This is going to be a lengthy multi-week activity.

1 DNA as a random walk

DNA can be modeled as a chain of segments each with length L, where neighboring segments are freely hinged. Real DNA actually acts more like a rubber hose than a chain, but provided the length L is chosen wisely, the two give identical predictions for long DNA molecules. This model of DNA gives shockingly accurate results for the force on the two ends as a function of the distance between them. Mathematically, this problem is called a "random walk," since the displacement corresponding to each segment is in a random direction. Random walks can also be used to describe diffusion, repeated measurements, or any number of other physical scenarios.

Missing /var/www/paradigms_media_2/media/activity_media/dna-chain.pdf

2 Monte Carlo

A Monte Carlo method is a way to compute something using random numbers. There are a wide variety of Monte Carlo algorithms, used for anything from integration to summation. In general, Monte Carlo algorithms are helpful when you have something that you can't solve analytically, and it is computationally prohibitive to just brute-force it.

We will apply a Monte Carlo simulation to examining DNA chains. For a long chain, it would be hard to attempt to computationally evaluate every possible combination of orientations for all N links, which makes the Monte Carlo approach appealing.

We will examine the distance in the x direction between the beginning and end of the chain. This quantity is relevant when an experimentalist attaches those ends to a fixed object and the other to a bead (which could be left free or held in optical tweezers). We will limit your DNA to move in two dimensions for ease of visualization, although of course real DNA can flop in three dimensions.

Your single-link tasks

1. Write code to pick a random angle for the link of DNA, and from this to find the x and y coordinates of the end of the link. The beginning of the link is at the origin.

2. Find the average value of x^2 by picking many random angles and averaging the x^2 for each one.

3. Find the average value of x in the same way. Does it match what you expect?

Extra fun Plot the average value of x versus the number of random orientations used in the simulation. Also plot the average value of x^2 in the same way.

Log fun Create a log-log plot of the extra-fun data, and explain the shape of the curve.

When you have finished this task, please raise your hand to show your result with a member of the teaching team!

Your multi-link tasks Please create a new python file for this task. Note that x and y are the coordinates of the *end* of the **entire** chain.

1. Write code to pick a configuration of N random angles describing a chain with N links, each with length L. I encourage you to try several different values for N, but among others please experiment with $N = 100$.

2. Write a program to plot a random chain. Each line segment on you plot should have an equal length, since each link has an equal length. You should be able to rerun your program a few times in order to see a few random chains.

3. Find the average value of x^2 for your N-link chain by averaging over many random chains. How does this compare with the average value of x^2 for a single-link chain?

4. Plot the average value of x^2 versus the number of links in the chain. Look for an analytic function that will match this plot. *Note that this will require computing a separate average for each chain length you consider.*

5. Plot the average value of $\sqrt{x^2 + y^2}$ versus the number of links in the chain. Look for an analytic function that will match this plot. *Note that this will require computing a separate average for each chain length you consider.*

Less bendy Adjust your code to constrain the relative angle between two neighboring segments to have less than a given magnitude. How does that change your results?

3 Probability density

We have now calculated the average values of x and x^2. These are in themselves interesting quantities, but that is a pretty limited result. Things get more interesting (and harder) if we can predict the distribution of values for x, which we call the probability density. The **probability density** $P(x)$ is the probability per x that the system will have a value x. Another perspective is that $P(x)\Delta x$ tells us the fraction of systems that have x values between $x - \Delta x/2$ and $x + \Delta x/2$. In many cases a probability density will tell us all that we want to know about a system.

In order to compute $P(x)$, we are going to need to make it discrete, since we can't in a finite amount of time compute an infinite number of values (for all possible x). We do this by dividing the domain of x up into *bins*. A **bin** corresponds to a range of x values, so that we can count how many times we observe systems in that range.

Old-style cubits A cubit is a measure of length from your elbow to your longest finger. It's got a problem in that each of us have our own cubit. Let's investigate the distribution $P(c)$ of cubits among physicists. To do this, we'll need to take lots of measurements (one class-worth) and collect data. Odds are that none of us have an equal cubit, so we'll need to divide up the range of possible cubits into bins. I'll pick bins with a width of 0.5 cm, and draw them on the board. *Please measure your cubit, and then come to the board and add a tally mark in the appropriate bin.*

We can then approximate $P(c)$ as having a value that is $\frac{n}{0.5\text{ cm}}$ in each bin.

I plan to approximately plot our probability distribution of cubits for the class.

Average notation In physics, we often use the notation where $\langle x \rangle$ refers to the average value of x, and $\langle x^2 \rangle$ refers to the average value of x^2. Computing an average such as this using Monte Carlo requires that you construct many random systems (in our case DNA chains) and then compute the average by adding up the x (or x^2) for each chain, and dividing by the total number of chains.

As a suggestion (but not a requirement), your code may be easier to write and to read if you separate the process of creating a random DNA chain and computing the x which is its final coordinate into a separate function. This way the "create a random system" logic can be separated from the "average over many random systems" logic.

Note that once you have an estimate for the probability density $P(x)$, you can compute averages over the distribution using

$$\langle x^a \rangle = \int P(x) x^a dx \tag{1}$$

This is a major advantage of knowing the probability distribution: it allows you to compute averages that you didn't previously consider.

Single-link probability density task Please create a new python file for this task. Note that once again x and y are the coordinates of the *end* of the chain, but now the chain will have just one link in it.

1. Create a `numpy` array `n` with length N_{bins} to hold the number of times you have observed x in each bin. This array should initially be full of zeros, since you haven't yet observed any x values.

2. Generate a random configuration for a single-link chain and compute x, and then figure out which bin x is in. This bin will be given by:

$$\text{bin number} = \frac{x + N \cdot L}{2N \cdot L} N_{\text{bins}} \text{ (rounded down to an integer)} \tag{2}$$

   where N is the number of links in the chain (1 in this case), L is the length of each link, and N_{bins} is the number of bins.

   Once you have this bin number, you will want to increment the corresponding element of your `n` array.

3. Create a loop that generates many configurations of your single-link chain, and for each one increments the count for the appropriate bin.

4. Create a probability density array by dividing by the number of samples you took and by $\Delta x = \frac{2NL}{N_{\text{bins}}}$, the size of each bin. In this case $N = 1$, but it's worth including the factor of N to avoid problems on the next task.

5. Plot the probability density $P(x)$ for a single-link chain.

6. On your $P(x)$ plot, draw a vertical line at the square root of the average x^2, i.e. $\sqrt{\langle x^2 \rangle}$.

Multi-link probability-density task Having examined the probability density for x of a single-link DNA chain, we will now examine the probability density for an N-link chain. Conceptually, the code to compute the probability density will look very much like the single-link code, except that the range of possible x values will be wider. Once again, please start a new file for this program.

1. Create a `numpy` array called `n` with length N_{bins} to hold the number of times you have observed x in each bin. You will probably need a greater value of N_{bins} than you were able to use for a single link.

2. Generate a random configuration for an N-link chain and compute x, and then figure out which bin x is in. This bin will be given by:

$$\text{bin number} = \frac{x + N \cdot L}{2N \cdot L} N_{\text{bins}} \text{ (rounded down to an integer)} \tag{3}$$

   where N is the number of links in the chain, L is the length of each link, and N_{bins} is the number of bins.

   Once you have this bin number, you will want to increment the corresponding element of your `n` array.

3. Create a loop that generates many configurations of your single-link chain, and for each one increments the count for the appropriate bin.

4. Create a probability density array by dividing by the number of samples you took and by $\Delta x = \frac{2NL}{N_{\text{bins}}}$, the size of each bin.

5. Plot the probability density $P(x)$ for your N-link chain.

6. On your $P(x)$ plot, draw a vertical line at the square root of the average x^2, i.e. $\sqrt{\langle x^2 \rangle}$.

7. Set $N = 1$ to confirm that in this case your N-link chain gives the same prediction as the single-link code.

Extra fun Adjust your code to constrain the relative angle between two neighboring segments to have less than a given magnitude (e.g. between $-\pi/2$ and $\pi/2$), making the DNA less flexible. How does that change your results?