6.S083 Fall 2019: Problem set 2

Submission deadline: Friday November 8, 2019 at 5pm.

Exercise 1: Frequencies

1. Write a function `counts` that accepts a vector `data` and calculates the number of times each value in `data` occurs.

   To do so, use a `Dict` called `counts`; use the `haskey` function to find out whether the `Dict` contains a given key or not.

   Test that your code is correct with the data vector `[1, 0, 1, 0, 100]`.

   You will use this function throughout this problem set.

Exercise 2: Sum of two dice

Consider rolling two (unbiased, 6-sided) dice.

1. Write a function `roll_dice(n)` to roll n dice and return a vector of the results.

2. Write a function `sum_dice(n)` to calculate the *sum* of the result of rolling n dice. This function should use the function `roll_dice`.

3. Write a function `sum_experiment`. This should take two parameters, the number of dice `num_dice` and the number of experiments `num_experiments`. It should use the functions that you already wrote to sum `num_dice` dice rolls for `num_experiments` times, and return a vector of the results.

4. Count how many times each outcome occurs and make a bar graph of the resulting probability distribution of the random variable "sum of n dice" for $n = 2$. Make sure that you **normalize** the distribution to display probabilities instead of frequencies.

   You may use the functions `keys` and `values` to extract the keys and values from a dictionary. You may need to convert the resulting objects into arrays using the `collect` function.

   What shape does this distribution have?

5. In this case, we can actually calculate the mathematically *exact* probability distribution of the population, equivalent to an infinite number of dice rolls, as follows.

   The possible outcomes are pairs (i, j) of integers, each of which (we are supposing) is equally likely. Iterate over all possible pairs, count the number of states with each value of the sum, then convert into *probabilities*.

6. Plot this exact probability distribution on the same figure as the approximate one obtained in [4].
7. Repeat [4]–[6] for the sum of 3 dice and for the sum of 4 dice. (You do not need to write a general function for n dice, but you may do so if you prefer.)
8. What type of probability distribution do we seem to get as the number of dice gets larger?

Exercise 3: Width of a probability distribution

Consider the problem of tossing a single unbiased coin with possible outcomes that we will represent as ± 1, where "tails" corresponds to -1.

1. Write a function `num_tails(n)` that counts the number of tails that occur in n tosses.
2. Write a function `tails_experiment(num_tosses, num_times)` that repeats [1] a certain number of times and returns a vector of the data "number of tails for each run".
3. Fix `num_times` as 30 and run `tails_experiment` as a function of N for N between 1 and 1000. Collect vectors `means` and `variances` of the mean and variance of the resulting data.
4. Plot the means as a function of N. What is the behaviour as a function of N (i.e. guess which function of N it is)? Plot the variances as a function of N. What is the behaviour?
5. Plot the standard deviation as a function of N. How does it change with N?
6. If you want to estimate the probability of tails, you would divide the number of tails by N, and the error estimate by N too. How would that error then behave? This shows how much better you can do by taking more data in the calculation.

Exercise 4: Distribution of the "atmosphere"

In this question we will implement a (very!) simple model of the density of the atmosphere via a random walker.

Due to gravity, the walker has a higher probability p of moving downwards (to $x-1$) than upwards (to $x+1$), with probability $1-p$.

At $x = 1$ there is a **boundary condition**: it hits a reflective boundary (the "surface of the Earth"). We can model this using "bounce-back": if the particle tries to jump downwards from $x = 1$, it hits a reflective boundary and bounces back to $x = 1$ (i.e. it remains in the same place).

1. Write a simulation of this model in a function `atmosphere` that accepts `p`, the initial height `x0`, and the number of steps N as variables.

2. Simulate it for 10^7 time steps with $x_0 = 10$ and $p = 0.55$ and store the walkers's position during the simulation in a vector.

3. Plot the normalized probability distribution that the walker is at each height i.

4. What does the resulting figure look like? What form do you think this distribution has? Verify your hypothesis by plotting the distribution using different scales. You can increase the number of time steps to make the results look nicer.

5. Make an interactive visualization of how the distribution develops over time. What happens for longer and longer times?

Exercise 5: Number of sites visited

Suppose we model an animal or molecule wandering around by a random walker. We might be interested in the question "how much territory is covered in a given time?". The number of sites visited up to time N by a random walker turns out to be quite difficult to calculate analytically, so let's do it computationally.

1. Write a function `number_sites_visited` that takes a parameter N, the total time. Initialize the walker at site $x = 0$ and let it jump left and right with equal probability.

   To keep track of *which* sites are visited, you can use a `Set`. This is a new data structure that contains at most one copy of each element.

   The function should return the number of sites visited at each time n from 1 to N.

2. Plot a couple of runs with $N = 10^5$. What kind of behaviour do you see?

3. Run the function 100 times and average the resulting *vectors* to find the mean number of sites visited at time n from 1 to N.

4. Plot this and use different scales to guess how it grows over time (e.g. is it linear? exponential?) Add the function that you guess to your graph.

   You can add a function plot with e.g. `plot!(x -> x^2)`.

Exercise 6: Hitting a target

Think of a molecule moving inside a cell in your body. The molecule needs to reach the entrance of a channel embedded in the cell wall in order to leave the cell.

We will model this as a random walker in a 2-dimensional square box of size $L \times L$. The walls are reflective, except for a window of width w that starts in the center of the top wall, where the walker is *absorbed* (exits the cell). The walker starts in the center of the cell. We call the time taken to reach the window the **exit time**, **hitting time**, or **first-passage time**.

1. Write a simulation of the 2D walker with coordinates x and y. At each time step, it chooses whether to move horizontally, with probability $1/2$, or vertically, with probability $1/2$, and then in each direction whether to add 1 or subtract 1 in that direction.

   If it tries to jump over a reflective wall, it returns to the same spot. The simulation continues *until* the walker reaches the window. [What kind of control flow construct should you use to implement this?]

   To avoid the simulation running "forever", add a counter of the number of steps taken, and exit with an error message (using the `error` function) if the number of steps taken is too huge.

2. Write an interactive visualization of one trajectory of the walker moving around the box as a function of time. You should draw the walker's "trail" over the whole history of its motion, as well as a point at its current position.

3. Write a function `exit_time(L, w)` that returns only the exit time, as a function of L and w.

4. Plot the probability distribution of the exit time for $L = 6$ and $w = 1$, for example over $N = 10^6$ runs. What type of distribution does it seem to be?

5. If your computer is fast enough, investigate how the mean exit time depends on the geometrical parameters L and w of the system.