Tutorial 2: Sterile neutrino and cosmology – assignments

S. Gariazzo

March 25, 2021

In this lecture we will compute cosmological observables and study the effect of sterile neutrinos on the CMB spectrum. The tools we will use are `CLASS` and `MontePython`.

First of all, we should learn how to use `CLASS` and how to configure the physics we want to consider. The code can be executed in two ways: running it via command line with an input (`*.ini`) file, or using the python wrapper `classy`.

1 CLASS

The parameters that configure the content of the physics are the same in both cases, and the easiest way to learn about them is to open the `explanatory.ini`, provided with the code. The user is encouraged to avoid editing the original `explanatory.ini`, in order to preserve the information on the default parameters as provided by the developers.

The file is divided in several sections, according to the category of parameters that are available in the code. The section names are:

- General parameters
- Species parameters
- Exotic energy injection parameters
- Non-linear parameters
- Primordial parameters
- Spectra parameters
- Lensing parameters
- Distortions parameters
- Output parameters

You can easily guess where you will be able to find information on what you are interested on. The content of the universe, for example, is described in "Species parameters": there, you will find information on the available species, which include standard (photons, baryons, ultra-relativistic species, cold and non-cold dark matter, curvature, dark energy) and non-standard species (decaying dark matter, dark matter interacting with dark radiation, dark matter interacting with baryons to be added). Other useful parameters are described in "General parameters" (output quantities, modes, Hubble parameter, BBN, reionization), "Primordial parameters" (for inflation), "Non-linear parameters" (how to compute the non-linear matter power spectrum), "Lensing parameters" (lensing potential and lensing of the CMB spectrum) and "Output parameters" (name of the output files, verbosity of the code).

For defining the species that constitute the universe, `CLASS` allows to specify them in different ways. For example, the photon density can be defined by `T_cmb`, `Omega_g` or `omega_g`, the neutrino mass (massive neutrinos are considered non-cold dark matter species) is indicated by `m_ncdm`, `Omega_ncdm` or `omega_ncdm`, and so on.

A full description of all the parameters is well beyond the scope of this course. In the following section we will see some examples of viable `CLASS` ini files, you may look in the `explanatory.ini` for more details.

1.1 Command line use

The easiest way to use the code is to write a configuration file, let us call it `mymodel.ini`. The file must include at least a few lines to configure `CLASS`: all the mandatory parameters that are not specified in the file are fixed by the code to the Planck best-fit or to some reasonable value. In order to consider a cosmological model with three massless neutrinos, this is an example of our `mymodel.ini` file:

```
N_ur=3.044
N_ncdm=0
root=output/massless3_

T_cmb=2.7255
h=0.67
Omega_b=0.04
Omega_cdm=0.3
reio_parametrization = reio_camb
z_reio=9
P_k_ini type = analytic_Pk
A_s=2.2e-9
n_s=0.96
lensing=yes
non_linear = halofit
write warnings=yes
output = tCl,pCl,lCl,mPk
temperature contributions=tsw,eisw,lisw,dop,pol
modes=s
```

Notice that the first three lines are those that define the number of ultra-relativistic neutrinos, of massive neutrinos and the path where to save the data. The rest of the lines define the content of the universe. Notice that there is nothing special in the choice of parameters, you may change the configuration and use the Planck best-fit values if you prefer. Notice also that we are asking the code to compute temperature, polarization and lensing power spectra for the CMB and the matter power spectrum (`output = tCl,pCl,lCl,mPk`), the lensed CMB spectrum (`lensing=yes`) and also the non-linear matter power spectrum (`non_linear = halofit`).

Once the file is saved, we can run the code using `./class mymodel.ini`, assuming that we are inside the `CLASS` folder and that the file is saved in the same place. Once the execution is completed, you will find several new files in the output folder:

- `massless3_00_cl.dat` (non-lensed CMB spectra);
- `massless3_00_cl_lensed.dat` (lensed CMB spectra);
- `massless3_00_pk.dat` (matter power spectrum from linear theory);
- `massless3_00_pk_nl.dat` (non-linear matter power spectrum).

Let us now repeat the execution of `CLASS` with a different model. We want now to consider one massless and two massive neutrinos, to simulate the case when the lightest neutrino is massless and the ordering of the neutrino masses is normal. In such case, from the squared-mass splittings obtained by atmospheric and solar neutrino oscillations we obtain that the masses of the three standard neutrinos are 0, ~ 10 and 50 meV, respectively. We also have to reduce the number of ultra-relativistic species in order to reflect the presence of only one massless neutrino. In `CLASS`, these values are defined using:

```
N_ur=1.0176
N_ncdm=2
m_ncdm=0.01,0.05
root=output/massive3_NO_
```

while the rest of the input file presented above remains unchanged. Notice that the comma separates the values of the two masses, expressed in eV.

Before moving to sterile neutrinos, we should now practice a bit with plotting the output. Here you will see two figures obtained from the output files of the two runs discussed above: figure 1 shows the comparison of the lensed temperature spectrum of the CMB comparing the two cases we computed (stored in the `*_cl_lensed.dat` files), while figure 2 reports the same but for the linear (`*_pk.dat`) and non-linear (`*_pk_nl.dat`) power spectra. Can you reproduce the figures? The file content is described in the headers, so you should open them in order to understand how to read and plot the information.

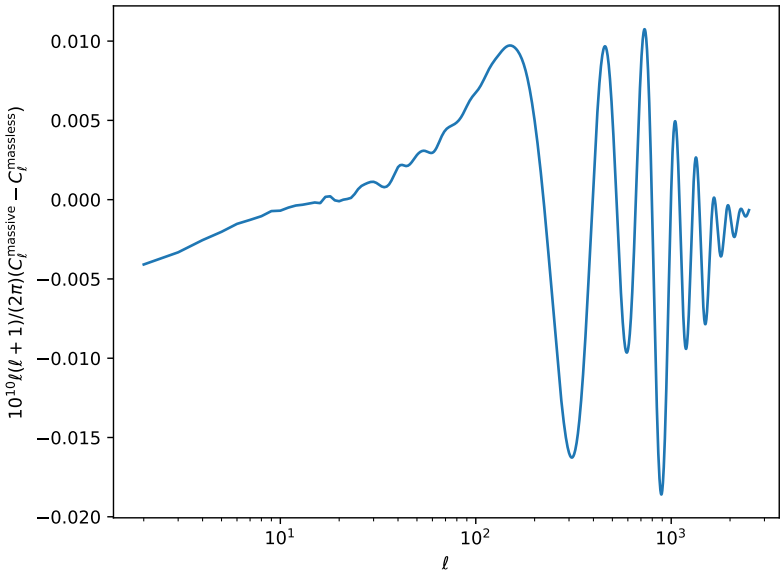

Figure 1: Difference between the lensed CMB temperature of the massless and massive neutrino case described in the text.

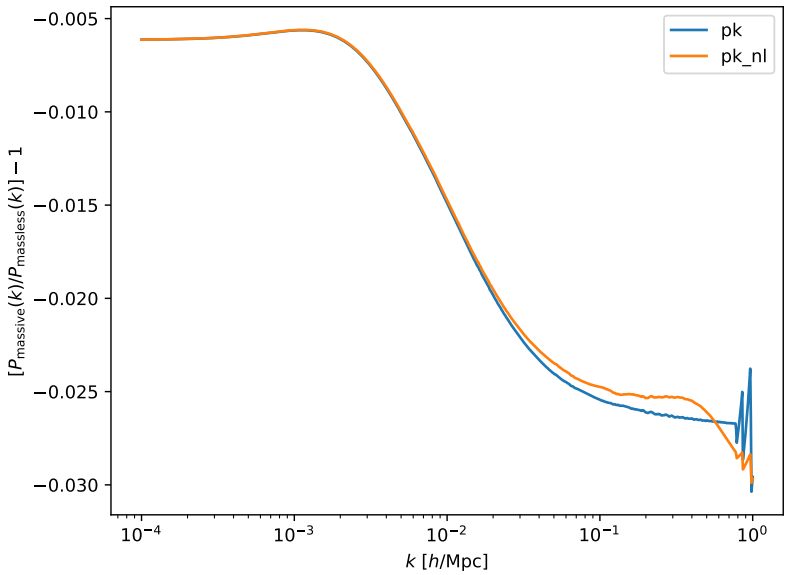

Figure 2: Difference between the matter power spectra (linear and non-linear) of the massless and massive neutrino case described in the text.

1.2 Python wrapper

We should now verify if the `CLASS` python wrapper works correctly, because it will be necessary for the `MontePython` exercise later. If you have properly installed it, when you run `python -c 'from classy import Class'` from your command line you should obtain no errors. Remember to install and use it with the same python version, or within the same environment if you use Anaconda or similar. If it works, we can write a python script which imports the `CLASS` wrapper, configures it and runs the numerical calculations:

```
params={
    "T_cmb":2.7255,
    "h":0.67,
    "Omega_b":0.04,
    "Omega_cdm":0.3,
    "reio_parametrization": "reio_camb",
    "z_reio":9,
    "P_k_ini type": "analytic_Pk",
    "A_s": 2.2e-9,
    "n_s": 0.96,
    "lensing": "yes",
    "non linear": "halofit",
    "output": "tCl,pCl,lCl,mPk",
    "modes":"s",
    "P_k_max_1/Mpc":3.0
}
from classy import Class
#define Class object
mymodel = Class()
# pass input parameters
mymodel.set(params)
mymodel.compute()
```

Notice that the `params` *dictionary* contains almost the same parameters we defined in `mymodel.ini`: the python wrapper and the ini files are read in the same way by `CLASS`. If you execute these lines, you will have to wait for few seconds in order to allow `CLASS` to compute most of the expensive computations.

At this point, we need to understand how to extract the output, which is not stored in files but it must be obtained calling the methods of `mymodel` (which is an instance of the `Class` class[1]). In particular, the quantities stored in the output files we discussed above when talking about the command-line output, are obtained using the following methods:

- `mymodel.raw_cl(max_ell)`
- `mymodel.lensed_cl(max_ell)`
- `mymodel.pk(k, z)` (non-linear!)
- `mymodel.pk_lin(k, z)` (linear)

Notice that the first two functions return a dictionary, which contains several items depending on the requested outputs (they may include "ell", "tt", "ee", "pp" and more: check which ones are available with `mymodel.raw_cl(max_ell).keys()`), while the last two functions return a number, corresponding to the linear or non-linear power spectrum at the given wavemode k and redshift z. You should also pay attention to the normalization of the CMB spectra: by default, `CLASS` saves $\ell(\ell+1)/(2\pi)\,\mathcal{C}_\ell$, as described in the output files, while in the code you obtain the $\mathcal{C}_\ell$ without any prefactors.

In order to plot the lensed $\mathcal{C}_\ell$s and the non-linear power spectrum for the `mymodel` case defined above, this is an example showing how to proceed (add these lines to those written previously):

```
import matplotlib.pyplot as plt
import numpy as np
cls = mymodel.lensed_cl(2500) #cls up to ell=2500
plt.figure()
plt.semilogx(cls["ell"][2:], 1e10/2/np.pi*cls["ell"][2:]*(cls["ell"][2:]+1)*cls["tt"][2:])
plt.show()
# plt.savefig("...") to save it to file
plt.figure()
```

[1]Notice that "class" is a reserved word in python: it cannot be used for variables, modules or objects. For this reason, the developers called the `CLASS` wrapper "`classy`" and the included class "`Class`" (with capital C).

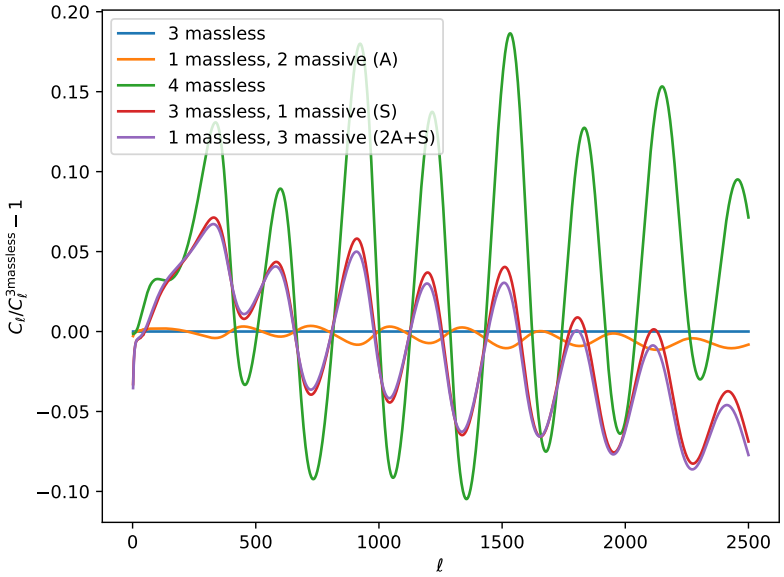

Figure 3: Comparison of the TT spectrum for different neutrino models.

```
kk=np.geomspace(1e-4,1,1000) #list of 1000 points uniformly spaced in the logarithm
pks = [mymodel.pk(k, 0) for k in kk] #compute pk(k, z=0) for each k in the kk list
plt.loglog(kk, pks)
plt.show()
```

Let us move now to one step further. Using the python wrapper, and starting from the examples provided in the previous section, where we compared the case of massless and massive neutrinos, plus the lines of code provided in this section, you should now be able to add one sterile neutrino. You can either consider a massless (in such case you will probably want to add one degree of freedom to ultra-relativistic particles) or a massive one (implemented through a non-cold dark matter species, ncdm). Assume a mass equal to 1 eV for the additional neutrino. Consider the following models: three massless neutrinos, one massless and two massive neutrinos (active), four massless neutrinos, three massless and one massive neutrino (sterile), one massless and three massive neutrinos (2 active, one sterile). Can you compare the CMB spectra obtained when considering these models? The result should be similar to figure 3.

Before moving to the next section, let me add another parameter that relates to the sterile neutrino. Until now, we have considered only neutrinos which provide a fixed contribution to the total amount of relativistic energy density in the very early universe, i.e. each neutrino always contributes with $\Delta N_{\rm eff} \simeq 1$. Sometimes, and in particular this applies to the sterile neutrino case, it is useful to allow the neutrino to contribute less than 1. In `CLASS`, the parameter `deg_ncdm` (it behaves exactly as `m_ncdm` when `N_ncdm` > 1) controls the degeneracy of the neutrino family under consideration. This parameter is equal to one by default, but you can decrease (or increase) the contribution of each family by varying this parameter. In the code, it acts as a multiplicative factor in front of the momentum distribution function of the given family, therefore altering its contribution to $N_{\rm eff}$. While we normally do not want it to vary in the case of active neutrinos, it may be useful for the sterile case. The parameter `deg_ncdm` renormalizes the neutrino momentum distribution function with a constant, which is equal to $\Delta N_{\rm eff}$ if one considers a Dodelson-Widrow model for the thermalization of the new state. Figure 4 shows the effect of adding one sterile neutrino with different degeneracy parameters (`deg_ncdm` equal to 0.1, 0.5, 1).

2 `MontePython` (optional)

In this last section, we will apply what we learned until now to perform a Markov Chain Monte Carlo (MCMC) scan with `MontePython`, to study cosmological constraints on sterile neutrino properties. Although the method under consideration can be easily adopted to perform a real analysis, in this example we will use fake cosmic microwave background data to save computation time and installation problems.

`MontePython` is a software, developed in python, which uses `CLASS` to compute the cosmological observables. After downloading the code, firstly we have to indicate in `default.conf` the path to our `CLASS` folder.

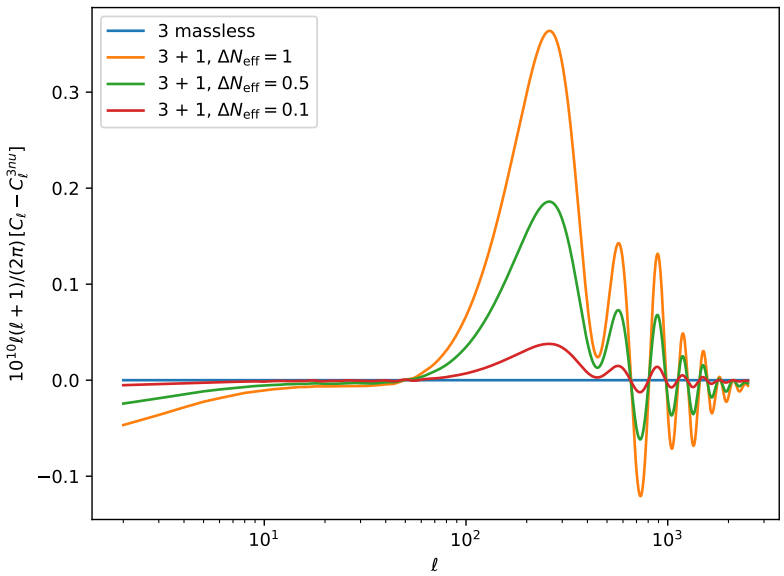

Figure 4: Comparison of the TT spectrum for different neutrino models.

You can copy and edit `default.conf.template` to have an example on how to write the configuration. After this is done, we can check if it works using a simple example:

```
python montepython/MontePython.py run -o test -p input/example.param
```

This will create a `test` folder and run a very short MCMC with the configuration stored in `input/example.param`. If you open this file, you will see that it defines a run considering a fake Planck likelihood (the experiments are defined in the list `data.experiments=['fake_planck_bluebook']`), performing a scan over the six parameters that describe the standard cosmological model, called ΛCDM (see the lines starting with `data.parameters`), and computing only 10 MCMC steps.

If it worked, we can configure a run that also considers the sterile neutrino parameters. `MontePython` accepts as physical parameters anything that constitutes a valid input for `CLASS`, plus a few more, mostly managed by the `update_cosmo_arguments` function in `montepython/data.py`. This means that the names of the parameters we have to vary are those discussed in the previous sections: we have to fix `N_ur`, `N_ncdm` > 1, `m_ncdm` and `deg_ncdm` appropriately. One possible example, considering `deg_ncdm` $= 1$ by default, is:

```
data.parameters['m_ncdm']     = [1.,    0,   10, 0.5, 1,   'cosmo']
data.cosmo_arguments['N_ur'] = 3.044
data.cosmo_arguments['N_ncdm'] = 1
```

Notice the syntax for defining fixed parameters that must be passed to `CLASS` (`data.cosmo_arguments[...]=...`). The syntax for varying parameters, instead, is: [mean, min, max, 1-sigma, scale, role]. In details:

- "mean" is the initial value for the parameter.
- "min", "max" define the allowed range (they can be None if the range is irrelevant).
- "1-sigma" defines the expected final 1σ range, but is actually used only for defining the initial MCMC steps.
- "scale" can be used to renormalize the parameter.
- "role" can be "cosmo" (`CLASS` parameters), "nuisance" (internal parameters of some likelihood), "derived" (parameters computed by `CLASS` from the available "cosmo" parameters).

In case we want to vary more than one neutrino mass at the time, we have to pay attention to the fact that `MontePython` considers each mass as an independent parameter, so that they cannot go in the same list in `MontePython` and they need to be stored with a different syntax. Let us consider a case with two massive neutrinos (`data.cosmo_arguments['N_ncdm'] = 2`), for example: in `CLASS` we would write `m_ncdm=0.05,1`. In `MontePython`, we have to use:

```
data.parameters['m_ncdm__1']     = [1.,    0,   0.1, 0.01, 1,   'cosmo']
data.parameters['m_ncdm__2']     = [1.,    0.1,  10,  0.5, 1,   'cosmo']
```

In this way, `MontePython` will concatenate the varying parameters in a single `m_ncdm` with the appropriate values before passing it to `CLASS`.

To summarise, in order to perform a MCMC analysis including the six parameters of the ΛCDM model plus three massless neutrinos and one massive sterile neutrino, we have to add few lines to those existing in `input/example.param`. It is better to copy the file into a new one (`nus.param`, for example) and add the following lines to the new param file:

```
data.parameters['m_ncdm']     = [1.,    0,   10, 0.5, 1,   'cosmo']
data.parameters['deg_ncdm']   = [1.,    0,    3, 0.2, 1,   'cosmo']
data.cosmo_arguments['N_ur'] = 3.044
data.cosmo_arguments['N_ncdm'] = 1
```

It is good to keep `data.N=10` in the param file, as it will save us from very long runs if we submit by mistake: this value is easily overwritten at runtime. Let us now perform a very short run to check that the setup works:

```
python montepython/MontePython.py run -o nus -p nus.param
```

Some output will appear in your terminal, possibly ending with a list of obtained points and a resume line such as `# 10 steps done, acceptance rate:  0.4`, if everything worked.

We can now perform a longer run, which we will use to extract the parameter constraints. Notice that if we use the same output folder, we do not need to specify again the param file, as `MontePython` will use the one stored in the folder itself during the previous run. A possibility to speed-up the computation is to give to `MontePython` an estimate for the covariance matrix of the parameters, with the option `-c`. Several of them are available in the `covmat/` folder, you may want to use the `covmat/fake_planck_lcdm.covmat`, which at least implements correctly the correlations for the six parameters of the ΛCDM model. Even with the covariance matrix, it is convenient to reduce the initial jump factor (option `-f`) from the default value 2.4 to something like 1.2. This will increase the initial acceptance of points while reducing a bit the speed of exploring the parameter space, but in any case the jump is adjusted while the MCMC proceeds. Finally, we have to specify the number of points that we want to compute, instead of the `N=10` defined in the param file, but this is easily done adding the option `-N`, let's say with 10000 points:

```
python montepython/MontePython.py run -o nus -N 10000 -c covmat/fake_planck_lcdm.covmat -f 1.2
```

This command will take several hours to complete. Notice also that 10000 points are probably too few for a decent analysis, and that running with the real likelihoods generally takes a bit longer. Here, however, we just want to understand how to use `MontePython` and not to reproduce Planck constraints or obtain competitive results, for which longer computation times are required.

Once it finished (or during the run, if you are impatient), you can check what `MontePython` stores in the `nus` folder. You will notice a `log.param` file, which is basically a copy of your input param file, a log file, and a list of `*.txt` and `*.paramnames` files, which store the MCMC points and the names of the parameters. In order to analyze the files and produce some default plots, we can use the `MontePython info` command:

```
python montepython/MontePython.py info nus
```

If the length of the chain is not sufficient (easily possible with `N=10000`), you will see an error message, otherwise some plots and various other files will be stored in `nus/` and `nus/plots/`.

Assume you ran a sufficiently long chain, what you obtain is something like the set of files you will find at `http://personalpages.to.infn.it/~gariazzo/courses/2103_GGI_nus/nus.tar.gz`. Download and extract the compressed file: you will see the chain files, some useful output files generated with the `info` command (`*.bestfit`, `*.h_info`, `*.v_info`, `*.tex`, `*.covmat`) and a `plots/` folder. Can you extract the upper limits on $\Delta N_{\rm eff}$ and m_s? What do they tell us about the presence of a sterile neutrino in the universe?