IC252:Data Science II

Lab 8

Submitted by ADARSH SANTORIA (B21176)

In this assignment, you will verify the phenomena of the "weak law of large numbers (WLLN)" and the "central limit theorem (CLT)" numerically.

1. Generate a random data set $x_1, \ldots, x_m$ of size m for

   (a) exponential r.v. Exp(1),

   (b) uniform r.v. Unif(1, 2),

   (c) Bernoulli r.v. Bern(.2).

   Plot the sample mean $(x_1 + \ldots + x_m)/m$ for $m = 10, 100, 500, 1000, 5000, 10000, 50000$ for (a), (b), (c). This way, verify the WLLN.

2. Generate a random data set of size 1000 for each of independent and identically distributed

   (a) n exponential r.v.s Exp(1),

   (b) n uniform r.v.s Unif(1, 2),

   (c) n Bernoulli r.v.s Bern(.2).

   numerically compute and plot the distribution of the sample mean $(X_1 + \ldots + X_n)/n$ and corresponding "normal" approximation for $n = 1, 2, 4, 8, 16, 32$ for (a), (b), (c). This way, verify the CLT.

Notes:

1. While generating data sets, consider the number (to be with the precision of) decimal place of 1. That is, if the random number generated is 1.4789 then consider it as 1.4.

   Optional: To get better, smoother results, (1) in Problems 1 and 2, consider the number (to be with the precision of) decimal place of 2. Also, in Problem 2, you may consider the sample size of 10000.

2. You may use in-built functions/library to generate samples from desired underlying distributions.

3. In Problem 1, the x-axis of the plot should be the sample size m and y-axis should be the sample mean.

4. In problem 2, numerically compute the distribution of X_i's from a data set of size 1000 and then plot. (In reality, the distribution of X_i is exactly exponential, uniform or Bernoulli depending upon the subproblem (a), (b) or (c). But in this problem, you will be working with numerical data and finding distribution from this numerical data.)

Ouputs on terminal are below:-

```
Pi = 3.4 for 100 simulations
Pi = 3.068 for 1000 simulations
Pi = 3.135 for 10000 simulations
Integral = 1.594 for 100 simulations
Integral = 1.562 for 1000 simulations
Integral = 1.571 for 10000 simulations
e = 2.72108843537414 96 for 100 simulations
e = 2.7292576419213974 for 1000 simulations
e = 2.70197243988111 34 for 10000 simulations
```

Monte Carlo Method is used to solve these by running different simulations.

Commented Codes are below:-

```
import numpy as np              #importing numpy module
'''a'''
n=[100,1000,10000]              #makinging a list of given sample sizes
for i in n:                     #running a loop for given sample sizes
   c=0                          #making a variables and add 1 if favourable
   for j in range(i):           #running a loop for given sample sizes
      x=np.random.uniform(-1,1) #making a random 2-d co-ordinate (x,y) where
      y=np.random.uniform(-1,1) # -1>=x>=1 & -1>=x>=1
      o=(x**2 +y**2)**(0.5)     # o is distance from origin
      if o<=1:                  # if o<=1, it is a favourable case
         c+=1
   print('Pi =',round(4*c/i,3),'for',i,'simulations')
#Pi is approximated using Monte Carlo
'''b'''
for i in n:                     #running a loop for given sample sizes
   f=0
#making a variable and adding value of different points on function with respect to x
   for j in range(i):           #running a loop for given sample sizes
      x=np.random.uniform(0,1)  #making a variable x to store a random number between 0 & 1
      f+=2/(1+x**2)
```

```
   print('Integral =',round(f/i,3),'for',i,'simulations')
#integral of function is approximated using Monte Carlo
'''c'''
for i in n:                  #running a loop for given sample sizes
   c=0                       #making a variable c
   l=np.arange(1,i+1)        #making a list containing all values from 1 to given sample sizes
   for j in range(10000):    #running 10000 simulations
      np.random.shuffle(l)   #randomly shuffling list l
      for k in range(i):     #running for loop in l
         if l[k]==k+1:
            c+=1             #counting number of favourable cases in c
            break
   print('e =',10000/(10000-c),'for',i,'simulations')
#e is approximated using Monte Carlo
```