PSTAT160A F21 Python HW 2

October 14, 2021

0.1 PSTAT 160A Fall 2021 Python Homework 2

Due date: Friday, October 15, 11:59 p.m. via GauchoSpace

Instructions: Please upload your PDF or HTML file on Gradescope with filename "PythonHW2_YOURPERMNUMBER".

0.2 Problem 1 (10 pts total)

Background: A stochastic model for a car insurance company's total cost of damages from traffic accidents goes back to the work by Van der Lann and Louter, "A statistical model for the costs of passenger car traffic accidents", Journal of the Royal Statistical Society (1986).

For every $k = 1, 2, 3 \ldots$ we denote by the random variable X_k the US dollar amount of a damage from a policy holder's traffic accident which will occur during the year 2020.

We assume that X_1, X_2,... is an i.i.d. sequence of exponential distributed random variables with an average claim size of \$1,500 USD.

The (random) total number of accidents N in 2020 is expected to be Poisson distributed with 25 claims on average.

It is assumed that the number of accidents is independent of the US dollar amount of damages for each accident. That is, the random variable N is independent of the random variables X_1, X_2,...

The total costs for the insurance company by the end of 2020 will thus be given by the **random sum** S_N defined as

$$S_N = X_1 + X_2 + \cdots + X_N = \sum_{k=1}^{N} X_k.$$

Note again that the total number N of accidents is random

The goal of the current exercise is to approximate the expected total costs

$$\mathbb{E}[S_N]$$

for the insurance company in 2020 via simulations.

As usual, we start with loading some packages:

[1]:
```
import numpy as np
```

0.2.1 Step 1: (5 Points)

Write a function called randomSum(...) which simulates the random variable S_N.

Input: * averageClaimSize: Average USD amount per claim * averageNumberOfClaims: Average number of claims/accidents in 2020

Output: * sampleRandomSum: A single scalar being one sample from the random variable S_N

Hint: Use build-in functions from the NumPy-package in your code in order to sample from a Poisson distribution and from an exponential distribution!

[2]:
```
def randomSum(averageClaimSize, averageNumberOfClaims):

    N = np.random.poisson(averageNumberOfClaims, 1)
    X = np.random.exponential(averageClaimSize,  N)
    sampleRandomSum =  sum(X)

    return sampleRandomSum
```

[3]:
```
## TEST YOUR FUNCTION HERE
randomSum(1500,25)
```

[3]:
```
22011.696488825983
```

0.2.2 Step 2: (3 Points)

Write a simulator function called simulator() which uses the function randomSum() from Step 1 to simulate $M \in \mathbb{N}$ samples from the random variable S_N.

Input: * averageClaimSize: Average USD amount per claim * averageNumberOfClaims: Average number of claims/accidents in 2020 * M: Number of Simulations

Output: * samples: An array of length M with samples from the random variable S_N.

[4]:
```
def simulator(averageClaimSize, averageNumberOfClaims, M):

    samples = list()
    for i in range(M):
        samples.append(randomSum(averageClaimSize, averageNumberOfClaims))
    return samples
```

[5]:
```
## TEST YOUR FUNCTION HERE
simulator(1500,25,10)
```

[5]:
```
[30797.667599947203,
 29375.039613715995,
 29087.21552614803,
 36682.22534288014,
 27938.61363348124,
 52011.128883882186,
```

```
38524.08699156176,
45314.4783906399,
72779.38887472807,
36988.625473571]
```

0.2.3 Step 3: (2 Points)

As we have shown in class, it holds via **Wald's Identity** that the expectation of the random sum S_N is given by the formula

$$\mathbb{E}[S_N] = \mathbb{E}[N] \cdot \mathbb{E}[X_1] = 25 \cdot \$1,500 = \$37,500. \tag{1}$$

Check via the empirical mean that

$$\frac{1}{M} \sum_{m=1}^{M} s_N^{(m)} \approx \mathbb{E}[S_N] = \$37,500$$

where $s_N^{(1)}, s_N^{(2)}, \ldots, s_N^{(M)}$ denote M independent realizations (samples) from the random variable S_N.

Use $M = 10, 100, 1000, 10000, 50000$ simulations.

That is, write a function MCsimulation(...) which uses the function simulator(...) from Step 2 to compute the empirical mean.

Input: * averageClaimSize: Average USD amount per claim * averageNumberOfClaims: Average number of claims/accidents in 2020 * M: Number of Simulations

Output: * empricialMean: A real number in $\mathbb{R}_+$.

[6]:
```
def MCsimulation(averageClaimSize, averageNumberOfClaims, M): # 2 points

    samples = list()
    for i in range(M):
        samples.append(randomSum(averageClaimSize, averageNumberOfClaims))

    empricialMean = sum(samples)/M


    return empricialMean
```

[7]:
```
## TEST YOUR FUNCTION HERE
MCsimulation(1500, 25, 1)
```

[7]:
```
35669.16711415436
```

[8]:
```
## Compute the absolute error
print(np.absolute(MCsimulation(1500, 25, 10)-37500))
print(np.absolute(MCsimulation(1500, 25, 100)-37500))
```

```
print(np.absolute(MCsimulation(1500, 25, 1000)-37500))
print(np.absolute(MCsimulation(1500, 25, 10000)-37500))
print(np.absolute(MCsimulation(1500, 25, 50000)-37500))
```

```
8135.835205103911
146.22143726561626
125.91138632463844
214.81236153500504
2.8655910863744793
```

0.3 Problem 2 (5 Points)

A health insurance will pay for a medical expense subject to a USD 100 deductible. Assume that the amount of the expense is **Gamma** distributed with scale parameter 100 and shape parameter 2 (the mean is 100*2 dollars). This can be simulated using np.random.gamma(shape, scale, n)

Compute the empirical *mean* and empirical *standard deviation* of the payout by the insurance company by using 100,000 samples.

[9]:
```
# WRITE YOUR OWN CODE HERE! FEEL FREE TO INSERT MORE CELLS!
# ADD SOME COMMENTS TO YOUR CODE!

n= 100000 # smaple number

expense = np.random.gamma(shape = 2,scale=100, size = n)

expense_mean = sum(expense)/n  # apply empirical mean

expense_std = np.sqrt(sum((expense - expense_mean)**2)/n) # apply empirical␣
↪standard deviation


print("expense_mean:", expense_mean)
print("expense_std:", expense_std)
```

```
expense_mean: 199.2422617325569
expense_std: 140.26971384540863
```

0.4 Problem 3 (5 Points)

Since the beginning of fall quarter, Adam goes to Woodstock's Pizza every day, orders a slice of pizza, and picks a topping - pepper, mushrooms, pineapple, or onions - uniformly at random.

1. Implement a simulator which uniformly samples from one topping:

[10]:
```
# WRITE YOUR OWN CODE HERE! FEEL FREE TO INSERT MORE CELLS!
# ADD SOME COMMENTS TO YOUR CODE!
toppings = ['pepper','mushroom','pineapple','onion']
def topping_simulation():
    topping = np.random.choice(toppings)
```

```
    return(topping)
```

[11]:
```
topping_simulation()
```

[11]:
```
'mushroom'
```

[12]:
```
toppings[3]
```

[12]:
```
'onion'
```

2. On the day that Adam first picks pineapple, find the empricial *mean* and empirical *standard deviation* of the number of prior days in which he picked mushroom by running 100,000 simulations. [As you might realize, this is very similar to the question about rolling 5's before the first '6' appears that we did in class – now we solve it/verify the answer by simulation]

[13]:
```
# WRITE YOUR OWN CODE HERE! FEEL FREE TO INSERT MORE CELLS!
# ADD SOME COMMENTS TO YOUR CODE!

sum_1 = 0
result = []
for i in range(0,100000):  # 100000 simulation
    sum_1 = 0
    topping_today = topping_simulation()  # get the topping for current day
    while (topping_today != "pineapple"):   # run the simulation if get␣
↪pineapple
        if (topping_today == "mushroom"):    # if adam choose mushroom then the␣
↪counter+1
            sum_1 +=1
        topping_today = topping_simulation()
    result.append(sum_1)

result = np.array(result)
mean = np.mean(result)
std =  np.std(result)
print(mean)
print(std)
```

```
0.99762
1.4059567331891833
```