Project I: Probability via Simulation

CSE 107 Probability and Statistics for Engineers

Prof. Chen Qian

1 Motivation

Even though we have learned several techniques for computing probabilities, and have more to go, it is still hard sometimes. Imagine I asked the question: "Suppose I randomly shuffle an array of the first 100 integers in order: [1, 2, ... , 100]. What is the probability that exactly 13 end up in their original position?" I'm not even sure I could solve this problem, and if so, it wouldn't be pretty to set up nor actually type into a calculator.
But since you are a computer scientist, you can actually avoid computing hard probabilities! You could also even verify that your hand-computed answers are correct using this technique of "Probability via Simulation".

2 Probability via Simulation

We first need to define another notion or way of thinking about a probability. If we had some event E, then we could define $\mathbb{P}(E)$ to be the long-term proportion of times that event E occurs in a random experiment. That is

$$\frac{\#\text{of trials where E occured}}{\#\text{number of trials}} \to \mathbb{P}(E)$$

as the number of trials goes to ∞.
For example, if E is the event we roll a 4 on a fair six-sided die, the probability is $\mathbb{P}(E) = 1/6$. That means, if I were to roll this die 6 million times, I should expect to see about 1 million 4's! In reverse, if I didn't know $\mathbb{P}(E)$ and wanted to compute it, I could just simulate many rolls of this fair die! Obviously, the more trials, the better your estimate. But you can't possibly sit around forever rolling this die - a computer can do this MUCH faster, simulating millions of trials within seconds.

Example. Suppose a weighted coin comes up heads with probability 1/3. How many flips do you think it will take for the first head to appear? Use code to estimate this average!

Solution. The first thing we need to do is to simulate a single coin flip. Recall that to generate a random number, we use the numpy library in Python:

```
np.random.rand () #return a single float in the range [1, 0)
```

Since np.random.rand() returns a random float between $[1, 0)$, this function returns a value $< p$ with probability exactly p! For example if $p = 1/2$, then np.random.rand() $< 1/2$ happens with probability 1/2 right? In our case, we'll want p = 1/3, which will execute with probability 1/3. This allows us to simulate the event in question: the first "Heads" appears whenever np.random.rand() returns a value $< p$. And, if it is $\geq p$, the coin flip turned up "Tails".

To simulate an event, it is important to use a good random number generator. A pseudorandom number generator (PRNG), also known as a deterministic random bit generator (DRBG),is an algorithm for generating a sequence of numbers whose properties approximate the properties of sequences of random numbers. The PRNG-generated sequence is not truly random, because it is completely determined by an initial value, called the PRNG's seed (which may include truly random values). Although sequences that are closer to truly random can be generated using hardware random number generators, pseudorandom number generators are important in practice for their speed in number generation and their reproducibility.

One important rule for excellent RNGs is - do not use system generators, e.g. Python random(), C rand(). Almost all of these generators are badly flawed. Even when they are not, there is no guarantee that they were not flawed in earlier releases of the library (e.g. see note on Python below) or will not be flawed in future releases. ALWAYS USE YOUR OWN RANDOM NUMBER GENERATOR. That way you can ensure your code is portable and you can try different RNGs if you suspect the one you are using is causing a problem.

In this project, in addition to system generators, we'll see two other PRNG generators - KISS [5] and SHR3. The library for these PRNG generators are provided that you can directly use them. If you are interested in the implementation, you can read the supplementary materials in the references.

Description

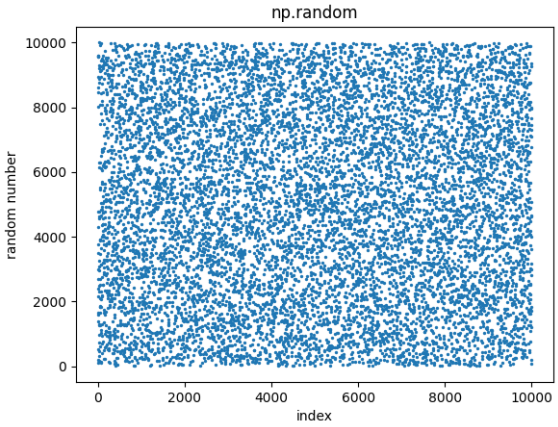
This program will require you to evaluate the distribution of random number generated by three RNGs - system generators(random() for Python and rand() for C), KISS and SHR3.

(a) You'll generate num = 10,000 random number in range (0, num), and make scatter plot for those 3 RNGs respectively. Plot the distribution of the generated number with the x-axis being the index of number, and y-axis being the value of number.

You'll need to implement the following three functions:
distribution_random(), distribution_KISS(), and distribution_SHR3().

In practice, you could set different seeds to get different random number sequences. But in this project, you're required to set the **seed = 0** for system random function. The seeds for KISS/SHR3 are given in the file.

Here's an example result of np.random() in python. Your plot should look something like this:



(b) Then you are required to simulate a ping pong game with one of the above mentioned random number generator - **system random function** with **seed = 5**, and compute the probability you win a ping pong game up to n points, when your probability of winning each point is p (and your friend wins the point with probability $1-p$). Assume you have to win by (at least) 2; for example, if $n = 21$ and the score is 21 - 20, the game isn't over yet.

- Simulate $ntrials = 5,000$ games, and make a single plot with the x-axis being p for different values of p in $\{0, 0.04, 0.08, \cdots, 0.96, 1.0\}$ and the y-axis being the probability of winning the overall game (use your previous function). Plot 3 "curves" in different colors, one for each n in 3,11,21.

- Write AT MOST 2-3 sentences identifying the interesting pattern you notice when n gets larger (regarding the steepness of the curve), and explain why it makes sense.

- Each curve you make for different values of n always (approximately) passes through 3 points. Give the three points $(x_1, y_1), (x_2, y_2), (x_3, y_3)$, and explain why mathematically this happens in AT MOST 2-3 sentences.

You'll need to implement the following functions:
pingpong(), sim_one_game(), plot_output()
Your plot should look something like Fig 1:

(c) Let's learn how to do approximate quantities that are hard to compute exactly! By the end of this,

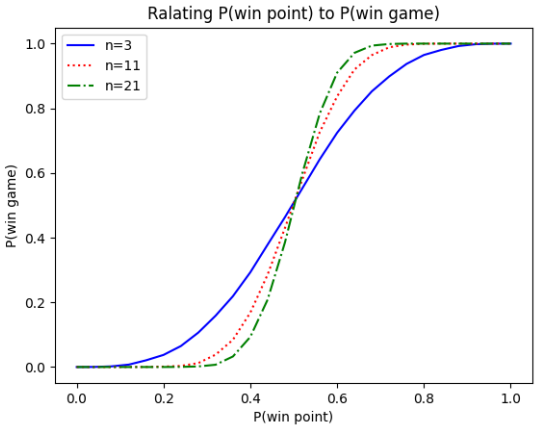


Figure 1: Example plot of (b)

we'll see how long it actually takes to "catch'em all"! You are given a file *data/pokemon.txt* which contains information about several (fictional) Pokemon, such as their encounter rate and catch rate.

The data for this problem follows the following format.

- Col 1: pokemon_id: A unique identifier for the Pokemon.
- Col 2: is_legendary: A 1 if the Pokemon is legendary, and 0 otherwise.
- Col 3: height: The height of the Pokemon in meters.
- Col 4: weight: The weight of the Pokemon in kilograms.
- Col 5: encounter_prob: The probability of encountering this Pokemon in the wild grass. Note the sum of this entire column is 1, since when you make an encounter, exactly one of these Pokemon appears.
- Col 6: catch_prob: Once you have encountered a Pokemon, the probability you catch it. (Ignore any mechanics of the actual game if you've played a Pokemon game in the past.)

Write your code for the following parts in the provided file: pokemon.py or pokemon.cpp
You'll need to implement the following functions:

- part_a: Compute the proportion of Pokemon that are legendary, the average height, the average weight, the average encounter_prob, and average catch_prob.
- part_b: Compute the proportion of Pokemon that are legendary, the average height, the average weight, the average encounter_prob, and average catch_prob OF ONLY those Pokemon STRICTLY heavier than the median weight.
- part_c: Suppose you are walking around the wild grass, and you wonder: how many encounters do you expect to make until you ENCOUNTER each Pokemon (at least) once?
- part_d: Suppose you are walking around the wild grass, and you wonder: how many encounters do you expect to make until you CATCH each Pokemon (at least) once?

- For part_c and part_d, you need to consider that the probability to encounter/catch each pokemon is different.
- When you need to use random function, use system random function with **seed = 1**.

Details for python

- You may define helper functions, but DO NOT MODIFY the parameters or names of the provided functions. Do NOT add any import statements.

- For python, if you want to run it, just type:

  python3 prog1.py

  python3 pokemon.py

  Note that you need to install the package numpy and matplotlib using commands:

  pip3 install numpy

  pip3 install matplotlib

  You also need to install the library for PRNG using commands:

  pip install simplerandom

- Random number generator usage

  - System random function:

  >>> `np.random.seed(seed)` #Initialize the random number generator.

  >>> `np.random.random()` #Return the next random floating point number in the range [0.0, 1.0).

  >>> `random.randint(a, b)` #Return a random integer N in the range [a,b].

  >>> `np.random.choice(a, p)` #Return a random element from the non-empty sequence a, with p being the probabilities associated with each entry in a.

  - simplerandom.iterators:

  >>> `import simplerandom.iterators as sri`

  #Initialize the random number generator KISS/SHR3. **Do not** change the seeds in your program.

  >>> `rng_kiss = sri.KISS(123958, 34987243, 3495825239, 2398172431)`

  >>> `rng_shr3 = sri.SHR3(3360276411)`

  #Return the next random unsigned 32-bit integers. You can cast it to the range you need by %num.

  >>> `next(rng_kiss)`

  >>> `next(rng_shr3)`

Details for C/C++

- You may define helper functions, but DO NOT MODIFY the parameters or names of the provided functions.

- For C/C++, if you want to run it:

  make

  ./prog1

  ./pokemon

  Makefile and rng generator (marsaglia-rng.cpp) are provided. You don't need to change them.

- Random number generator usage

  - System random function:

  >>> `srand(seed)` #Initialize the random number generator.

  >>> `rand()` #Return the next random integer number in the range [0, RAND_MAX).

  >>> `rand()/double(RAND_MAX)` #Return the next random float number in the range [0.0, 1.0).

>>> `rand()%N` #Return a random integer in the range [0,N).

Since C/C++ does not have a function like random.choice in python, you may need to use these function to select a data from the data array.

- Other random function:

marsaglia-rng.cpp includes KISS and SHR3 rng generators. marsaglia-rng.cpp has already set the seeds, you don't need to change them. Example usage can be found in prog1.cpp.

They generate unsigned 32-bit integers. You'll need to cast them to the range you need by %num.

- Random pick with probability. (c)

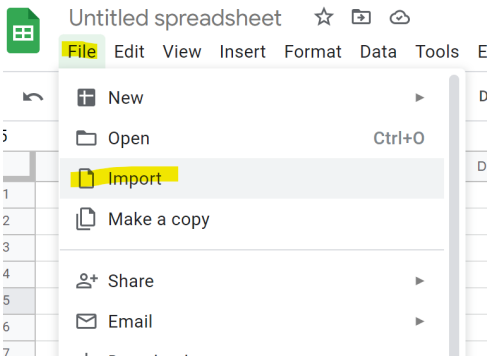
  Note that in pokemon game, the probability to encounter/catch each pokemon is different, which means you'll randomly select one pokemon with probability.

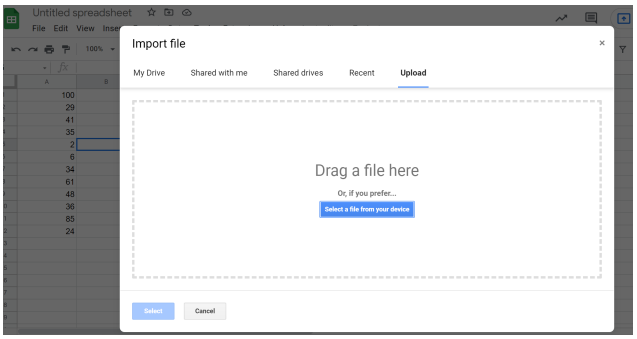
  Here's a simple example of how to do this:

  Let's assume we have 4 pokemons with encounter probability: $\{A : 0.5, B : 0.2, C : 0.2, D : 0.1\}$. First, do a cumulative probability distribution(cpd) like this: $\{A : 0.5, B : 0.7, C : 0.9, D : 1\}$. If you generate a random float number $= 0.75$, compare it with the cpd: $0.7 < 0.75 < 0.9$. That means you select pokemon C. If you generate a random float number $= 0.95$, compare it with the cpd: $0.9 < 0.95 < 1$. That means you select pokemon D.

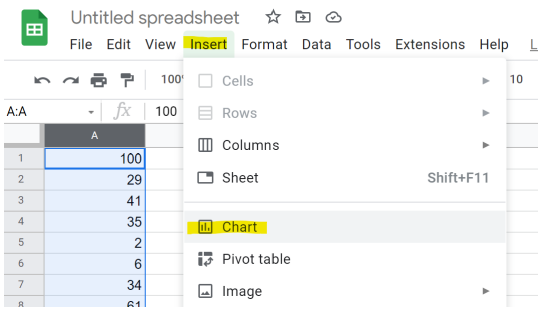
- Instruction of how to plot in Google Sheet.

  -step 1: import the data from .txt file.





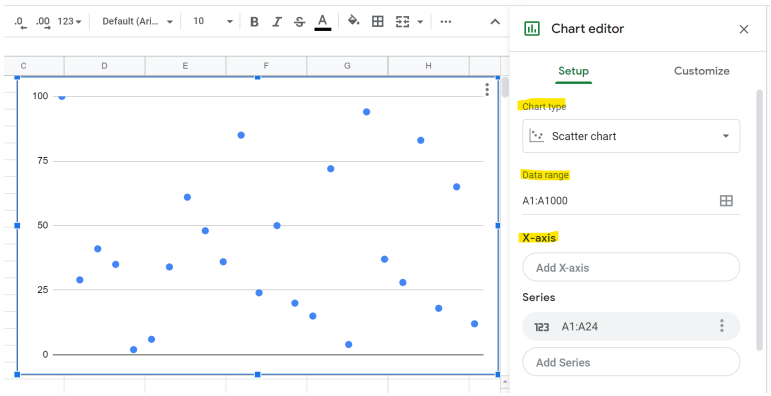
  -step 2: Scatter plot of a single column of data. You need to first select the target column, and then insert chart.



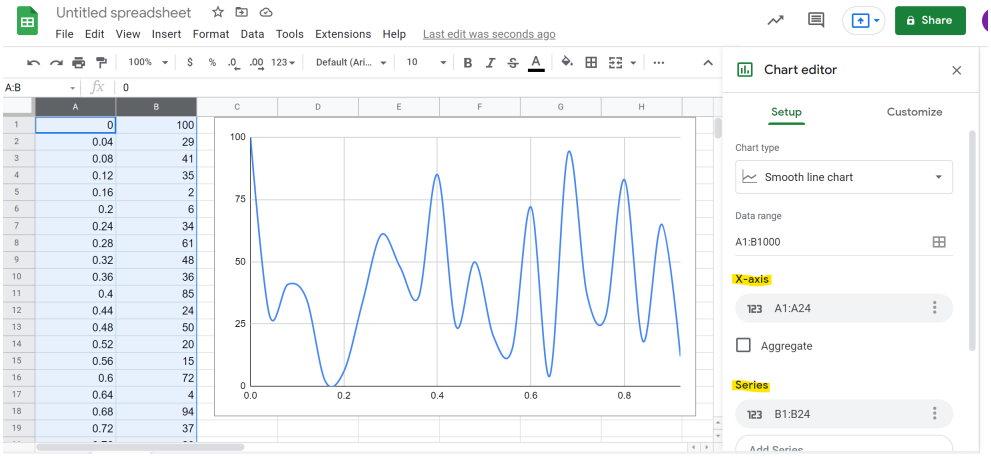
  You can further change the Chart type, data range, X-axis here in **Setup**. You can also change color and add title in **Customize**.

  -step 3: Make a plot with given x-axis and y-axis data (in ping pong game simulation).

  You first need to import the data of probability of winning the overall game (computed using your program). Next, you'll create another column for different values of p in {0,0.04,0.08,...,0.96,1}. Then, select these two column and insert chart.



You'll need to set the **Chart type** to be "Smooth line chart", **X-axis** to be the column of p value, and **Series** to be the column of probability data.



3 What to submit

You're required to submit a zip file with your code and a program report (pdf). Name your zip file as Program1_Yourname.

The program report should follow the requirement of program description, and include:

- 3 figures for (a)
- 1 figure for (b)
- screenshot of result for (c)

References

[1] `https://en.wikipedia.org/wiki/Pseudorandom_number_generator`

[2] `https://en.wikipedia.org/wiki/List_of_random_number_generators`

[3] `https://pypi.org/project/simplerandom/`

[4] `https://github.com/cmcqueen/simplerandom`

[5] Rose, Gregory G. "KISS: A bit too simple." Cryptography and Communications 10.1 (2018): 123-137.