Hand-in Exercise 4

Deadline: Dec 22, 23:59

Read these instructions carefully <u>before</u> you start coding.

This fourth hand-in exercise covers material from **lectures 11 and 12**. 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 or previous assignments, 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 starts with a reference to a relevant tutorial (T) and/or 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. **Simulating the solar system**
The most simple systems to simulate are systems that do not have a lot of degrees of freedom or particles, one of the best examples of such a limited system is our Solar system. The Solar system has one sun and eight planets. If we also include the moon in our simulation this means that our simulation only has 10 particles. In these kinds of simulations the most brute force method of summing all the forces directly is computational feasible.

We first need to set up initial conditions for the Solar system extremely precisely, with current day positions. This can be done by using `Time` from `astropy` to set the current time and `get_body_barycentric_posvel` to get the current positions and velocities of the planets of interest. As an example if we want to know the current position of the Mars, this could be done as follows:

```
# we need to import the time module from astropy
from astropy.time import Time
# import some coordinate things from astropy
from astropy.coordinates import solar_system_ephemeris
from astropy.coordinates import get_body_barycentric_posvel

# set the current time
t = Time("2020-12-07 10:00")

with solar_system_ephemeris.set('jpl'):
    mars = get_body_barycentric_posvel('mars', t)

print(mars)
```

This method returns a coordinate object, using this object it is possible to select separately the positions and the velocities. Instead of using units like km (the standard type of units used by `astropy`), it is more convenient to use Astronomical Unit (AU). Using `units` from `astropy` you can easily convert the units to proper astronomical units. This works as follows:

```
from astropy import units as u
marsposition = mars[0]
marsvelocity = mars[1]

# calculate the x position in AU
print(marsposition.x.to_value(u.AU))

# calculate the v_x velocity in AU/day
print(marsvelocity.x.to_value(u.AU/u.d))
```

(a) (2 points) *[L11]* Generate initial conditions for the solar system (i.e. 'earth', 'sun', 'moon', 'mercury', 'venus', 'mars', 'jupiter', 'saturn', 'uranus', 'neptune') using the units shown in the example above, also use masses from astropy when possible. Make a plot showing the (x, y) positions and another showing the (x, z) positions for all the objects at the "current time" defined in the example above.

(b) (5 points) *[L11, T10.1]* We are going to write an algorithm that only calculates the forces between the sun and the other planets, so basically a steady potential for the Sun. The appropriate differential equations of the system to solve in this case have the form $\mathbf{a}_i = \mathbf{F}(\mathbf{x}_i)$. Here i is the current time step and $\mathbf{F}(\mathbf{x})$ specifies the used force law, which only depends on the distance. Use the **leapfrog** algorithm to calculate the forces between the sun and the other planets. For that, use the equations for $\mathbf{x}_{i+1}$ and $\mathbf{v}_{i+1/2}$ provided in Lecture 11 for the **leapfrog** algorithm, and do not forget to kick (i.e. apply the acceleration) your initial conditions for the velocity. Compute the positions of the objects over a time of 200 years, using a time step of 0.5 days. Make plots with the positions of all the objects versus time (make three separated plots, one for the x position, another for y position, and another for z position).

(c) (5 points) *[L11]* In the previous part we calculated the evolution of the Solar system using the approximation that only the Sun acts with a force on the other planets, this ignores planet-planet interactions. In this exercise, you will write a code that takes into account all the particle-particle (PP) interactions in your initial conditions. When we want to take into account also the planet-planet interactions we need to design a complete particle-particle (PP) algorithm.

In this case we do want to calculate all the forces, with the specific acceleration given by:

$$\mathbf{f}_{21} = -\frac{G}{|\mathbf{r}_2 - \mathbf{r}_1|^3}(\mathbf{r}_2 - \mathbf{r}_1). \tag{1}$$

This means that it is possible to construct a rank-3 tensor which consists of a matrix with elements that are vectors, in general this acceleration matrix is given by (in 3 dimensions):

$$\begin{pmatrix} 0 & \mathbf{f}_{12} & \mathbf{f}_{13} \\ \mathbf{f}_{21} & 0 & \mathbf{f}_{23} \\ \mathbf{f}_{31} & \mathbf{f}_{32} & 0 \end{pmatrix}. \tag{2}$$

and using the fact that this is an antisymmetric matrix, we can write the acceleration in a tensor as:

$$A = \begin{pmatrix} 0 & \mathbf{f}_{12}m_2 & \mathbf{f}_{13}m_3 \\ -\mathbf{f}_{12}m_1 & 0 & \mathbf{f}_{23}m_3 \\ -\mathbf{f}_{13}m_1 & -\mathbf{f}_{23}m_2 & 0 \end{pmatrix} \tag{3}$$

Using this information, write a code that takes into account all the particle-particle interactions in your initial conditions. Make the same plots as for (b). *Hint: Note that if by using a double for loop, this matrix can be filled with $N(N-1)/2$ calculation steps, after this we can sum along the second axis and obtain the total force on the corresponding particles. To be clear one thing we absolutely do not want to do is calculate the force N^2 times, this means that we increase the number of calculations by more than a factor 2.*

(d) (6 points) *[L11, T10.2]* Repeat the calculations but this time using a self-written Runge-Kutta 4 or Bulirsch-Stoer algorithm. Make the same three plots as before, including the positions obtained from both the **leapfrog** algorithm and the second algorithm you have chosen.

2. **Spiral and elliptical galaxies**

In our Local Universe there are two main classes of galaxies, presenting different physical properties: elliptical and spiral. Spiral galaxies have younger stars, their stars orbit on the same plane, they are still forming several new stars every year and are found in less dense environments. Elliptical galaxies, instead, have older stars, the orbit of their stars are randomly oriented, they barely form new stars and are in general in denser environments. Classifying galaxies using photometric information has been a hot topic for the last decade. In this exercise we will use logistic regression to train and classify galaxies.

Download the dataset from `https://home.strw.leidenuniv.nl/~garcia/NUR/galaxy_data.txt`. This file contains 5 columns: the first column corresponds to $\kappa_{\rm co}$, a parameter indicating how much a galaxy is dominated by ordered rotation; the second column contains an estimate of their color, where higher values corresponding to redder colors (i.e., older stellar populations); in the third column you can find a measure of how extended each galaxy is; the fourth column contains the flux of an emission line used to measure the star formation rate of galaxies, while the last columns is a morphology flag: 1 for spirals and 0 for ellipticals.

(a) (1 point) *[L12, T11.1]* Preparing the dataset: take as your n features the quantities in the first four columns. Concatenate the features in a matrix with dimensions $m \times n$. Then apply feature scaling to have features with approximately mean 0 and standard deviation 1. Print all features for the first 5 objects.

(b) (4 points) *[L12, T11.1]* Choose two sets of two columns that you want to use in the next part of the exercise. For both pair of columns, implement logistic regression and use any minimization routine you've previously written. Check on convergence any way you like, and plot the value of the cost function for the different iterations until converged.Compare both pairs of columns and comment on the differences in the cost function.

(c) (3 points) *[L12, T11.1]* Compute the number of true/false positives/negatives (on the training set, we don't have a test set here to keep things simple), then compute the F_1 score and output all of these for both pairs of columns. Finally, for each pair, make a plot of the two columns against each other and overplot the decision boundary. Comment on your results.