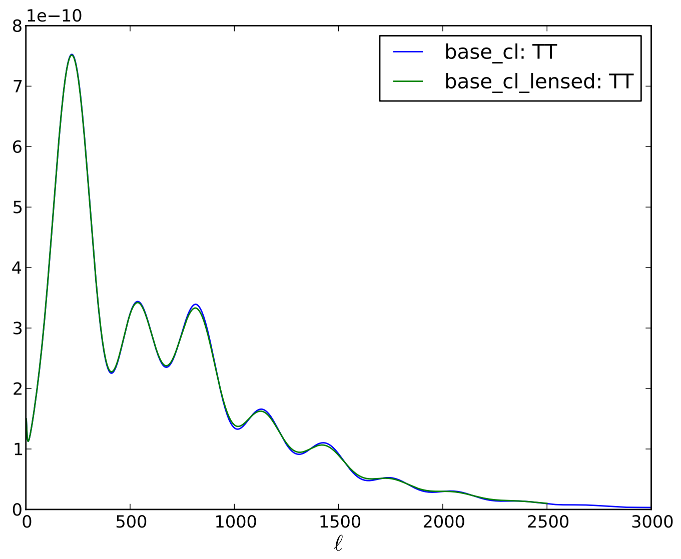
The solution of these exercises is of course not unique. You could e.g. run with the input file:

```
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.py output/base_cl.dat output/base_cl_lensed.dat --scale lin
```

producing the figure



or in matlab:

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

Solution 1b:

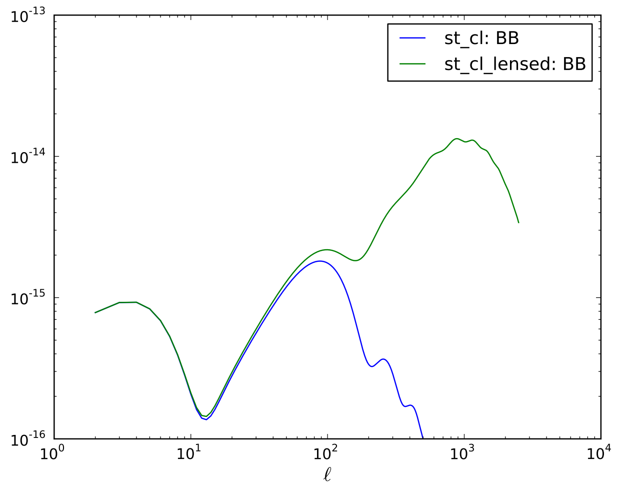
You could e.g. run with the input file:

```
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.py output/st_cl.dat output/st_cl_lensed.dat -y BB --scale loglog
```

producing the figure



or in matlab:

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

Solution 1c:

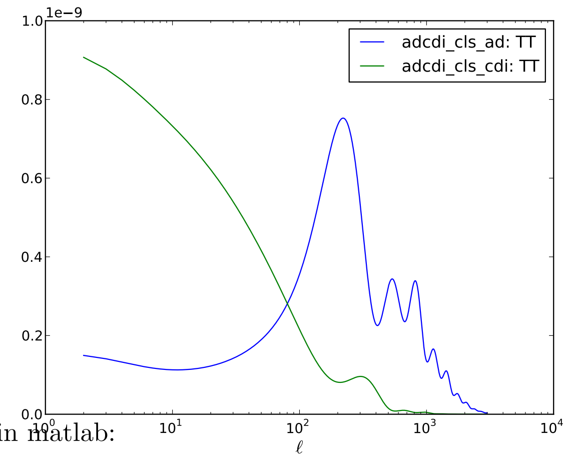
You could e.g. run with the two input files:

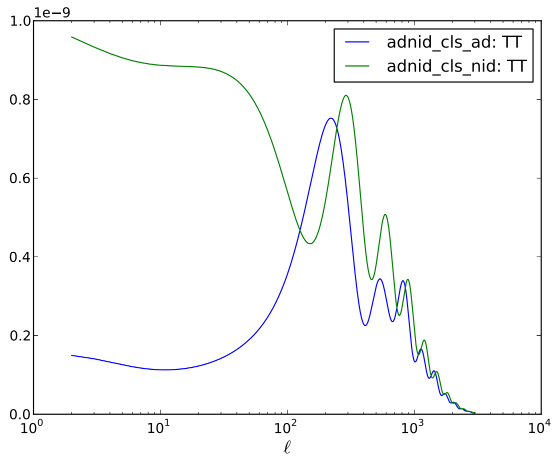
```
ic = ad cdi
f_cdi = 2.
n_cdi = 1.
output = tCl,pCl,lCl
lensing = yes
root = output/adcdi_
#
background_verbose = 1
thermodynamics_verbose = 1
perturbations_verbose = 1
transfer_verbose = 1
primordial_verbose = 1
spectra_verbose = 1
nonlinear_verbose = 1
lensing_verbose = 1
output_verbose = 1
```

```
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.py output/adcdi_cls_ad.dat output/adcdi_cls_cdi.dat --scale loglin
python CPU.py output/adnid_cls_ad.dat output/adnid_cls_nid.dat --scale loglin
```





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 1d:

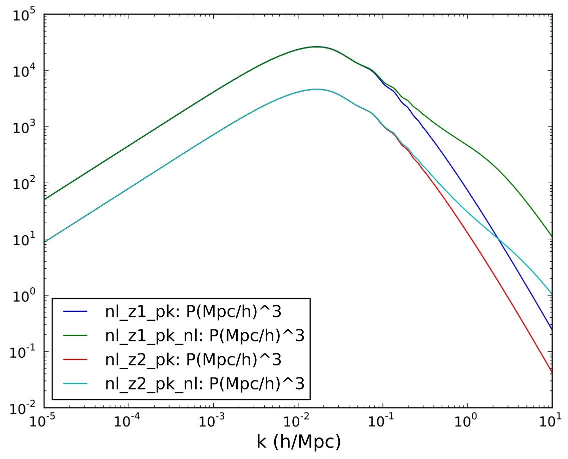
You could e.g. run with the input file:

```
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.py output/nl_z1_pk.dat output/nl_z1_pk_nl.dat output/nl_z2_pk.dat
output/nl_z2_pk_nl.dat --xlim 1.e-5 10
```

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')
```

Solution of exercise 1e

The part of `source/input.c` dealing with baryon density can be modified as such:

```
/* Omega_0_b (baryons) */
class_call(parser_read_double(pfc,"Omega_b",&param1,&flag1,errmsg), errmsg , errmsg);
class_call(parser_read_double(pfc,"omega_b",&param2,&flag2,errmsg), errmsg , errmsg);
/* Read the baryon fraction eta_b! */
class_call(parser_read_double(pfc,"eta_b",&param3,&flag3,errmsg), errmsg , errmsg);

/* Test that at most one flag has been set: */
class_test(class_at_least_two_of_three(flag1,flag2,flag3), errmsg,
           "In input file , you can only pass one of Omega_b, omega_b or eta_b");

if (flag1 == _TRUE_) pba->Omega0_b = param1;
if (flag2 == _TRUE_) pba->Omega0_b = param2/pba->h/pba->h;

/* Set Omega_b in background structure. Formula hardcoded :( */
if (flag3 == _TRUE_) pba->Omega0_b = 1.81e6*param3*pow(pba->T_cmb,3)/pba->h/pba->h;
```

Solution of exercise 1f

First step: Modifications of `input.h`

```
enum target_names {theta_s, Omega_dcdmdr, omega_dcdmdr, Omega_scf, Omega_ini_dcdm, omega_ini_dcdm,
                                  tn_sigma8};
enum computation_stage {cs_background, cs_thermodynamics, cs_perturbations,
                        cs_primordial, cs_nonlinear, cs_transfer, cs_spectra};
#define _NUM_TARGETS_ 7 //Keep this number as number of target_names
```

Second step: Modifications of `input.c`, in function `input_try_unknown_parameters()`

```
/** These two arrays must contain the strings of names to be searched
      for and the coresponding new parameter */
  char * const target_namestrings[] = {"100*theta_s","Omega_dcdmdr","omega_dcdmdr","Omega_scf",
                                       "Omega_ini_dcdm","omega_ini_dcdm","sigma8"};
  char * const unknown_namestrings[] = {"h","Omega_ini_dcdm","Omega_ini_dcdm","scf_shooting_parameter",
                                        "Omega_dcdmdr","omega_dcdmdr","A_s"};
  enum computation_stage target_cs[] = {cs_thermodynamics, cs_background, cs_background,cs_background,
                                        cs_background, cs_background, cs_spectra};
```

and then, just after

```
  if (flag == _TRUE_)
    input_verbose = param;
  else
    input_verbose = 0;
```

you can add

```
  /** Optimise flags for sigma8 calculation.*/
  pt.k_max_for_pk=1.0;
  pt.has_pk_matter=_TRUE_;
  pt.has_perturbations = _TRUE_;
  pt.has_cl_cmb_temperature = _FALSE_;
  pt.has_cls = _FALSE_;
  pt.has_cl_cmb_polarization = _FALSE_;
  pt.has_cl_cmb_lensing_potential = _FALSE_;
  pt.has_cl_number_count = _FALSE_;
  pt.has_cl_lensing_potential=_FALSE_;
  pt.has_density_transfers=_FALSE_;
  pt.has_velocity_transfers=_FALSE_;
```

and later, still in the same function, there is a pat starting by `switch (pfzw->target_name[i])`. It contains different cases. You should add one more case (the cases can be written in any order):

```
    case tn_sigma8:
      output[i] = sp.sigma8-pfzw->target_value[i];
      break;
```

Third step: Modifications of `input.c`, in function `input_get_guess()`: there is a part starting by `switch (pfzw->target_name[index_guess])`. It contains different cases. You should add one more case (the cases can be written in any order):

```
    case tn_sigma8:
      /* Assume linear relationship between A_s and sigma8 and fix coefficient
         according to vanilla LambdaCDM. Should be good enough... */
      xguess[index_guess] = 2.43e-9/0.87659*pfzw->target_value[index_guess];
      dxdy[index_guess] = 2.43e-9/0.87659;
      break;
```

Finally, test your changes with this input file

```
output =tC,mPk
sigma8 = 0.9
P_k_max_1/Mpc = 1.
input_verbose = 1
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
```

Solution of exercise 2a.

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_
```

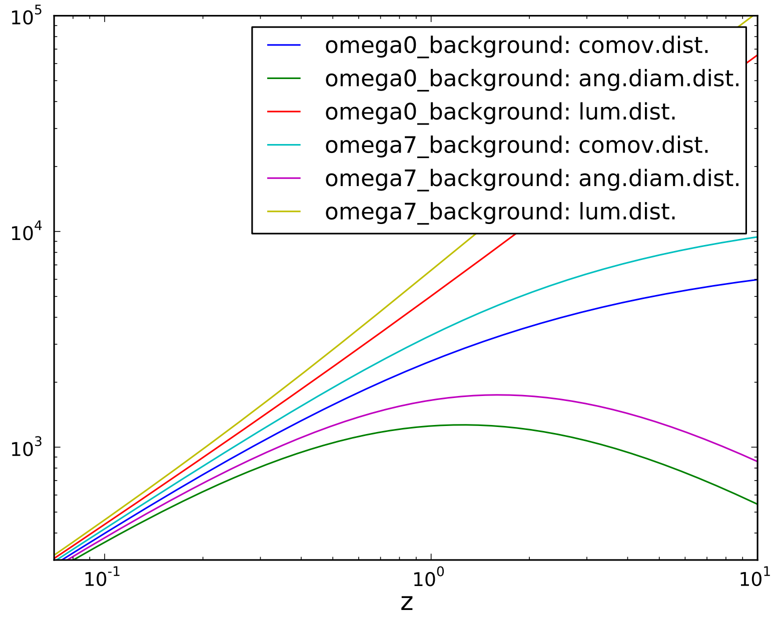
After running

```
> ./class omega0.ini
> ./class omega7.ini
```

we can reproduce the Dodelson plot (up to a rescaling of the vertical axis by 0.7/3000, as explained in the text) using our preferred plotting software. Using `CPU.py` you may type:

```
> python CPU.py output/omega0_background.dat output/omega7_background.dat -y dist
--scale loglog --xlim 0.07 10 --ylim 300 100000
```

(note that the three output we want have diet in their bname, so we did not need to list the exact three names). You get



Using the MATLAB function `plot_CLASS_output()` it is enough to type

```
plot_CLASS_output({'output/omega0_background.dat','output/omega7_background.dat'},...
                  {'dist'},...
                  'xvariable','z',...
                  'xlim',[0.07,10],...
                  'EpsFilename','Dodelson')
```

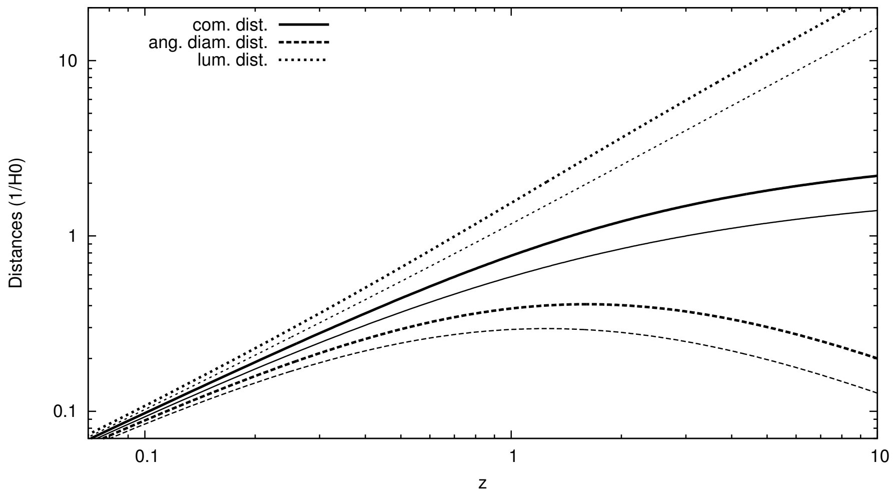
If we want to reproduce Dodelson's plot exactly with the same units, line styles, etc., we can customise the plot using, for instance, gnuplot. Then, we first go to the `output/` directory and open gnuplot by typing
`> gnuplot`
we can now enter:

```
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 2b.

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("w0_efld",pba->w0_efld);
Omega_tot += pba->Omega0_efld;
...
pba->Omega0_efld = 0.;
pba->w0_efld = -1.;
...
```

In `include/background.h`:

```
...
double Omega0_efld;
double w0_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->w0_efld));
  rho_tot += pvecback[pba->index_bg_rho_efld];
  p_tot += (pba->w0_efld) * pvecback[pba->index_bg_rho_efld];
}
...
if (pba->has_efld == _TRUE_) {
   class_test(pba->w0_efld>=1./3.,
              pba->error_message,
              "Your choice for w0_efld=%g is suspicious, there would not be radiation domination at
```

```
                  pba->w0_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);
    ...
    class_store_columntitle(titles,"(.)rho_efld",pba->has_efld);
    ...
    class_store_double(dataptr,pvecback[pba->index_bg_rho_efld],pba->has_efld,storeidx);
    ...
```

We then run with two input files:

```
Omega_efld = 0.1
w0_efld = 0.1
write background = yes
root = output/efluid_
```

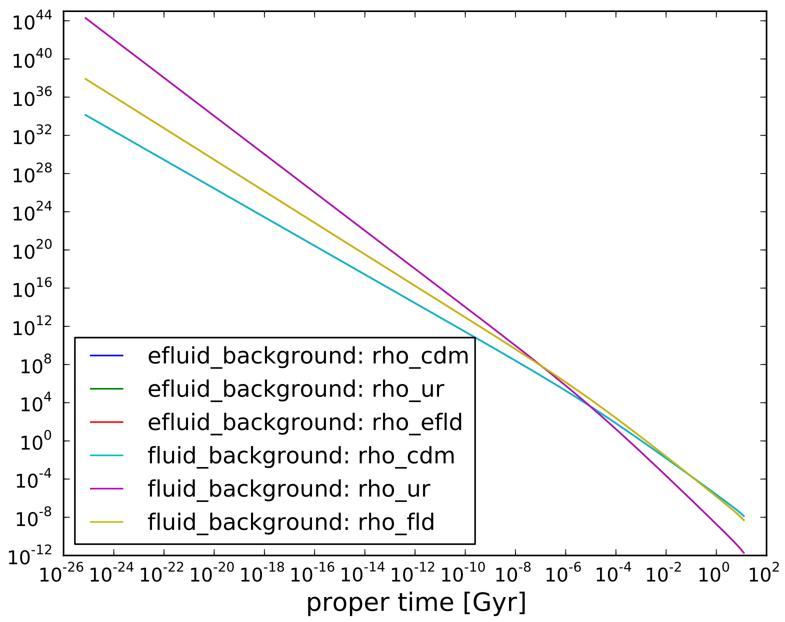
and

```
#output = tCl      # uncomment to check the remark in the exercise text: it will produce an error
Omega_fld = 0.1
w0_fld = 0.1
write background = yes
root = output/fluid_
```

To check that they give the same (and hence that our changes were correct) we can superimpose the background density evolutions with:

```
> python CPU.py output/efluid_background.dat output/fluid_background.dat --scale loglog
-x proper -y cdm ur fld
```

It gives:



Hence the two models are identical, as expected.
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.