MATH327: StatMech and Thermo, Spring 2025

Computer Assignment

Overview and instructions

In this computer assignment you will numerically analyse two types of diffusive behaviour in one-dimensional random walks. After warm-up exercises on pseudo-random numbers and inverse transform sampling, analysing 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 assignment. For any reasonable programming language, the numerical computations for each exercise should complete in a few minutes or less.

This assignment is **due by 17:00 on Friday, 21 February 2025**. Submit your work by file upload on Canvas. **Both** your answers to the questions below (including plots) **and** the code that produces your results must be submitted. You are free to upload multiple files in various formats, a combined tar/zip archive, or a single document including everything, as you prefer. For example:

- A common approach in past years was to answer the non-computational questions using pen and paper, and then photograph / scan this using either an app like CamScanner or an actual scanner (available in our Library). If you have trouble saving the plots as pdf or png files, you can submit screenshots of them.
- Alternatively, the answers can be typed up into a pdf (which can also include the plots), for instance using MS Word, LibreOffice Writer, or LaTeX.
- It's also possible to put everything (code, plots, and non-computational work) into a single Jupyter (ipynb) notebook. You may have noticed that the demo takes this approach, using LaTeX to format mathematical expressions.

Similarly, the code can all be in a single file, or you can use a different file for each task. For whichever method you choose, make sure to include enough intermediate steps for me to be able to follow your logic. The correctness of the submitted code contributes to the marks listed below. While no marks will be deducted for stylistic reasons, it is in your interest to make your code as readable as possible, to simplify my marking and your debugging

Marking is typically slower for this assignment compared to traditional homework, due to the need to check both the code and the non-computational work. You can help to speed up this process by submitting code in its native format (for example, a `.py` file for Python code or a `.m` file for MATLAB code). Use of resources beyond the module materials (for example, asking ChatGPT to help debug your code) must be explicitly

referenced in your submissions. Clear and neat presentations of your workings and the logic behind them will contribute to your mark. Anonymous marking is turned on, and I will aim to return feedback by Monday, 10 March 2025.

You should already be familiar with the Department's academic integrity guidance for 2024–2025, which states that 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 that you have successfully passed the Academic Integrity Tutorial and Quiz in the course of your studies. You also affirm that the work you are submitting is your own and you have not commissioned production of the work from a third party or used artificial intelligence (AI) software in an unacceptable manner to generate the work. (Generative AI software applications include, but are not limited to, ChatGPT, Bing Chat, DALL.E and Bard.) You also affirm that you have not plagiarized material from another person or source, nor fabricated, falsified or embellished data when completing this assignment. You also affirm that you have not colluded with any other student in the preparation or production of your work. The marks achieved on this assessment remain provisional until they are ratified by the Board of Examiners in June 2025.

Exercise 1: Pseudo-random numbers

Background

We have discussed how statistical mechanics is based on incorporating an element of randomness into analyses of complex systems. Because computer programs are deterministic, they do not produce true randomness.[1] 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 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 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.

[1] Quantum computers and related devices promise the potential to generate truly random information. This is part of the motivation for large investments in quantum technologies around the world.

Task

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

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

Clearly $\int p(u)\,du = \int_0^1 du = 1$, as required. What are the exact mean $\mu = \langle u \rangle$, expectation value $\langle u^2 \rangle$, and standard deviation σ of this probability distribution?

`[3 marks]`

Initialize the PRNG with seed $s = 327$. 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 compute

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

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 five computations of $\overline{u}_R$ another 99 times, ignoring $\overline{\sigma}_R$ for simplicity. Together with the results you reported above, this gives a total of 100 estimates of the random variable $(\overline{u}_R - \mu)^2$ for each R, which we can use to approximate each 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?

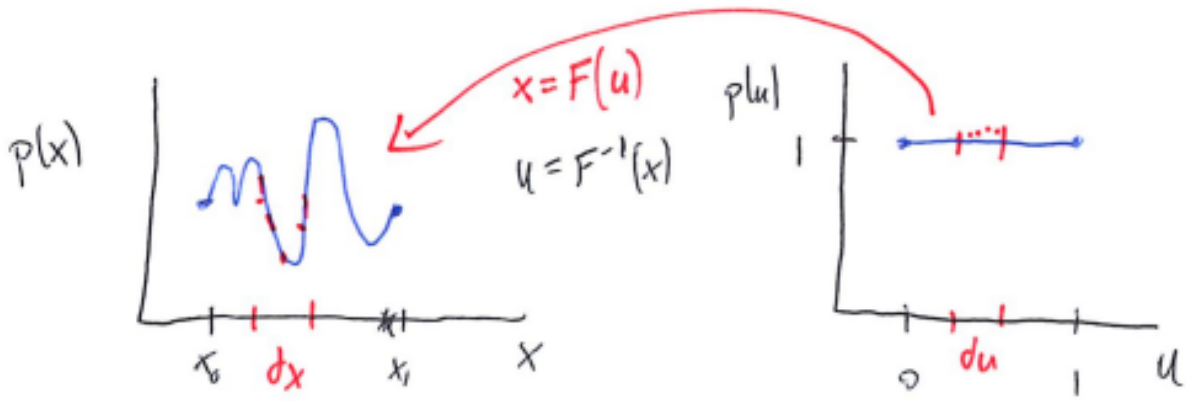
Hints: Include 0 on the y-axis of your plot to maintain a sense of scale. Python's Matplotlib plotting library provides (via its `pyplot` module) the option `xscale('log')` that sets a logarithmic scale for the x-axis, to produce even spacing between the five values of R.

`[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 relates $p(x)$ and the inverse transformation $F^{-1}(x)$:

$$p(x) = p(u)\frac{du}{dx} = p(u)\frac{d}{dx}F^{-1}(x), \tag{4}$$

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

Using `random.random()` to generate uniformly distributed pseudo-random numbers u, define

$$x = F(u) = \frac{\arccos(1-2u)}{\pi}. \tag{5}$$

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 $p(x)$?

Hint: You can look up derivatives or integrals, citing any sources you use.

`[5 marks]`

Reset by initializing the PRNG with seed 327, and generate $R = 1\,000\,000$ pseudo-random numbers x_r via Eq. 5. Use these to numerically estimate the mean and standard deviation of $p(x)$. How do your numerical results compare to your exact μ and σ above? Five significant figures should suffice for these comparisons.

Hint: Numerical Python (NumPy) provides an `arccos` function that may be useful.

`[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.

`[4 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. 5. By independently generating R different N-step random walks you can analyse 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 in the limit $R \to \infty$, what are the analytic predictions for $\langle X(N) \rangle$ and the diffusion length

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

each as a function of N? [In the lecture notes, $\ell_2(N)$ is called $\Delta X(N)$; the new terminology will be useful for Exercise 5.]

`[2 marks]`

(b) Fixed number of steps

Reset by initializing the PRNG with seed 327. 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

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

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

`[8 marks]`

(c) Diffusion constant

Reset by initializing the PRNG with seed 327. 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.)

`[6 marks]`

Now fit your numerical results to the function

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

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

`[4 marks]`

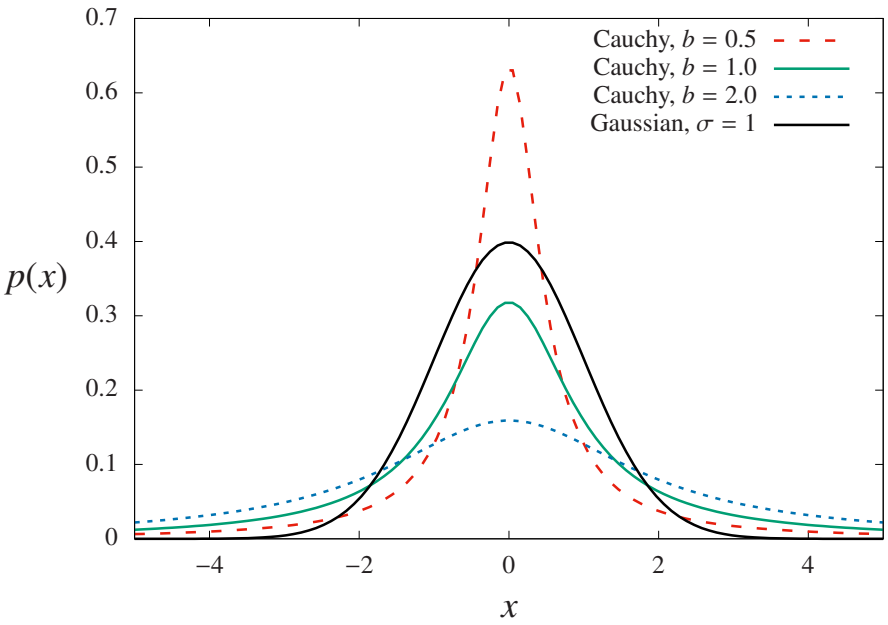
Exercise 4: Cauchy–Lorentz distribution

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

$$p_C(x) = \left(\frac{1}{b\pi}\right) \frac{1}{1+(x/b)^2} \qquad x \in \mathbb{R}, \tag{6}$$

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 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 analyse.

Task

Fix $b = 1/2$ in the Cauchy–Lorentz distribution, so that Eq. 6 becomes

$$p_C(x) = \left(\frac{2}{\pi}\right) \frac{1}{1+4x^2} \qquad\qquad x \in \mathbb{R}. \tag{7}$$

Evaluate the integral of this distribution over its full range, $\int_{-\infty}^{\infty} p_C(x)\, dx$.

[2 marks]

Our usual starting point to analyse a probability distribution $p(x)$ is to find 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. 7, consider instead the functions

$$f(a) = \int_{-a}^{a} x\, p_C(x)\, dx = \frac{2}{\pi} \int_{-a}^{a} \frac{x}{1+4x^2}\, dx$$
$$g(a) = \int_{-a}^{a} x^2\, p_C(x)\, dx = \frac{2}{\pi} \int_{-a}^{a} \frac{x^2}{1+4x^2}\, dx\,.$$

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

[4 marks]

To numerically analyse 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. 7?

Hints: Guided by Eq. 4, 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. You can choose the constant of integration so that $x \to -\infty$ as $u \to 0$ and $x \to \infty$ as $u \to 1$.

[6 marks]

Now reset by initializing the PRNG with seed $s = 327$. 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 $-4 \le x \le 4$ with roughly 201 bins should suffice to show all the interesting features. In Python this can be done by providing

```
bins = np.arange(-4.0, 4.0, 8.0/201.0)
```

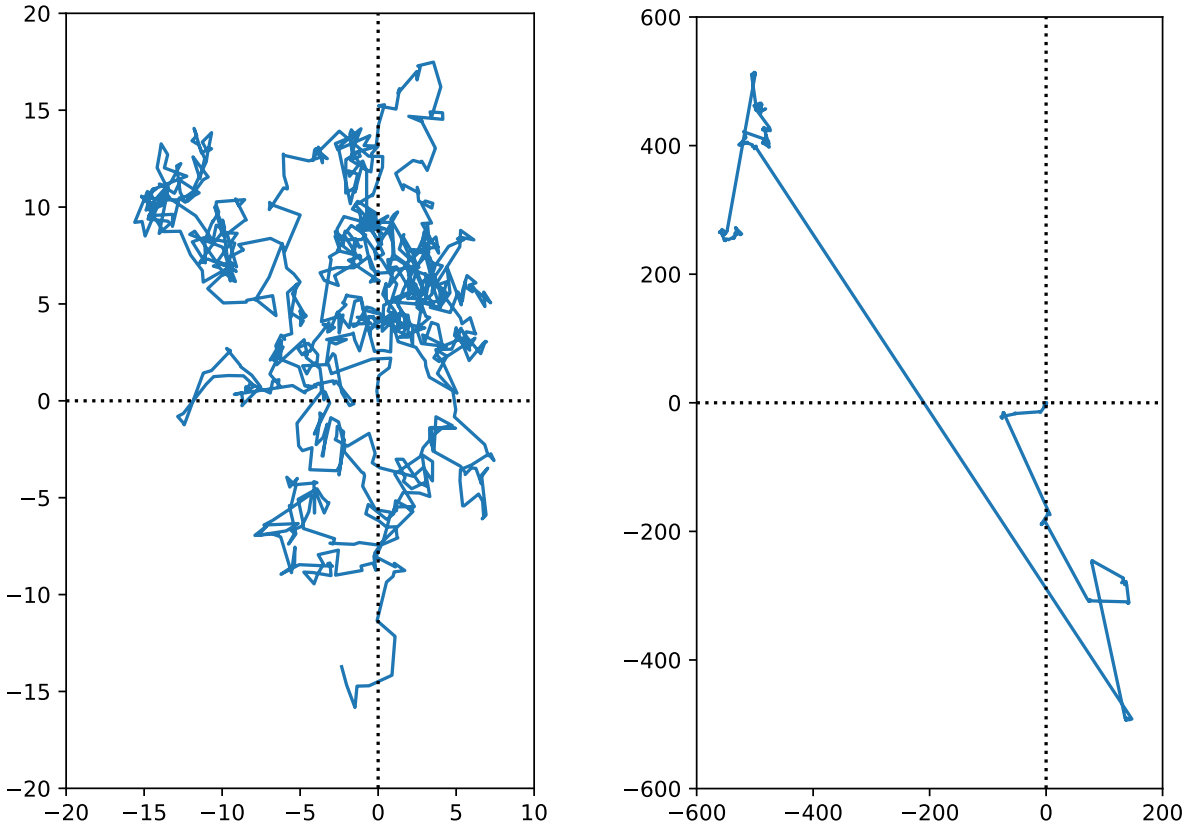
to the Matplotlib `hist` function used previously. In this exercise it is optional to plot $p_C(x)$ itself on top of this histogram. If you choose to do so, you may need to adjust its normalization (and you should think about why this is needed).

[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**

$$\ell_\theta(N) = \left\langle |X(N)|^\theta \right\rangle^{1/\theta}, \tag{8}$$

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 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. 7, and explores whether or not α depends on θ.

Task a: Fixed number of steps

Reset by initializing the PRNG with seed 327. With fixed $N = 100$, generate R 100-step random walks using the transform you found in Exercise 4, computing

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

for each of the four $R = 100$, 1000, $10\,000$ and $100\,000$. Use the four 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 PRNG with seed 327. Then fix $R = 10\,000$ and estimate $\overline{\ell_\theta(N)}_R$ for *every* $N = 1, 2, \cdots, 250$, 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]