Hand-in Exercise 1

Deadline: Oct 09, 23:59

Read these instructions carefully before you start coding.

This first hand-in exercise covers material from the **first three lectures** (up to and including integration). Unless noted otherwise, you are expected to code up your own routines using the algorithms discussed in class and **cannot use special library functions** (except exp()). Extremely simple functions like `arange`, `linspace` or `hist` (without using advanced features) are OK. When in doubt, write your own, or ask us. You can use the routines you wrote for the tutorials, however, your routines must be written **by yourself**. Codes will be checked for blatant copying (with other students as well as other sources), routines which are too similar get **zero points**.

For every main question you should write a **separate program**. We **must** be able to run everything with **a single call to a script** `run.sh`[1], which downloads any data (data needed for an exercise **may not** be included in your code package but needs to be retrieved at runtime), runs your scripts and generates a PDF containing all your source code and outputs **in the following format**:

- Per main question, the code of any shared modules.
- Per sub-question, an explanation of what you did.
- Per sub-question, the code specific to it.
- Per sub-question, the output(s) along with discussion/captions.

Your code may be handed however you'd like, for example by emailing a zip file or by sharing a github repository. If you organize your code in multiple folders, `run.sh` should be **in the top folder**. Have a fellow student test that your code works by running ./run.sh to make sure there are no permission errors! Exercises that are not run with this single command or do not have their code and output in the PDF get **zero points** (this includes solutions in Jupyter notebooks).

Ensure that your code runs to completion on the `pczaal` computers using `python3`(!). Codes that do not get **zero points**. Your code should have a total run time of **at most 10 minutes**, solutions generated will not be checked beyond this limit. If, **during testing**, a part of your code takes long to run, remember that you can run it once and read in its output (saved to file) in the rest of your code – however, the rules as stated above still apply to whatever you hand in at the end (everything run with a single command, all outputs produced on the fly, total runtime limit, etc). It is possible to test your own code on the pczaal remotely by using ssh, you can directly log into a pczaalXX computer (XX is between 00 and 21) using `ssh -XY [username]@pczaalXX.strw.leidenuniv.nl` to test your code.

For all routines you write, **explain** how they work in the comments of your code and **argue** your choices! Similarly, whenever your code outputs something **clearly indicate** next to the output/in the PDF what is being printed. This includes discussing your plots in their captions.

If a part of your code **does not run**, explain what you did so far and what the problem was and still include that part of the code in the PDF for possible partial credit. If you are unable to get a routine to work but you need it for a follow-up question, use a library routine in the follow-up (and clearly indicate that and why you do so).

Each sub-question now starts with a reference to a relevant tutorial (T) as well as a reference to the relevant lecture (L). For example, *[L2,T2.3]* means the question is related to lecture 2 and question 3 in tutorial 2. Your previously written code for these will be a great starting point!

[1]See the example here: `https://github.com/Fonotec/NURSolutionTemplate`.

1. **Cooling rates in cosmological simulations**
When running large simulations of (parts of) the Universe it is not possible to model the interstellar medium (ISM) of galaxies in high detail, because the resolution of a typical run is around $10^6\,\mathrm{M}_\odot$. To solve this problem cosmological simulations use something called cooling tables. These are tables with the cooling rate as a function of density and temperature, for different redshifts. These different cooling rates are calculated using specialized software that models the ISM at these specific redshifts. In the case of the EAGLE simulation, Cloudy is used to generate the interpolation tables. The tables are made publicly available in [1].

In this exercise we will build an interpolator to calculate the cooling rate as a function of redshift, densities and temperatures. In general what [1] use is the fact that the cooling rate is mainly traced by 11 elements (H, He, C, N, O, Ne, Mg, Si, S, Ca and Fe). This means that during the run of the simulation the abundance of these 11 elements is stored in the particles, which allows quite accurate estimations of the cooling rate in galaxies. Because the abundance of the different metals is stored, the cooling rate for the different elements can be calculated and from this the total cooling rate can be obtained as follows:

$$\Lambda = \Lambda_{\mathrm{H,He}} + \sum_{i>He} \Lambda_{i,\odot} \left(\frac{n_\mathrm{e}}{n_{\mathrm{e},\odot}}\right)\left(\frac{n_i}{n_{i,\odot}}\right) \tag{1}$$

$$= \Lambda_{\mathrm{H,He}} + \sum_{i>\mathrm{He}} \Lambda_{i,\odot} \frac{n_\mathrm{e}/n_\mathrm{H}}{(n_\mathrm{e}/n_\mathrm{H})_\odot} 10^{[i/\mathrm{H}]} \tag{2}$$

where $10^{[i/\mathrm{H}]} \equiv (n_i/n_\mathrm{H})/(n_i/n_\mathrm{H})_\odot$ (using that $n_\mathrm{H} = n_{\mathrm{H},\odot}$). This means that in order to calculate the total cooling rate we need to calculate $(n_\mathrm{e}/n_\mathrm{H})_\odot$ by interpolating again over density, temperature and redshift. For the sake of simplicity we will assume in this exercise that the mass fraction of Helium to Hydrogen is given by 0.258. This means that we only need to interpolate for $(n_\mathrm{e}/n_\mathrm{H})$.

You can download the cooling data using `wget https://www.strw.leidenuniv.nl/WSS08/coolingtables_highres.tar.gz` and untar the file in a folder called `CoolingTables`. The goal of this exercise is to make a 3D linear interpolator for this dataset in temperature, density and redshift.

To inspect the files you downloaded you can use `hdfview`, `HDFCompass` and `h5py`[2].

(a) (8 points) *[L1, T1.2, T1.5]* Use your 3D linear interpolator to make a single plot of the cooling rate as a function of temperature for densities of ($1\ \mathrm{cm}^{-3}$, $10^{-2}\ \mathrm{cm}^{-3}$, $10^{-4}\ \mathrm{cm}^{-3}$, $10^{-6}\ \mathrm{cm}^{-3}$) at a redshift of $z = 3$ at a metallicity of 25% solar. Solar metallicity is given in Tab. 1.

Table 1: Solar abundances to adopt (taken from [1])

Element	n_i/n_H	Mass Fraction
H	1	0.7065
He	0.1	0.2806
C	2.46×10^{-4}	2.07×10^{-3}
N	8.51×10^{-5}	8.36×10^{-4}
O	4.90×10^{-4}	5.49×10^{-3}
Ne	1.00×10^{-4}	1.41×10^{-3}
Mg	3.47×10^{-5}	5.91×10^{-4}
Si	3.47×10^{-5}	6.83×10^{-4}
S	1.86×10^{-5}	4.09×10^{-4}
Ca	2.29×10^{-6}	6.44×10^{-5}
Fe	2.82×10^{-5}	1.10×10^{-3}

(b) (2 points) *[L1, T1.2, T1.5]* Use your 3D linear interpolator to make a plot of the cooling rate as a function of temperature for the allowed redshift ($0-\sim 9$) range and a density of

[2]you can read files using h5py using `f = h5py.File("mytestfile.hdf5", "r")`, accessing individual elements can be done using `a = f["keyname"]`. More information about h5py can be found at `https://docs.h5py.org/en/stable/quick.html`

$n = 0.0001$ cm^{-3}. For your plot use 50% solar metallicity. Make a mp4 movie[3] that shows the evolution of the cooling function as a function of temperature for the redshift range between 0 and ∼9, use atleast 100 different redshifts in this movie and a framerate of 10.

2. **Redshift distribution of galaxies**
   It is possible to derive the redshift distribution of a large sample of galaxies by cross-correlating their positions on the sky with spectroscopic sources, of which the redshift is known. The galaxies are statistically most likely to be at the redshift of the sources they cluster most strongly with. One can show that we can write the the cross-correlation signal w of these galaxies (sample "g") with spectroscopic sources (sample "s") in redshift bin z_i as a linear combination of the cross-correlations of the spectroscopic sources with those at different redshifts:

$$w_{\rm gs}(z_i) = \sum_j f_j w_{\rm ss}(z_i, z_j). \tag{3}$$

   Here f_j is the fraction of the galaxies that are in redshift bin z_j. Our goal is to use the measured clustering signals to find the unknown f_j.

   You can find an idealized data set with a unique solution for all f_j at `https://home.strw.leidenuniv.nl/~daalen/Handin_files/`. There are 16 redshift bins, equally spaced between 0 and 0.8 so each has a width $\Delta z = 0.05$.

   (a) (8 points) *[L2,T2.3]* Note that we can write equation (3) in matrix form, with the vector of unknowns being $\mathbf{f}$. Write an LU decomposition code and solve for $\mathbf{f}$. Output both matrices after the LU decomposition, output $\mathbf{f}$, and the sum of the fractions $\sum_j f_j$.[4] **Important:** All variables must be single-precision, e.g. of type `numpy.float32`.
   Hint: You should find that $\sum_j f_j \approx 1$!

   (b) (2 points) *[L2,T2.3]* Do a single iterative improvement on $\mathbf{f}$. Output the improved $\mathbf{f}$ and the sum of the improved fractions $\sum_j f_j$.

3. **Satellite galaxies around a massive central**
   The spherical distribution of satellite galaxies around a central is often fitted with an NFW profile, under the assumption that it traces the dark matter halo mass. However, the following number density profile turns out to be a better match, from near the centre to well outside the virial radius:

$$n(x) = A \langle N_{\rm sat}\rangle \left(\frac{x}{b}\right)^{a-3} \exp\left[-\left(\frac{x}{b}\right)^{c}\right]. \tag{4}$$

   Here x is the radius relative to the virial radius, $x \equiv r/r_{\rm vir}$, and a, b and c are free parameters controlling the small-scale slope, transition scale and steepness of the exponential drop-off, respectively. A normalizes the profile such that the 3D spherical(!) integral from $x = 0$ to $x_{\rm max} = 5$ gives the average total number of satellites:

$$\iiint_V n(x)\,{\rm d}V = \langle N_{\rm sat}\rangle\,. \tag{5}$$

   (a) (6 points) *[L3]* Take $a = 2.2$, $b = 0.5$ and $c = 3.1$. Write a numerical integrator to solve equation (5) for A given those three parameters, taking $\langle N_{\rm sat}\rangle = 100$. Output A. *Hint: First think about what* dV *is for a spherical integral!*

   (b) (3 points) *[L1]* Using these same parameter values, make a log-log plot of the single points for $n(10^{-4})$, $n(10^{-2})$, $n(10^{-1})$, $n(1)$ and $n(5)$ with an axis range from $x = 10^{-4}$ to $x_{\rm max} = 5$. Interpolate the values in between based on just these points and put on the log-log plot the interpolation – make sure to argue in the comments of your code why you chose to interpolate in a certain way.

[3]See the hand-in exercise template at `https://github.com/Fonotec/NURSolutionTemplate` for an example on how to make a mp4 movie

[4]See the solution template on how to include outputs of your scripts in your reports (`https://github.com/Fonotec/NURSolutionTemplate`)

(c) (3 points *[+1.5]*) *[L1]* In the case of counting satellite galaxies around central galaxies we need to use Poisson statistics in order to understand if individual central galaxies have more satellite galaxies than normal. We would like a function that returns the Poisson probability distribution for integer k:

$$P_\lambda(k) = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{6}$$

given a positive mean λ. Note that the probability distribution is normalized, i.e. $\sum_{k=0}^{\infty} P_\lambda(k) = 1$ for any non-zero λ. Write a function that returns $P_\lambda(k)$. When your code is run, output $P_\lambda(k)$ to at least six significant digits for these values: $(\lambda, k) = (1, 0)$, $(5, 10)$, $(3, 21)$ and $(2.6, 40)$. Bonus points if it can also correctly give $(101, 200)$ (without hard-coding of course).
Important: For this exercise, we are limiting memory. That means not using standard Python types, because these can use an almost unlimited number of bytes. Instead, you need to use `numpy` types of at most 64 bits, like `numpy.int64` or `numpy.float64`.
Hint: If useful, you can turn this into a continuous function by taking a floating point k and rounding it. Regardless, you should find $P_\lambda(k) > 0$ for any non-zero λ and finite k – if you find zero or a negative number, think about how you could change the calculation. You don't need to use a gamma-function, but if you want to, you'll have to write it yourself!

References

[1] Robert P. C. Wiersma, Joop Schaye, and Britton D. Smith. The effect of photoionization on the cooling rates of enriched, astrophysical plasmas. *Monthly Notice of the Royal Astronomical Society*, 393:99–107, February 2009.