Hand-in exercise 1

Deadline: April 9, 15:30

This first hand-in exercises covers material from the **first seven lectures** (up to and including fitting and modelling data). 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()). You can use the routines you wrote for the tutorials, however, your routines must be written **by yourself**. Codes will be checked for plagiarism (with other students as well as other sources), routines which are too similar get **zero points**.

To hand in your solutions, put your code up on an otherwise empty **GitHub repository** (like the one you made during tutorial 1) and share it with us (`daalen`, `fonotec` and `syeda-lammim-ahad` – no commits after the deadline!). Your code can be written as a single program or as separate pieces, however, we **must** be able to run everything with **a single call to a script** `run.sh`[1], which should **generate a PDF** containing all your source code followed by all outputs of the code. 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 (e.g. 2(h)), 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, total runtime limit, etc).

For all routines you write, **explain** how they work in the comments of your code! Similarly, whenever your code outputs something **clearly indicate** next to the output/in the PDF what is being printed. 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).

1. Preparing some useful routines.
   (a) We are going to need a function that returns the Poisson probability distribution for integer k:

$$P_\lambda(k) = \frac{\lambda^k e^{-\lambda}}{k!}, \tag{1}$$

   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 $(2.6, 40)$. **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!

   (b) Write a random number generator that returns a random floating-point number between 0 and 1. At minimum, use some combination of an (M)LCG and a 64-bit XOR-shift. We'll test its quality: like in the tutorial, have your code plot sequential random numbers against each other in a scatter plot (x_{i+1} vs x_i) for the first 1000 numbers generated. Additionally, have your code generate 1,000,000 random numbers and plot the result of binning these in 20 bins 0.05 wide. Change your RNG parameters if necessary. Make sure you use a fixed seed and set it only once for your entire program. The seed value should be the first output when we run your code.

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

2. Satellite galaxies are often fitted with an NFW profile, under the assumption that they trace 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:

$$n(x) = A \langle N_{\rm sat} \rangle \left(\frac{x}{b}\right)^{a-3} \exp\left[-\left(\frac{x}{b}\right)^{c}\right]. \tag{2}$$

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:

$$\iiint_V n(r)\,{\rm d}V = \langle N_{\rm sat} \rangle . \tag{3}$$

(a) Have your code randomly generate three numbers $1.1 < a < 2.5$, $0.5 < b < 2$ and $1.5 < c < 4$. Write a numerical integrator to solve equation (3) for A given those three parameters, taking $\langle N_{\rm sat} \rangle = 100$. Output the numbers a, b, c and A.

(b) Make a log-log plot of and plot 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 – make sure to argue in the comments of your code why you chose to interpolate in a certain way.

(c) Numerically calculate ${\rm d}n(x)/{\rm d}x$ at $x = b$. Output the value found alongside the analytical result, both to at least 12 significant digits. Choose your algorithm such as to get them as close as possible.

(d) Now we want to generate 3D satellite positions such that they statistically follow the satellite profile in equation (2); that is, the probability distribution of the (relative) radii $x \in [0, x_{\rm max})$ should be $p(x)\,{\rm d}x = n(x) 4\pi x^2\,{\rm d}x / \langle N_{\rm sat} \rangle$ (with the same a, b and c as before). Use one of the methods discussed in class to sample this distribution. Additionally, for each galaxy generate random angles ϕ and θ such that the resulting positions are (statistically) uniformly distributed as a function of direction. Output the positions (r, ϕ, θ) for 100 such satellites.

(e) Repeat (d) for 1000 haloes each containing 100 satellites. Make another log-log plot showing $N(x) = n(x) 4\pi x^2$ over the same range as before, but now over-plot a histogram showing the average number of satellites in each bin. Use 20 logarithmically-spaced bins between $x = 10^{-4}$ and $x_{\rm max}$ and don't forget to divide each bin by its width. Do your generated galaxies match this distribution?

(f) Write a root-finding algorithm to find the solution(s) to $N(x) = y/2$ in the same x-range, where y is the maximum of $N(x)$. Use the same parameter values as before. Output the root(s).

(g) Take the radial bin from (d) containing the largest number of galaxies. Using sorting, calculate the median, 16th and 84th percentile for this bin and output these values. Next, make a histogram of the number of galaxies in this radial bin in each halo (so 1000 values), each bin of this histogram should have a width of 1. Plot this histogram, and over-plot the Poisson distribution (using your code from 1(a)) with λ equal to the mean number of galaxies in this radial bin.
Hint: If the histogram and the Poisson distribution don't seem to match within a reasonable uncertainty, something's wrong!

(h) The normalization factor A depends on all three parameters. Calculate A at 0.1 wide intervals in the ranges of a, b and c given above. You should get a table containing 6240 values. Choose an interpolation scheme and write a 3D interpolator for A as a function of the three parameters based on these calculated values.

3. Now we are going to turn things around. Download these files:

   `https://home.strw.leidenuniv.nl/~daalen/files/satgals_m11.txt`
   `https://home.strw.leidenuniv.nl/~daalen/files/satgals_m12.txt`
   `https://home.strw.leidenuniv.nl/~daalen/files/satgals_m13.txt`
   `https://home.strw.leidenuniv.nl/~daalen/files/satgals_m14.txt`
   `https://home.strw.leidenuniv.nl/~daalen/files/satgals_m15.txt`

   Each file contains haloes in a certain mass bin with variable numbers of satellites. Write down the log-likelihood corresponding to a set of random realizations of the satellite profile in equation (2) with some unknown $\langle N_{\rm sat}\rangle$. Retain only the terms with a (residual) dependence on a, b and/or c (including $A = A(a, b, c)$).

   (a) Find the a, b and c that maximize this likelihood. Do this separately for each different file/mass bin. Output these values.

   (b) Write an interpolator for a, b and c as a function of halo mass, **or** fit a function to each. Argue your specific choice! Plot the results.

Most recent changes: specified repository should be fresh, changed 20 to 21 in exercise 1(a), added memory limit to 1(a), underscored text in 2(b), and added links to exercise 3.