6.S083 / 18.S190 Problem set 2: Probability and modelling recovery

Submission deadline: 11:59pm on Tuesday, April 14

In this problem set we will look at probability distributions and how to model recovery from an infection.

Note: If you are unable to get `Interact.jl` to work to create interactive visualizations, note this fact on your pset submission. Where it says to create an interactive visualization, instead write a function that creates one of the plots, and run that function for different parameter values manually, showing the resulting plots.

Similarly, if you have problems running simulations for larger values of the parameters, just do what you can and make a note.

Exercise 1: Frequencies

In this exercise we will write a more general function to count occurrences of values in a set of data. In class we used as `Vector`, but that is difficult to use with data that jumps around a lot.

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 **dictionary** called `counts`.
   A dictionary maps a **key** to a **value**. We want to map integers to integers, so we create an empty dictionary with

   ```
   d = Dict{Int, Int}()
   ```

   Use the `haskey` function to find out whether the `Dict` contains a given key or not.

   Use indexing to add a new (key, value) pair, or to retrieve a value:

   ```
   d[3] = 1
   d[3]
   ```

   The function should return the dictionary.

2. Test that your code is correct by applying it to obtain the counts of the data vector `vv = [1, 0, 1, 0, 1000, 1, 1, 1000]`. What should the result be?

The dictionary contains the information as a sequence of **pairs** mapping keys to values. This is not a particularly useful form for us. Instead we would prefer a vector of the keys and a vector of the values, sorted in order of the key.

Make a new version of the `counts` function where you do the following (below). Start off by just running the following commands each in their own cell on the

dictionary you got by running the previous `counts` function on the vector `vv` so that you see the result of running each command. Once you have understood what's happening at *each* step, add them to the `counts` function in a new cell.

3. Extract vectors `ks` of keys and `vs` of values using the `keys()` and `values()` functions and convert the results into a vector using the `collect` function.
4. Define a variable `p` as the result of running the `sortperm` on the keys. This gives a **permutation** that tells you in which order you need to take the keys to give a sorted version.
5. Use indexing `ks[p]` to return the sorted keys and values vectors.

   [Here we are passing in a *vector* as the index. Julia extracts the values at the indices given in that vector]
6. Test that your new `counts` function gives the correct result for the vector `v` by comparing it to the true result (that you get by doing the counting by hand!)
7. Make a function `probability_distribution` that normalizes the result of `counts` to calculate the relative frequency, i.e. to give a probability distribution (i.e. such that the sum of the resulting vector is 1).

   The function should return the keys (the unique data that was in the original data set, as calculated in `counts`, and the probabilities (relative frequencies).

   Test that it gives the correct result for the vector `vv`.

   We will use this function in the rest of the exercises.

Exercise 2: Modelling recovery

In this exercise, we will investigate the simple model of recovery from an infection that was described in lectures. We want to study the time τ to recover.

In this model, an individual who is infected has probability p to recover each day. If they recover on day n then $\tau = n$. We see that τ is a random variable, so we need to study its **probability distribution**.

1. Define the function `bernoulli(p)` from lectures. Recall that this generates `true` with probability p and `false` with probability $(1 - p)$.
2. Write a function `geometric(p)`. This should run a simulation with probability p to recover and wait *until* the individual recovers, at which point it returns the time taken to recover.
3. Write a function `experiment(p, N)` that runs the function from [2] `N` times and collects the results into a vector.

4. Run an experiment with $p = 0.25$ and $N = 10,000$. Plot the resulting probability distribution, i.e. plot $P(\tau = n)$ against n, where n is the recovery time.

5. Calculate the mean recovery time and add it to the plot using the `vline!()` function and the `ls=:dash` argument to make a dashed line.

   Note that `vline!` requires a *vector* of values where you wish to draw vertical lines.

6. What shape does the distribution seem to have? Can you verify that by using one or more log scales?

7. Write an interactive visualization that repeats [4] for p varying between 0 and 1 and N between 0 and $100,000$.

   As you vary p, what do you observe? Does that make sense?

   Note that you can make a range for p using something like `0:0.01:1.0`, and you can make a `@manipulate` with additional sliders using a double `for` loop: `for a in 1:10, b in 1:10`.

8. Fix $N = 10,000$ and calculate the *mean* time $\langle \tau(p) \rangle$ to recover. Plot this as a function of p. Can you find the relationship between the two quantities?

Exercise 3: More efficient geometric distributions

Let's use the notation $P_n := \mathbb{P}(\tau = n)$ for the probability to recover on the nth step.

Probability theory tells us that in the limit of an infinite number of trials, we have the exact results $P_1 = p$, that $P_2 = p(1-p)$, and in general $P_n = p(1-p)^{n-1}$.

1. Fix $p = 0.25$. Make a vector of the values P_n for $n = 1, \ldots, 50$. You must use a loop or similar construction; do *not* do this by hand!

2. Plot P_n as a function of n. Compare it to the result from the previous exercise (i.e. plot them both on the same graph).

   How could we measure the *error*, i.e. the distance between the two graphs? What do you think determines it?

If p is *small*, say $p = 0.001$, then the algorithm we used in Exercise 2 to sample from geometric distribution will be very slow, since it just sits there calculating a lot of `false`s! (The average amount of time taken is what you hopefully found in [1.8])

Let's make a better algorithm. Think of each probability P_n as a "bin" of length P_n. If we lay those bins next to each other starting from P_1 on the left, then P_2, etc., there will be an *infinite* number of bins filling up the interval between 0 and

1. (In principle there is no upper limit on how many days it will take to recover, although the probability becomes *very* small.)

Now suppose we take a uniform random number r between 0 and 1. That will fall into one of the bins. If it falls into the bin corresponding to P_n, then we return n as the recovery time!

3. To draw this picture, we need to add up the lengths of the lines from 1 to n for each n, i.e. calculate the **cumulative sum**. Write a function `cumulative_sum`, which returns a new vector. (Of course, you should only do this for a finite number of values! Say those that you found in [2].)
4. Plot the resulting values on a horizontal line. Generate a few random points and plot those. Convince yourself that the probability that a point hits a bin is equal to the length of that bin.
5. Calculate analytically the sum of P_1 up to P_n. (Hint: This should be a calculation that you did in high school or in Calculus I.)
6. Use the result of [5] to find analytically which bin n a given value of $r \in [0, 1]$ falls into using the inequality $P_{n+1} \leq r \leq P_n$.
7. Implement this using the `floor` function.

Exercise 4: A simple infection model

In this exercise we will investigate a *highly* simplified model of the process of infection and recovery. (In the next problem set we will develop a much better model.)

The model is as follows: An individual starts in state `S` ("susceptible"). When they are in state `S`, they have a probability p_E to become exposed (state `E`) at each step. Once they are exposed, they have probability p_I to become infectious (state `I`). When they are infectious, they have a probability p_R to recover at each step.

Let's denote by τ_S the length of time spent in state `S`, and similarly for τ_E and τ_I.

1. How does the total time τ_{total} to go from `S` to `R` relate to these times? What is the relation with the geometric random variables from the previous exercises?
2. Write a function `total_time(p_E, p_I, p_R)` that calculates the total time to go from state `S` to state `R`.
3. Run a Monte Carlo simulation to calculate and plot the probability distribution of τ_{total} for $p_E = 0.25$, $p_I = 0.1$ and $p_R = 0.05$.
4. What happens to the probability distribution of the total time as you add more states, say `E_1`, `E_2`? You may suppose that the probability to move

to the next state is the same value p for all states. How could you use s such states?

[This is a visual representation of a famous theorem that we will discuss in class.]

5. **Extra credit:** Write a simulation that runs N individuals and keeps track at each step of how many people are in which state. Plot the resulting graph of the number of people in the `S`, `I`, `E` and `R` states as a function of time.

Exercise 4: Helping with transcripts

Correct another 10 + 20 lines of the transcripts (see problem set 1 for details) and report which ones you did.