PS 7 - Monte Carlo Integration & Simulations

Math087 - George McNinch

due 2025-03-30

1. Consider the function $f(x) = \dfrac{1}{x}$ defined on the interval $I = \left[\dfrac{1}{2}, 1\right]$. Note that f is a decreasing function on the interval, and in particular

$$\frac{1}{x} \le 4 = f(1/2).$$

for each $x \in I$. According to the Fundamental Theorem of Calculus, one knows the following:

$$\int_{1/2}^{1} \frac{1}{x} dx = \ln(x)\Big|_{1/2}^{1} = -\ln(1/2) = \ln(2).$$

a. If X and Y are random variables uniformly distributed respectively on the intervals $[1/2, 1]$ and $[0, 4]$, explain why

$$P\left(\frac{1}{2} \le X \le 1, 0 \le Y \le \frac{1}{X}\right) = \frac{\ln(2)}{2}.$$

b. Write a `python` function which takes as argument a whole number `n` and estimates $\ln(2)$ by generating `n` random points `(x,y)` in the region $[1/2, 1] \times [0, 4]$, counting the number `m` of those points `(x,y)` for which y is *below* the graph $y = \dfrac{1}{x}$, and using the ratio `m/n` to produce an estimate of $\ln(2)$.

Include the text of your function in your problem submission, and include a brief explanation of how it works.

Compare your result to `npumpy.log(2)` (note that `numpy.log` is the natural logarithm). How large must `n` be in order that your estimate matches `numpy.log(2)` to 2 decimal places?

Hints/suggestions:
You should execute the following code to create a random number generator in `python`:

```python
from numpy.random import default_rng
rng = default_rng()
```

Now `rng.random()` will return a random number in the interval $[0, 1]$ (*try it!*).
You should write a `python` function

```python
def estimate_log_two(n):
    # ...
    # ...
```

that takes as argument a variable `n` and return an estimate of $\ln(2)$; it should proceed as follows:

- generate a list `xl` of length `n` of random numbers between `0.5` and `1`.
- generate a list `yl` of length `n` of random numbers between `0` and `4`.
- count the number `m` of pairs `(x,y)` from the list `zip(xl,yl)` for which `y < 1.0/x`.

Then `m/n` is an estimate for $\ln(2)/2$ (why?).

Assignment continues on next page

2 *Jane's Fish Tank Emporium* (*JFTE*) revisited.

In the course notebook, we discussed the operation of *JFTE* by considering the question: what is the optimal ordering strategy for fish tanks?

- Is it *on-demand* ordering (where an order is made after a sale)?
- Or is it better to have *standing orders* (where an order is made regularly – say, on a particular day of the week)?

In the notebook, we studied the case for which the probability of a customer arriving at the store on any particular day was `1/7`. Let's now consider the case where the probability of the arrival of a customer to the store depends on the day of the week (DOW), as follows:

Day	Sun	Mon	Tue	Wed	Thur	Fri	Sat
DOW	0	1	2	3	4	5	6
Prob	0.16	0.08	0.04	0.08	0.12	0.25	0.27

Here the `DOW` ("day of week") row just indicates that we view `Mon` as day 1 of a week, `Tue` as day 2, etc.

In the notebook, we constructed a `python` class `JFTE` to keep track of our simulations. The *constructor* of the class `JFTE` (i.e. its member function `__init__`) creates the `customer` instance variable; to do this, it invokes the function

```
def customer(prob=1./7):
  return rng.choice([1,0],p=[prob,1-prob])
```

Make an alternative to this function `customer` by creating a new function `customer_alt` taking an integer argument `m` which returns `1` with probability as indicated in the above table (for the `DOW` corresponding to `m`) and otherwise returns `0`.

Recall that we may use the *modulus* function `np.mod(m,7)` to compute the `DOW` of `m`. For example,. the condition `np.mod(m,7) == 3` is `True` if and only if `m` is a `Wed`.

Now edit the code for the `JFTE` class from the notebook, arranging the `__init__` function of your new class to instead uses your *new* function `customer_alt` to produce the instance variable `customers`. You can assume that the days for your simulations always begin on a Sunday!

The notebook implemented `strategy` functions `stand_order` and `order_on_demand` which take as arguments an instance of the class `JFTE`.

You may now apply these `strategy` functions to an instance of the `JFTE` class constructed using your alternative customer-arrival function.

Run the simulation 10 times with both strategy functions, as was done in the notebook. Discuss similarities/differences between the results obtained in the notebook.

In addition to discussion, be sure to include the code for your function `customer_alt` and a summary of the results of your 10 simulations for each strategy.

Assignment continues on next page

3. More *JFTE*

   In this problem, we again consider the *JFTE* enterprise; let's return to the "constant" customer arrival probability described in the notebook.

   For each strategy `stand_order` and `order_on_demand` , compute the average `storage_days` and the average `sales` for 10 simulations. (So you'll have averages for `stand_order` and averages for `order_on_demand`).

   If the storage costs are $1 per tank per day, use your averages to estimate what the profit per tank needs to be for `JFTE` to have a positive `net_profit` for each of these strategies.