컴퓨터응용확률 과제3

12161570 박성연

구현 환경 : Visual Code 1.29.1 , GNU Plot Version 5.2 patchlevel 4, anaconda prompt

구현 언어 : Python

1. [30pts] Write codes to simulate tossing a coin to see how the law of large numbers works. You may use any random number generator (using any programming language) as long as it follows a uniform distribution.

a. (10pts) Assuming a fair coin where P(HEAD)=P(TAIL)=0.5, plot a graph showing the proportions of heads with respect to the number of coin tosses ranged from 1 to 100. i.e., X-axis(1~100): the number of coin tosses Y-axis: the proportions of heads

동전을 던지는 임의의 값을 뽑기 위해서 random을 import 한다. `import random`

```
def head_P() :
    front = 0
    for i in range(0,1000):
        result = random.randrange(0,2)
        if(result == 1) :
            front += 1
    return (front / 1000)
```

동전을 던져서 head가 나온 확률을 구하는 함수 head_P를 정의한다.

이 때, front는 머리의 개수를 count하는 변수이다. for문을 총 1000번 돌려서 동전을 1000번 던지는 시행을 한다. Randomrange함수를 이용해서 0과 1 중에서 random하게 골라 result에 저장한다. Result가 1이면 head라고 생각하고 1의 개수를 센다. Head의 총 개수인 front를 1000으로 나눠서 확률을 계산한다.

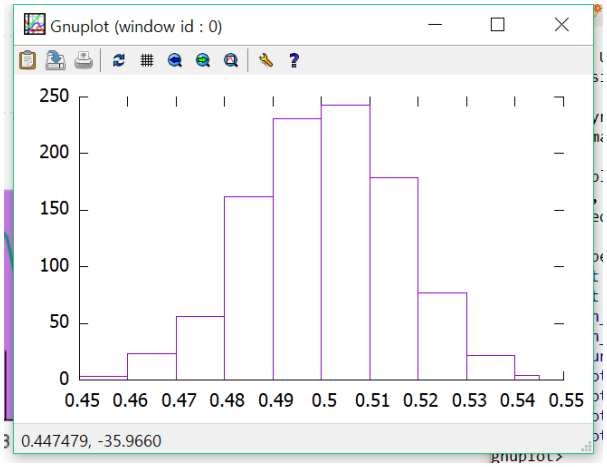
```
f=open("output_a.txt","w")
for i in range(1,1001):
    f.write(str(head_P())+"\n")
f.close()
```

'ouput_a.txt'를 만들고 쓰기 형식으로 파일을 연다. 동전을 1000번 던져서 확률을 내는 시행을 100번 돌려서 그 확률을 'output_a.txt'에 저장한다. 이 때 txt에 쓰려면 string 이나 list 여야 하기 때문에 string으로 변환을 해준다.

```
gnuplot> bin_width = 0.01
gnuplot> bin_number(x) = floor(x/bin_width)
gnuplot> rounded(x) = bin_width * (bin_number(x)+0.5)

gnuplot> plot 'C:\Users\hepbu\homework3\output_a.txt'using (rounded($1)):(1) smooth frequency with boxes
gnuplot> _
```

GNU Plot에 위와 같이 입력을 해서 확률의 개수 만큼 세고, 출력을 시킨다.



b. (10pts) Consider an unfair coin with P(HEAD)=0.6 and P(TAIL)=0.4. Plot a graph similar to the problem a.

a번과 같이 random을 import 해주고, 동전을 던져서 head가 나오는 확률을 계산하는 함수를 구현한다.

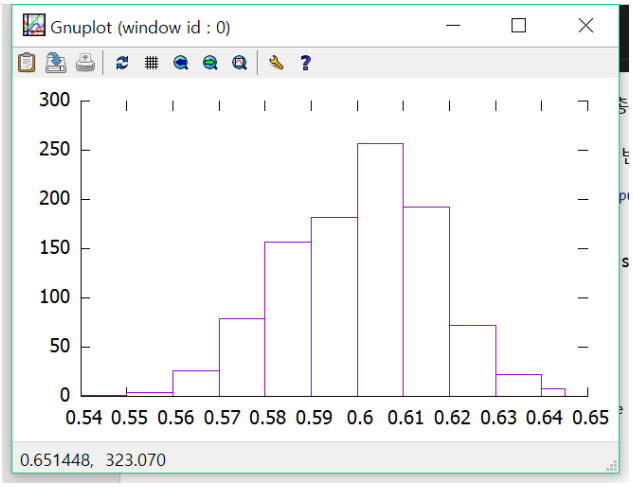
```
def head_P() :
    front = 0
    for i in range(0,1000):
        result = random.randrange(0,10)
        if(result < 6) :
            front += 1
    return (front / 1000)
```

randomrange함수를 이용해서 0부터 9까지의 범위 내에서 임의로 하나의 값을 result에 저장하게 한다. 이 때, head가 나올 확률이 0.6이기 때문에 result 값이 5 이하라면 head라고 생각하고 head의 개수를 1개 올려준다. a번과 같이 이 시행을 1000번 진행하고 총 head의 개수를 1000으로 나눠서 반환한다.

이 후 a번과 마찬가지로 이 함수를 1000번 시행하고, 그 확률들을 'output_b.txt'에 출력한다.

```
gnuplot> plot 'C:\Users\hepbu\homework3\output_b.txt'using (rounded($1)):(1) smooth frequency with boxes
gnuplot> _
```

a번과 마찬가지로 GNU Plot을 이용해서 setting을 하고 히스토그램을 출력해준다.



c. (10pts) Discuss if the graph converges to the value of P(HEAD) when the number of coin tosses are large enough.

위의 a와 b는 1000번 시행했을 때의 확률에 대한 분포를 구한 것이다. 시행 횟수가 커지면 커질수록 이론적으로 일어날 확률과의 오차가 줄어들 것이다. 따라서 정말 많이 해본다면 이론적인 확률, 즉 P(HEAD)와의 오차가 0으로 수렴하게 될 것이다. 이는 강한 큰 수의 법칙이라고도 한다.

2. [70pts] Write codes to generate a binomial random variable (Binomial ~ (n,p)) by extending the code from the problem 1.

a. (20pts) Setting p=0.5 and n=10, generate 1024 samples from the code and plot a histogram showing the frequencies of each k value. Compare your result with Figure 4.5 (Ross).

$$P(X = k) = \binom{N}{k} p^k (1-p)^{N-k}$$

이항분포를 구현하기 위해서는 확률(p=0.5), 시행 횟수(n=10)이 필요하다. 위의 식을 그대로 구현했을 때, 아래의 코드가 된다.

```
def binomial(n,p):
    q = 1- p
    for i in range(n+1) :
        dist = np.array(ncr(n,i)*(p**i)*(q**(n-i)))
    return dist
```

여기서 ncr은 조합을 구현한 함수로 다음과 같다,

```
def ncr(n, r) :
    if r in (0, n):
        return 1
    return ncr(n-1, r) + ncr(n-1,r-1)
```

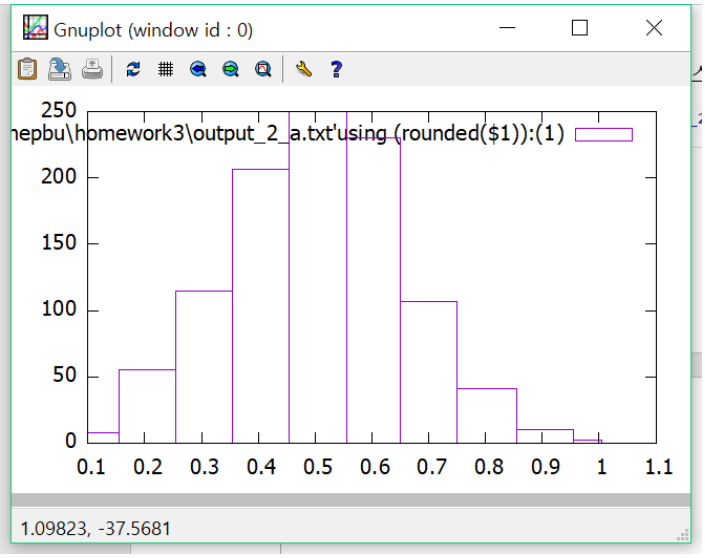
하지만 python math에서 제공하는 binomial random variable이 있다. 이는 parameter로 random한 값, 그 값이 나올 확률, 시행횟수를 받는다.

이 함수를 이용해서 구현해봤다.

```
f=open("output_2_a.txt","w")
n=10
p=0.5
for i in range(0,1025) :
    s = sum(np.random.binomial(1,p,n)==1)/n
    f.write(str(s)+"\n")
f.close()
```

먼저 n과 p를 정의하고 for문을 1024번 돌렸다. 1이 random하게 나올 확률은 0.5로 이를 n번 시행했을 때 나오는 1의 개수를 세고, 이 때 head를 1로 둬서 나온 값을 모두 더한 것이 나온 횟수를 세는 것과 같다. 그 값을 n으로 나누면 평균 값이 나온다. 이를 출력해서 GNU plot에서 문제 1번과 같이 평균들의 frequency에 대한 히스토그램을 그렸다.

```
gnuplot> plot 'C:\Users\hepbu\homework3\output_2_a.txt'using (rounded($1)):(1) smooth frequency with boxes
gnuplot> _
```



b. (20pts) Based on the samples generated in the problem a, estimate using ML estimate. Compute the probability P(=0.5)

```
f = open("output_2_a.txt","r")
lines = f.readlines()
count = 0
sum = 0
for i in lines :
    line = i.split("\n")
    count += 1
    sum += float(line[0])

print(sum/count)
```

Maximum Likelihood Estimate를 이용하면 위의 사건이 일어날 확률은 0.5이고, n번 시행했을 때 k번이 일어났다면 0.5 = k/n이라는 것이다. 이를 구현하기 위해서, 문제 a의 output파일을 읽고, 한 줄 씩 읽으면서, 한 줄에 하나씩 count해서 전체를 돌았을 때 몇 번 시행을 했는지 세도록 하고, 나온 확률들을 전부 더해준다.

그 후에 합들을 전체 시행 횟수로 나눠주면 MLE의 k/n을 구할 수 있다. 시행 결과는 아래와 같다.

```
(base) C:₩Users₩hepbu₩homework3>python homework3_2_b.py
0.49834146341463387
```

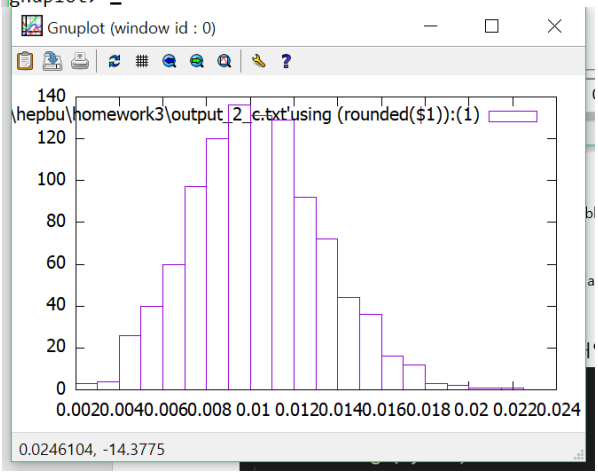
c. (10pts) Now, setting p=0.01 and n=1000, generate as many samples you wish and plot a histogram as the similar way to the problem a.

문제 a에서 n에는 1000을, p에는 0.01을 대입했다.

```
f=open("output_2_c.txt","w")
n=1000
p=0.01
for i in range(0,1025) :
    s = sum(np.random.binomial(1,p,n)==1)/n
    f.write(str(s)+"\n")
f.close()
```

결과를 받아서 GNU Plot에서 bin_width를 0.01로 설정하고 히스토그램을 그렸다.

```
gnuplot> plot 'C:\Users\hepbu\homework3\output_2_c.txt'using (rounded($1)):(1) smooth frequency with boxes
gnuplot> _
```



d. (20pts) Find and plot a pmf from the problem c, and compare your results with a poisson distribution with parameter

.

```
f = open("output_2_c.txt","r")
lines = f.readlines()
p = np.zeros(26)
count = 0
```

문제 c의 output에서 pmf를 구해보았다. 먼저 ouput 텍스트 파일을 읽기 형식으로 불러오고, 전체를 lines로 받았다. 문제 c의 GNU Plot에서 의 결과를 보니, 대략 0.000부터 0.025로 분포되어 있는 것 같아 보여서 배열을 25개 선언하고 0으로 초기화 했다. 또 전체 데이터 개수를 세어 줄 count 변수를 0으로 초기화 했다.

```
for i in lines :
    line = i.split('\n')
    if float(line[0])==0.000 : p[0] += 1
    elif float(line[0])==0.001 : p[1] += 1
    elif float(line[0])==0.002 : p[2] += 1
    elif float(line[0])==0.003 : p[3] += 1
    elif float(line[0])==0.025 : p[25] += 1
    count += 1
```

한 줄 씩 읽으면서 그 줄의 숫자가 0.000이면 0번째 배열에 1을 더하고 0.001이면 1번째 배열에 1을 더하는 방법을 25개의 배열에 모두 해주면서 각 확률마다 일어난 횟수와 for문 마지막에 count를 1씩 올려서 전체 경우의 수를 세었다.

```
for i in range(0,26) :
    print("P(X = 0.0"+'%02d'%i+") ="+str(p[i]/count)+"\t")
```

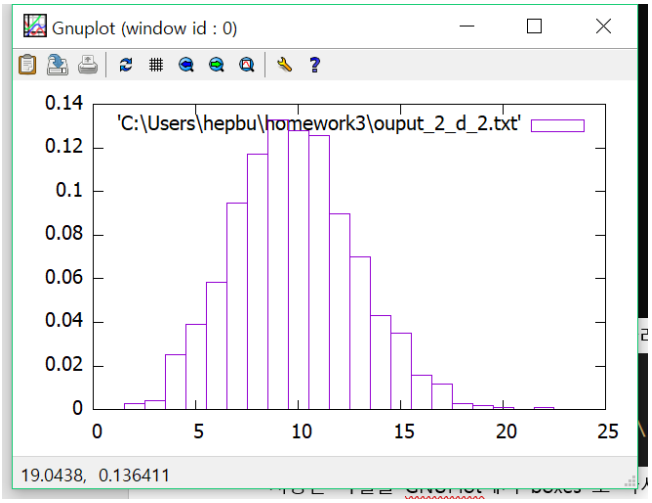
출력은 0부터 25까지 for문을 돌려서, 배열에 있는 값을 전체 시행 횟수인 count로 나눠주고 했다. 출력 결과는 아래와 같다.

```
(base) C:₩Users₩hepbu₩homework3>python homework3_2_d.py
P(X = 0.000) =0.0
P(X = 0.001) =0.0
P(X = 0.002) =0.002926829268292683
P(X = 0.003) =0.003902439024390244
P(X = 0.004) =0.025365853658536587
P(X = 0.005) =0.03902439024390244
P(X = 0.006) =0.05853658536585366
P(X = 0.007) =0.09463414634146342
P(X = 0.008) =0.11707317073170732
P(X = 0.009) =0.1326829268292683
P(X = 0.010) =0.1278048780487805
P(X = 0.011) =0.12585365853658537
P(X = 0.012) =0.0897560975609756
P(X = 0.013) =0.0702439024390244
P(X = 0.014) =0.042926829268292686
P(X = 0.015) =0.035121951219512122
P(X = 0.016) =0.015609756097560976
P(X = 0.017) =0.011707317073170732
P(X = 0.018) =0.002926829268292683
P(X = 0.019) =0.001951219512195122
P(X = 0.020) =0.000975609756097561
P(X = 0.021) =0.0
P(X = 0.022) =0.000975609756097561
P(X = 0.023) =0.0
P(X = 0.024) =0.0
P(X = 0.025) =0.0
```

이를 새로운 텍스트 파일을 만들어서 아래와 같이 데이터를 출력한다.

```
f=open("ouput_2_d_2.txt","w")
for i in range(0,26) :
    f.write(str(p[i]/count)+"\n")
f.close()
```

저장한 파일을 GNUPlot에서 boxes 로 다시 돌려보면 아래와 같은 그림이 나온다.



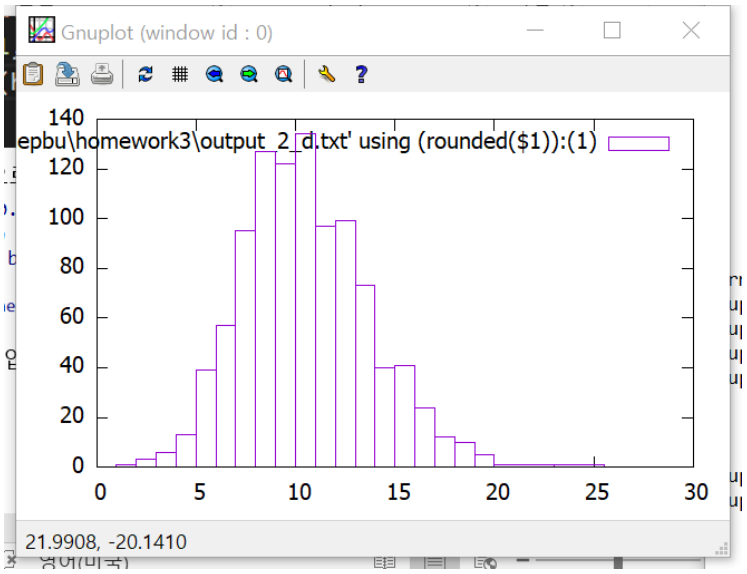
Python math 에서 기본적으로 제공하는 poisson 함수를 사용해서 출력해보았다.

```
f=open("output_2_d.txt","w")
a = np.random.poisson(10,1000)
for i in range(0,1000) :
    f.write(str(a[i])+"\n")
f.close()
```

Poisson 함수에는 원래 식에서도 사용되는 람다 값과 몇 번 시행하는지 n을 parameter로 받는다.

문제c에서 n은 총 1000번 이었기 때문에 n은 1000이고 람다는 문제에서 주어진 10을 입력했다.

출력한 데이터를 GNU Plot을 이용해서 문제 1.a. 와 마찬가지로 빈도수에 대해 히스토그램을 그렸더니 아래와 같은 결과가 나왔다.



C번 문제에 대해 pmf를 구하고 그린 그래프와 람다 값을 10으로 갖는 포아송 분포의 그래프를 비교해 봤을 때, 중심이 될 만한 위치와 값들 빈도수가 정말 비슷하다.