Computing Methods for Physics 1

Lecturer: Francesco Pannarale

Hands-on lab session 3, A.Y. 2022-23

The goal of this session is to practice `numpy` and `matplotlib`. To do this, random walks will be simulated and analyzed. **Avoid lists and for loops as much as possible!**

- **1D random walk** — The most elementary example of a random walk is the one on the integer number line, $\mathbb{Z}$, that starts in 0 and at each step moves $+1$ or -1 with equal probability. Use `numpy`, and `ndarray`'s in particular, to simulated a random walk of this kind with 10^4 steps.

  Instructions:

  – Use `help(numpy.random)` to select the best way to perform 10^4 fair draws times all at once. Do not use loops! This can be done in one line of Python, storing all draws in an `ndarray`.

  – Operate un such array with `cumsum` to establish the position of your random walker throughout its (random) steps. This method returns the cumulative sum of the elements along a given axis of an `ndarray`. This means that given $\{a_{ij\ldots k}\}$, with $i \in \{1, \ldots, N_i\}, j \in \{1, \ldots, N_j\}, \ldots, k \in \{1, \ldots, N_k\}$, you can use it to determine with a single call

  $$\sum_{j=1}^{N_j} a_{ij\ldots k}\,,$$

  for example. Use `help(numpy.cumsum)` to find out how.

  – Use `max()` and `min()` to find how far from $x = 0$ the walker went. Use `argmax()` and `argmin()` to find out the first time it reached these remote posistions. Finally, use the `numpy.where` method to find out all the steps at which the walker was in $x = 20$.

  – Plot the random walk and make an animation[1] of it.

- **Statistics on 1D random walks** — By simulating several random walks, a statistical analysis may be performed.

  Instructions:

  – Fill up a 2D `ndarray` with 10^3 random walks of 10^3 steps each. You will probably have to revisit how you use `cumsum`.

  – Determine which walker went the furthest away from $x = 0$. No loops, 1 line of code!

  – Determine how far from $x = 0$ each walker went. Use `numpy.amax(...)` to do this in 1 line of code.

  – Plot a histogram of these maximum distances reached Overlay the histogram of final distances and the histogram of all distances the walkers were ever at. [You will have to `numpy.flatten` your 2D array of random walks for the last histogram.]

[1] On Colab animations are slow: if you are using Colab generate frames in strides of 100. E.g., `ani = animation.FuncAnimation(fig, refresh, np.arange(1, Nsteps-1, 100), interval=50, blit=True, repeat=False)`.

  - Plot and animate a specific walk.
  - Make a static plot that overlays all walks and an animated version of it.
  - Experiment with adding gaussian noise at each step of a random walk.

- **3D random walks [advanced]** — Perform N 3D random walks and revisit the requests above. At the end of this, you will have what is essentially a rudimentary simulation of the 3D diffusion of a highly concentrated initial distribution of fluid.

  Notes:

  - There are 3 random draws (with options $\pm$ for x, y, and z).
  - If you focus on statistics/histograms, set $N = 10^3$ so you do not have low statistics.
  - If you focus on walk plots/animations, start with $N = 10$ to lower your waiting times and avoid cluttering in your figures. You will need the methods `scatter3D` and `plot3D`.
  - Try adding 0 as a third option when performing the random draws and observe how your results change.
  - Try constructing the walk by taking unitary steps in a direction that is uniformly sampled (over a sphere of radius 1) each time. Observe how your results change.