Programming and linear algebra (6BBR06)

CBL Assignment

November 18, 2024

1 Introduction

In this assignment, we will simulate the spreading of a virus among people attending a party, and study the effect of social distancing. This may indeed remind you of a certain era from a few years ago. While the case described resembles people, strong analogies apply to a simulation of the diffusion/reaction of colloidal particles.
This assignment consists of a number of steps to guide you through setting up the entire simulation. The questions gradually build up in complexity, but also become more conceptual and less detailed. The last task is completely open-ended.
This approach gives you a lot of freedom in your preferred implementation (e.g. functionality, structure/algorithm, parameters). The freedom to set your own objectives and ask your own questions may be daunting for some, if you are used to work in a directed manner, but is an essential part of building an academic mindset. This also means that after a certain point of this assignment, there is no specific targeted path towards what you need to do. Let the driver be your own curiosity.

2 Assessment

You should deliver a Jupyter notebook file that contains both the code as well as some argumentation/analysis. Each sub-question should be accompanied with a brief motivation/explanation and analysis, reflecting your main findings, results, implementation and discussion. The notebook should function when executing the cells from top-to-bottom. It should be handed in including the generated output (e.g. figures, printed numbers, etc).
The assessment is done following the rubrics posted on Canvas. In short, the following will be assessed:

- Code style (functionality): Do you use functions, sensible input/output arguments, is the code doing what it should, and does it work generically (is it robust)?
- Code style (commenting): Do you use logical function and variable naming, did you add comments on particular aspects of the code (where needed, don't overdo this), do you add function descriptions?
- Visualization: Are the graphs readable, is the type of graph sensible? Check for axis labels, legend if multiple lines.
- Analysis: Observations (what do you see, what happens?) and explanations (why does it happen?)
- Extension: You will create a framework in questions 1-8, after which you can use the framework to extend and solve your own research questions (question 9).

Tasks

1. The `random.gauss` function returns a single random number chosen from a normal (Gaussian) distribution. Study the documentation of this function and create a command that returns a single random number from a distribution with the mean $\mu = 0.7$ and standard deviation $\sigma = 0.25$. Print 10 numbers generated in this way using a loop construction.

2. We are going to need much more than one single number, and large arrays of random numbers can be generated efficiently using the `numpy.random` library. The function `numpy.random.normal(mu,sigma,size)`[1] is the equivalent for the Gauss distribution used above, and can be used to generate arrays of size `size`. Let's now see how well this function performs based on different input:
   - Generate an array consisting of random numbers of size `(n,1)`. Do this for multiple array sizes `n`, with `n` chosen at particular (smartly chosen) points between 1 and $1 \cdot 10^6$ numbers.[2]
   - Compute the mean and standard deviation of each of these arrays and store them separately.
   - Plot the absolute difference of the imposed mean `mu` and standard deviation `sigma` with the computed ones vs the array size, on a double-log scale.
   - Create a histogram of the largest array of random numbers.

We are going to simulate a person taking a random walk in $\mathbb{R}^2$, i.e. taking steps with a random direction and size on a 2D plane. The step size of a human is normally distributed with an average of 0.7 m and standard deviation 0.25 m, which we know how to generate. The direction is, however, taken from a *uniform* distribution (assuming the person is somewhat drunk). The `numpy.random.uniform` command returns random numbers from such a distribution. If you use this to generate a random number $\theta \in [0, 2\pi]$, we can compute the step direction s using:

$$s_x = \sin(\theta)\,, s_y = \cos(\theta)$$

The position of the person is stored in an array `pos`, where the first column is the x-position and the second column the y-position. The rows contain the different steps, e.g. time if you like that analogy.

3. Create a random walk of a person that starts at $(0, 0)$, and takes 1000 steps:
   - Create a function `takeStep(mu, sigma)`, which returns an (x, y) array containing the step in a *uniformly* distributed random direction with a *normally* distributed magnitude, based on the average `mu` and standard deviation `sigma`.

[1] `numpy` versions newer than 1.25 will classify this approach as 'deprecated', but it is arguably the best way of providing compatibility between different installations.

[2] You do not need to select all sizes between 1 and 1M, but e.g. try to use a logarithmic increasing size so that it covers multiple orders of magnitude.

- Create a for-loop to step through time, call `takeStep` every time step, and update the position.
- Plot the path that was taken by the person on the (x, y)-plane.

4. Our single person is alone and sad. Therefore, we introduce more people to the party.
   - Initialise $N_p = 100$ persons at random locations, on a domain bound by $x, y \in [0, 10]$.[3]
   - Update the `takeStep` function to create steps for all people at once
   - Plot the positions and paths of the people at different moments in time, or create an animation[4].

5. Add a function `getCrowdStatistics(pos)` that returns some crowd statistics, the mean, minimum and maximum distance among people:
   - Compute the inter-person distances[5]
   - Compute the mean, min and max of the inter-person distances, and return these values
   - Call the `getCrowdStatistics(pos)` each time step from the for-loop and store the results in separate array(s).
   - After the loop is completed, plot the statistics vs the time steps.

6. You'll notice that some persons wander off outside our original domain bounds. It's not fun if people are leaving the party, and it could be dangerous outside too, so we're going to restrict them to the party domain.
   - Assess the proposed steps and detect when a person would be leaving the domain, which we will deny.
   - For each step that will be denied, take a new step suggestion, test it again, etc. Define your own criteria on when to stop and what will happen with the person, but be sure to motivate your choice.
   - Try to animate the point positions to verify the behavior[6], and again use the function `getCrowdStatistics(pos)` to assess the confined crowd.
   - As a hint (not a requirement, just a suggestion), you could pass `pos` as well to the `takeStep` function and call this function recursively for those positions that were not accepted.

[3] You may still want to use `pos` to store the positions of the people, but it is recommended to only store the latest time step in this array, such that each rows now holds a position for an individual persons, instead of the motion through time.

[4] Creating animations in a jupyter notebook is not trivial, so feel free to search for possible solutions on this, or just simply plot at a few moment in time.

[5] You can do this following your own intuition, or use specialized functionalities found in e.g. `Scipy` or `numpy`.

[6] If the animation does not work for you, you might want to create a plot at a few moments in time

7. One person in the party is infected and spreads the disease.
   - Create an `isInfected` array of size $1 \times N_p$ where `False` indicates healthy and `True` indicates infected.
   - The chance of infection c depends on the inter-person distance d of an encounter, as given by the function:
     $$c(d) = \exp\left[-3.5d\right]$$
     For each encounter (or each encounter where the chance is non-negligible), use the random number generator to determine whether infection actually occurs and transfer the infection.
   - Simulate a group of 100 persons in a domain of $(x, y) \in [0, 25]$ for 300 time steps, and track and plot the infection count over time. Infected persons become contagious immediately.

8. Now add a social distance functionality to the simulation; similar to restricting the persons to the domain boundaries, now also include a function that tries to keep 1.5 m distance, or (when that is not possible) at least increases the inter-person distance with respect to the previous time step. Investigate the effect of the social distance parameter. You may need to make your own choices here. Motivate the implementation choices you make in the report.

9. When everything works as it should, you can add your own rules (e.g. healing or immunity, deaths, slower or faster moving infected (yikes!), infected that do not care about social distancing, etc). Fidn an interesting research objective yourself that challenges you to make a nice program. Carefully describe the objectives and boundary conditions, and then try to implement it. Continue adding rules/mechanics until you are satisfied with the outcome. Feel free to alleviate any rules imposed earlier in this assignment if deemed necessary. You can use any library that you find useful, but be aware that we may not have everything installed ourselves; make sure that the outcome + interpretation of this experiment is clear on its own.

References