MATH327: Statistical Physics, Spring 2023

Computer Project

Overview and instructions

In this computer project you will numerically analyze two types of diffusive behaviour in one-dimensional random walks. After warm-up exercises on pseudo-random numbers and inverse transform sampling, analyzing ordinary diffusion will allow you to verify your numerical results by comparing them with exact analytic predictions based on the law of large numbers and central limit theorem. You will then adapt these verified numerical methods to consider *anomalous diffusion*, where exact analytic predictions are not available.

There are five exercises below, four of which provide relevant background information in addition to the tasks for you to complete. While the exercises mention some syntax specific to Python, you may use a different programming language if you prefer. This demo illustrates all the Python programming tools needed for the project. For any reasonable programming language, the numerical computations for each exercise should complete in a few minutes or less.

This project is **due Tuesday, 21 February**. Submit your work by file upload on Canvas.[1] **Both** your answers to the questions below and the code that produces your results must be submitted. These can be uploaded in multiple files or in a tar/zip archive, as you prefer. With the exception of Mathematica `.nb` files, it will be quicker for me to check code submitted in its native format (for example, a `.py` file for Python code or a `.m` file for MATLAB code). Anonymous marking is turned on, and I will aim to return feedback by 3 March.

Exercise 1: Pseudo-random numbers

Background

We have discussed how statistical physics is based on considering systems that involve some element of randomness. Because computer programs are deterministic, it is not possible to use them to generate a truly random sequence of numbers.[2] Instead, computer algorithms generate *pseudo*-random numbers, which are entirely sufficient for our purposes.

A sequence of pseudo-random numbers *appears* random in the sense that knowing the first $N-1$ elements in the sequence does not suffice to predict

[1] By submitting solutions to this assessment you affirm that you have read and understood the Academic Integrity Policy detailed in Appendix L of the Code of Practice on Assessment and have successfully passed the Academic Integrity Tutorial and Quiz. The marks achieved on this assessment remain provisional until they are ratified by the Board of Examiners in June 2023.

[2] New quantum technologies are being developed as a way to produce truly random numbers. This is part of the motivation for large investments in quantum technologies around the world.

the Nth element with a high probability of correctness. Equivalently, it takes a very large number of elements for the sequence to start repeating itself. Such repetition *will* eventually happen, because computers encode numbers in a finite set of bits, which can represent only a finite set of numbers. For example, 32 bits can represent all integers from 0 through $2^{32} - 1 \sim 10^9$, while 64 bits increase the upper bound to $2^{64} - 1 \sim 10^{19}$. Python uses the Mersenne Twister algorithm as its default pseudo-random number generator (PRNG). This algorithm can provide $2^{19937} - 1 \sim 10^{6001}$ numbers before its sequence repeats.

We can view the absence of true randomness as an advantage rather than a limitation. Deterministic pseudo-random numbers allow our computer programs to be reproducible up to the (very high) precision of the computer. Each exercise below starts by initializing the PRNG with a "seed". Given the same seed, the PRNG will subsequently generate the same sequence of pseudo-random numbers. In Python, as shown in the demo, this initialization is done by calling the function `random.seed(s)`, where s is the seed we specify.

Task

The Python function `random.random()` generates a pseudo-random number u with the uniform probability distribution

$$p(u) = \begin{cases} 1 & \text{for } 0 \le u < 1 \\ 0 & \text{otherwise} \end{cases} . \tag{1}$$

Clearly $\int p(u)\,du = \int_0^1 du = 1$, as required. What are the exact mean μ and standard deviation σ of this probability distribution?

`[2 marks]`

Initialize the PRNG with seed $s = 271828$. For each of the five $R = 10$, 100, 1000 $10{,}000$ and $100{,}000$, generate a sequence of R pseudo-random numbers u_r distributed according to $p(u)$. Don't re-initialize the PRNG when changing R, or else these sequences will partially duplicate each other. Use each sequence to estimate the mean and standard deviation via the law of large numbers,

$$\overline{u}_R = \frac{1}{R}\sum_{r=1}^{R} u_r \qquad\qquad \overline{\sigma}_R \equiv \sqrt{\left(\frac{1}{R}\sum_{r=1}^{R} u_r^2\right) - \overline{u}_R^2}. \tag{2}$$

How do your numerical results compare to your exact analytic predictions above? Four significant figures should suffice for these comparisons.

`[5 marks]`

In class (and on page 15 of the lecture notes) we saw $\left\langle (\overline{u}_R - \mu)^2 \right\rangle \propto 1/R$. Let's test this numerically by repeating the above computation of $\overline{u}_R$ another 99 times, ignoring $\overline{\sigma}_R$ for simplicity. Together with the result you reported above, this gives a total of 100 estimates of the random variable $(\overline{u}_R - \mu)^2$, which we can use to approximate the expectation value as

$$\overline{(\overline{u}_R - \mu)^2} \equiv \frac{1}{100}\sum_{i=1}^{100} (\overline{u}_R - \mu)_i^2 . \tag{3}$$

Rather than reporting your results as numerical values, plot $R \times \overline{(\overline{u}_R - \mu)^2}$ vs. R and see whether the five points appear approximately constant. If so, is the size of this constant roughly what you would expect?

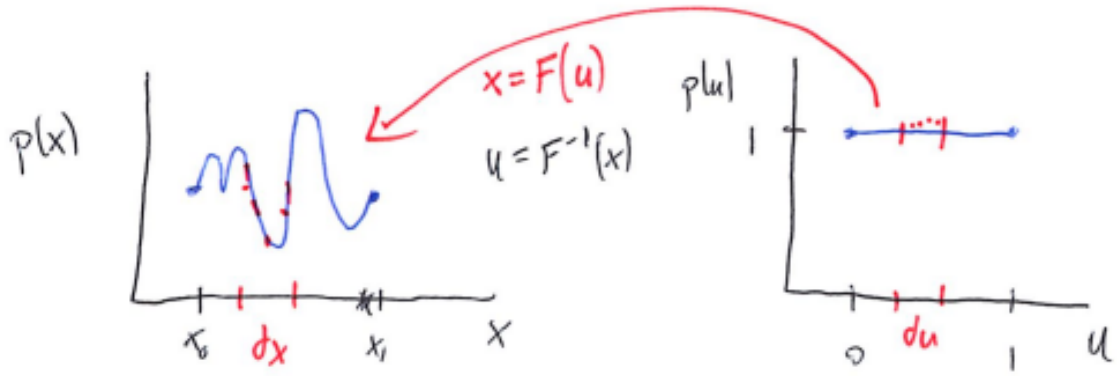
Hints: Include 0 on the y-axis of your plot to maintain a sense of scale. The Matplotlib Python plotting library provides (via its `pyplot` module) the option `xscale('log')` that sets a logarithmic scale for the x-axis, to produce even spacing between these five R.

[8 marks]

Exercise 2: Inverse transform sampling

Background

The uniform distribution is a bit boring. Inverse transform sampling is a technique that allows us to consider more interesting probability distributions, while still generating pseudo-random numbers using the `random.random()` function. The idea is illustrated by the sketch below.



In words, we take our uniformly distributed u_r and act on them with some invertible transformation $F(u)$ to define $x_r = F(u_r)$ that follow the distribution of interest, $p(x)$. We require $p(u)du = p(x)dx$, which allows us to relate $p(x)$ and the inverse transformation $F^{-1}(x)$:

$$p(x) = p(u)\frac{du}{dx} = p(u)\frac{d}{dx}F^{-1}(x),$$

hence the name "inverse transform sampling". This relation lets us either engineer an appropriate transformation $F(u)$ to produce a desired distribution $p(x)$, or determine the distribution that results from a given transformation.

Task

Based on the uniformly distributed pseudo-random numbers u generated by `random.random()`, define

$$x = F(u) = \arccos(1 - 2u) \,. \tag{4}$$

What is the probability distribution $p(x)$ of these random numbers x? What are the minimum and maximum possible values that x can take? What are the resulting exact mean μ and standard deviation σ of $p(x)$?

[5 marks]

Reset by initializing the random number generator with seed 271828. Now, using the `arccos` function provided by Numerical Python (NumPy), generate $R = 1{,}000{,}000$ pseudo-random numbers x_r via Eq. 4. Use these to numerically estimate the mean and standard deviation of $p(x)$, analogously to Eq. 2 in the previous exercise. How do your numerical results compare to your exact μ and σ above? Four significant figures should suffice for these comparisons.

[5 marks]

In a single plot, compare the histogram of the $1{,}000{,}000$ $\{x_r\}$ to the analytic $p(x)$ you found above. Do your numerical results match your prediction? Roughly 51 bins in the histogram should suffice for this comparison.

Hint: The demo shows how Matplotlib can `plot` a function $p(x)$ on top of a histogram produced using its `hist` routine.

[5 marks]

Exercise 3: Random walks

(a) Central limit theorem

Now consider a random walk that consists of N steps, with the length of each step being a pseudo-random number x_i obtained using Eq. 4. By independently generating R different N-step random walks you can analyze the final positions of the walks,

$$X_r(N) = \sum_{i=1}^{N} x_i \qquad\qquad r = 1, 2, \cdots, R.$$

Based on the central limit theorem, what are the analytic predictions for $\langle X(N) \rangle$ and the diffusion length

$$\ell_2(N) \equiv \sqrt{\left\langle [X(N)]^2 \right\rangle - \langle X(N) \rangle^2},$$

each as a function of N? (**Hint:** $\ell_2(N)$ would be called $\Delta X(N)$ in the lecture notes; the new terminology will be useful for Exercise 5.)

`[2 marks]`

(b) Fixed number of steps

Reset by initializing the random number generator with seed 271828. With fixed $N = 100$, generate R 100-step random walks for each of $R = 10$, 100, 1000, $10{,}000$ and $100{,}000$. Use the five resulting sets $\{X_r\}$ to numerically estimate both

$$\overline{X(N)}_R \equiv \frac{1}{R} \sum_{r=1}^{R} X_r(N) \qquad\qquad \overline{\ell_2(N)}_R \approx \sqrt{\left(\frac{1}{R} \sum_{r=1}^{R} [X_r(N)]^2 \right) - \overline{X(N)}_R^2}.$$

How do your numerical results compare to your exact analytic predictions for $N = 100$? Five significant figures should suffice for this comparison.

`[8 marks]`

(c) Diffusion constant

Reset by initializing the random number generator with seed 271828. Then fix $R = 10{,}000$ and compute $\overline{\ell_2(N)}_R$ for *every* $N = 1, 2, \cdots, 500$. Rather than reporting results as numerical values, plot $\overline{\ell_2(N)}_R$ vs. N. (**Hint:** You can ignore potential correlations between $\overline{\ell_2(N)}_R$ for different values of N.)

`[4 marks]`

Now fit your numerical results to the function

$$\overline{\ell_2(N)}_R = C + D\sqrt{N}.$$

Include your fit in your plot of $\overline{\ell_2(N)}_R$ vs. N. How do your fit results for C and D compare to the exact analytic predictions from the central limit theorem? (**Hint:** NumPy's `polyfit` routine can handle fits linear in $\sqrt{N}$.)

`[6 marks]`

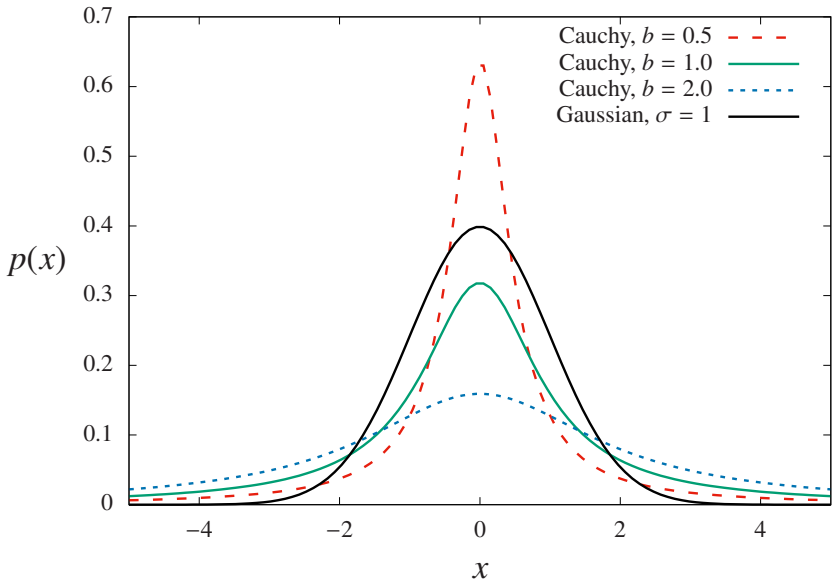
Exercise 4: Cauchy–Lorentz distribution

Background

So far we have been able to verify our numerical results by using the central limit theorem. We now turn to a case in which the central limit theorem will not be applicable, by considering

$$p_C(x) = \left(\frac{1}{b\pi}\right) \frac{1}{1+(x/b)^2} \qquad x \in \mathbb{R}, \tag{5}$$

which is known as the Cauchy–Lorentz (or just Cauchy) distribution. Here b is a constant parameter that controls the width of the peak around $x = 0$. The figure below illustrates this by plotting $p_C(x)$ for each of $b = 1/2$, $b = 1$ and $b = 2$, comparing them to the normal (gaussian) distribution $\frac{1}{\sqrt{2\pi}} e^{-x^2/2}$.



The figure shows how the peak of the Cauchy–Lorentz distribution around $x = 0$ becomes higher and narrower as b decreases. Even when its peak is very narrow, as $|x|$ increases $p_C(x)$ again becomes larger than the gaussian distribution, simply because the latter decreases exponentially quickly while $p_C(x)$ decreases only $\sim 1/x^2$. These "fat tails" at large $|x|$ make the Cauchy–Lorentz distribution both interesting and challenging to analyze.

Task

Fix $b = 1/\pi$ in the Cauchy–Lorentz distribution, so that Eq. 5 becomes

$$p_C(x) = \frac{1}{1+\pi^2 x^2} \qquad x \in \mathbb{R}. \tag{6}$$

What is the integral of this distribution over its full range,

$$I \equiv \int_{-\infty}^{\infty} p_C(x)\, dx = \int_{-\infty}^{\infty} \frac{1}{1+\pi^2 x^2} dx?$$

The usual starting point to analyze a probability distribution is finding its mean and standard deviation, by evaluating

$$\langle x \rangle = \int x\, p(x)\, dx \qquad\qquad \langle x^2 \rangle = \int x^2\, p(x)\, dx.$$

For the Cauchy–Lorentz distribution in Eq. 6, consider instead the functions

$$f(a) = \int_{-a}^{a} x\, p_C(x)\, dx = \int_{-a}^{a} \frac{x}{1+\pi^2 x^2} dx$$
$$g(a) = \int_{-a}^{a} x^2\, p_C(x)\, dx = \int_{-a}^{a} \frac{x^2}{1+\pi^2 x^2} dx.$$

How do $f(a)$ and $g(a)$ behave in the limit $a \to \infty$?

[6 marks]

Turning to a numerical analysis of the Cauchy–Lorentz distribution, the first step is to determine the transform $F(u)$ that will map the uniformly distributed pseudo-random numbers u (Eq. 1) to $x = F(u) \in \mathbb{R}$. What is the transform F that provides $x = F(u)$ distributed according to $p_C(x)$ in Eq. 6?

Hints: Guided by the relation

$$p_C(x) = p(u) \frac{d}{dx} F^{-1}(x),$$

it will suffice to propose an ansatz for $F(u)$ based on integrating $p_C(x)$, and then follow the steps in Exercise 2 to confirm that this ansatz produces the desired distribution. Integrating will introduce a constant of integration, which can be chosen so that $x \to -\infty$ as $u \to 0$ and $x \to \infty$ as $u \to 1$.

[6 marks]

Now initialize the random number generator with seed $s = 271828$. Generate $R = 1{,}000{,}000$ pseudo-random numbers $x_r = F(u_r)$ using the transform you found. Plot the histogram of these million $\{x_r\}$ and check whether it agrees with the Cauchy–Lorentz distribution shown above.

Hints: You will need to set an appropriate range for the x-axis of this histogram. A range $-3 \le x \le 3$ with roughly 200 bins should suffice to show all the interesting features. In Python this can be done by providing

```
bins = np.arange(-3.0, 3.0, 6.0/201.0)
```

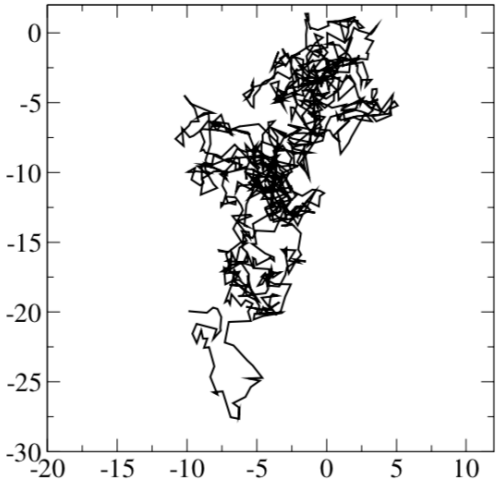
to the Matplotlib `hist` function used previously. In this exercise it is optional to plot $p_C(x)$ itself on top of this histogram. If you choose to do so, you may need to adjust its normalization (and you should think about why this is needed).

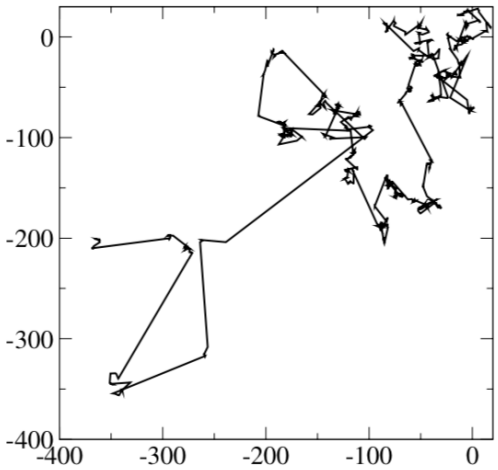
[8 marks]

Exercise 5: Anomalous diffusion

Background

The "fat tails" of the Cauchy–Lorentz distribution mean that $p_C(x)$ provides larger probabilities for *rare events* with large $|x|$ to occur, compared to the gaussian distribution. This is illustrated in the figures below, each of which shows a thousand-step random walk in two dimensions — randomly selecting both the size of each step and the direction $0 \leq \phi < 2\pi$ in which to step. The walk on the left uses step sizes drawn from a gaussian distribution. Even in two dimensions, random walks of this sort obey the law of diffusion, with a diffusion length growing proportionally to the square root of the number of steps, $\ell_2(N) \propto \sqrt{N}$.





The walk on the right instead uses step sizes drawn from a Cauchy–Lorentz distribution. Note that the axes for this figure cover a much larger range! The fat tails of the Cauchy–Lorentz distribution result in occasional very large jumps, leading to random walks that do not obey the law of diffusion.

Returning to one-dimensional random walks, some of the results from Exercise 4 motivate defining the **generalized diffusion length**

$$\ell_\theta(N) = \left\langle |X(N)|^\theta \right\rangle^{1/\theta}, \tag{7}$$

which depends on a positive real parameter $\theta > 0$. Since θ is not necessarily an integer, the absolute value is needed to ensure $\ell_\theta \in \mathbb{R}$, rather than becoming complex valued. If $\langle X(N) \rangle = 0$ and ℓ_θ is well-defined with $\theta = 2$, then this generalized diffusion length could reproduce the standard deviation ℓ_2 and exhibit the ordinary law of diffusion, $\ell_2 \propto N^{1/2}$.

For the Cauchy–Lorentz distribution, ℓ_θ is ill-defined for any $\theta \geq 1$. This parameter θ can take only values $0 < \theta < 1$. The resulting ℓ_θ exhibits **anomalous diffusion**,

$$\ell_\theta(N) \propto N^\alpha,$$

where the exponent is either $\alpha > \frac{1}{2}$ (called *super-diffusion*) or $0 < \alpha < \frac{1}{2}$ (called *sub-diffusion*). This exercise investigates the exponent α for the distribution $p_C(x)$ in Eq. 6, and explores whether or not α depends on θ.

Task a: Fixed number of steps

Reset by initializing the random number generator with seed 271828. With fixed $N = 100$, generate R 100-step random walks,

$$X_r(N) = \sum_{i=1}^{N} x_i \qquad\qquad r = 1, 2, \cdots, R,$$

for each of the five $R = 10$, 100, 1000, $10{,}000$ and $100{,}000$. Use the five resulting sets $\{X_r\}$ to numerically estimate

$$\overline{\ell_\theta(N)}_R \approx \left[\frac{1}{R} \sum_{r=1}^{R} |X_r(N)|^\theta \right]^{1/\theta}$$

for three values of $\theta = 0.1$, 0.5 and 0.9. (**Hint:** NumPy provides both an `abs` function to take the absolute value, and a `power` function to compute non-integer powers.)

[12 marks]

Task b: Anomalous diffusive exponent

Reset by initializing the random number generator with seed 271828. Then fix $R = 10{,}000$ and estimate $\overline{\ell_\theta(N)}_R$ for *every* $N = 1, 2, \cdots, 500$, again considering $\theta = 0.1$, 0.5 and 0.9. Instead of reporting your numerical results, plot all three $\overline{\ell_\theta(N)}_R$ vs. N in a single figure. (**Hint:** You can ignore potential correlations between $\overline{\ell_\theta(N)}_R$ for different values of N.)

[8 marks]

Now fit your numerical results for each $\theta = 0.1$, 0.5 and 0.9 to the function

$$\overline{\ell_\theta(N)}_R = DN^\alpha.$$

Report your results for D and α, and comment on their sensitivity to the value of θ. (**Hint:** Optionally testing different values of R, N or θ may help to distinguish between real sensitivity vs. statistical fluctuations, if you are unsure whether or not an observed effect is significant.)

[10 marks]