A4 Python codes

Jaemin Park

2021-01-13

차 례

Exercises

p.11 Implementation - basic

전반적인 코드 내용이 동일합니다.
난수생성시 np.random.rand함수와 np.random.uniform 함수로 나뉘었습니다.

R code

```
N <- 10^3
x <- runif(N)*2-1 # runif() generates U(0,1)
y <- runif(N)*2-1
t <- sqrt(x^2+y^2)
head(cbind(x,y,t)) # always display and check!
pi_hat <- 4*sum(t<=1)/N
pi_hat
```

Python code
1> uniform 함수로 난수생성 / pandas로 bind

Randon.uniform 으로 난수생성방법이 몇가지 있었습니다.
1. x = np.random.uniform(0,1,size = N)*2-1*
*2. x = np.random.uniform(low=0,high=1,size=N)*2-1
3. x = np.random.uniform(-1,1,MC_N)

uniform으로 난수생성시 numpy로 bind가 안되는듯 합니다. 제출물 모두 uniform-pandas로, rand는 numpy로 bind하였습니다.

```
import numpy as np
import pandas as pd
MC_N = 1000
x = np.random.uniform(-1,1,MC_N)
y = np.random.uniform(-1,1,MC_N)
t = np.sqrt(x**2+y**2)

bind=pd.DataFrame({'x':x,'y':y,'t':t})
#print(bind.head(5))
```

2> rand 함수로 난수생성 / numpy concatenate로 bind

```
np.random.seed(1234) #fix the random seed
N= 10**3
x = np.random.rand(N, 1)*2-1 # runif() generates U(0,1)
y = np.random.rand(N, 1)*2-1 # runif() generates U(0,1)
t = np.sqrt(x**2+y**2)
cbind=np.concatenate((x, y, t), axis=1)
print(cbind[:5]) #always display and check!
```

```
## [[-0.6169611  -0.19778718  0.64788947]
##  [ 0.24421754  0.8612288   0.8951856 ]
##  [-0.12454452  0.03067229  0.12826585]
##  [ 0.57071717  0.61916404  0.84207018]
##  [ 0.55995162  0.76354446  0.9468611 ]]
```

```
pi_hat = 4*np.sum(t <= 1)/MC_N
print(pi_hat)
```

```
## 3.06
```

p.12 Vectorized programming

R code

```
beg_time <- Sys.time()
set.seed(1234)
N <- 10^6
x <- runif(N)*2-1
y <- runif(N)*2-1
t <- sqrt(x^2+y^2)
pi_hat <- 4*sum(t<=1)/N
end_time <- Sys.time()
print(end_time-beg_time)
```

Python code 시간측정하는 함수를 제외하고 코드가 거의 일치합니다.
시간측정 함수
1. time.time() #import time
2. datetime.now() #from datetime import datetime
3. timeit.default_timer() #import timeit

```
import timeit
start_time = timeit.default_timer()
np.random.seed(1234)
N = 10**6
x = np.random.uniform(low=0,high=1,size=N)*2-1
y = np.random.uniform(low=0,high=1,size=N)*2-1
t = np.sqrt(x**2+y**2)
pi_hat = 4*np.sum(t<=1)/N
end_time = timeit.default_timer()
print("%f secs"%(end_time - start_time))
```

```
## 0.126648 secs
```

R code

```r
beg_time <- Sys.time()
set.seed(1234)
N <- 10^6
count <- 0
for (i in 1:N) {
x_i <- runif(1)*2-1
y_i <- runif(1)*2-1
t_i <- sqrt(x_i^2+y_i^2)
if (t_i <= 1) count <- count + 1
}
pi_hat <- 4*count/N
end_time <- Sys.time()
print(end_time-beg_time)
```

Python code

```python
import timeit
begin_i = timeit.default_timer()
MC_N = 100000
count = 0
for i in range(MC_N):
    x_i = np.random.uniform(-1,1,1)
    y_i = np.random.uniform(-1,1,1)
    t_i = np.sqrt(x_i**2+y_i**2)
    if (t_i <= 1):
        count += 1
pi_hat_i = 4*count/MC_N
end_i = timeit.default_timer()
print(end_i - begin_i,'secs')
```

```
## 1.6076865 secs
```

p.13 Implementation - varying number of trials

R code

```
pi_simulator <- function(N) {
set.seed(1234)
x <- runif(N)*2-1
y <- runif(N)*2-1
t <- sqrt(x^2+y^2)
pi_hat <- 4*sum(t<=1)/N
return(pi_hat)
}
num_trials <- 10^(2:7)
outcomes <- sapply(num_trials, pi_simulator)
results <- cbind(num_trials, outcomes)
results
```

Python code
list나 하나씩 출력형태로 작성한 학생도 있으나 pandas를 사용하여야 lecture note와 가장 근접하게 결과물이 나왔습니다.

```
def pi_simulator(MC_N):
    np.random.seed(seed=1234)
    x= np.random.uniform(low=-1, high=1, size=MC_N)
    y= np.random.uniform(low=-1, high=1, size=MC_N)
    t=np.sqrt(x**2+y**2)
    pi_hat=4*np.sum(t<=1)/MC_N

    return(pi_hat)

num_trials=list(map(lambda x: 10**x, range(2,8)))
outcomes=list(map(pi_simulator,num_trials))
results=pd.DataFrame({'num_trials': num_trials,'outcomes':outcomes})
results
```

```
##    num_trials  outcomes
## 0         100  2.960000
## 1        1000  3.060000
## 2       10000  3.135200
```

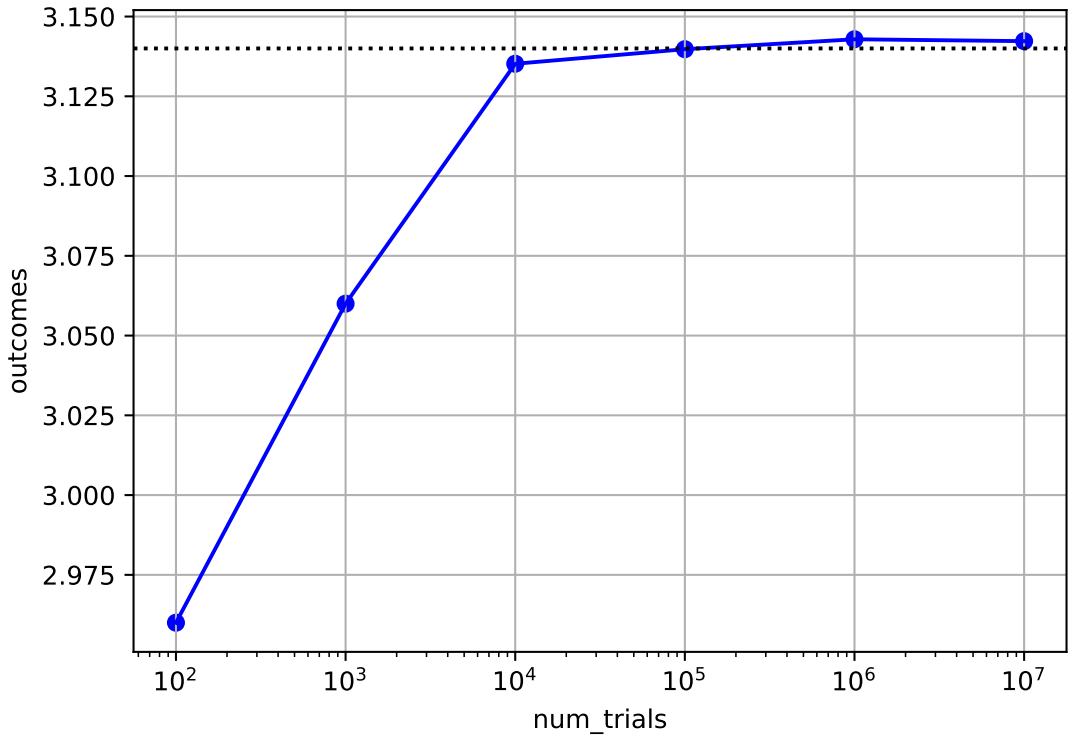
```
## 3      100000  3.139760
## 4     1000000  3.142876
## 5    10000000  3.142289
```

The previous figure was plotted by the following code. R code

```
results <- data.frame(results)
library(tidyverse)
ggplot(results, aes(x=num_trials, y=outcomes)) +
geom_point(color = "blue") + geom_path(color = "blue") +
geom_abline(slope = 0, intercept = 3.14159, linetype = "dotted") +
scale_x_log10() +
theme_minimal() + theme(text = element_text(size=25))
```

Python code

```
import matplotlib.pyplot as plt
plt.scatter(results['num_trials'],results['outcomes'], c='blue')
plt.plot(results['num_trials'],results['outcomes'], c='blue')
plt.axhline(3.14,0,1,color='black',linestyle=':')
plt.xscale('log')
plt.grid(True,axis='both')
plt.xlabel('num_trials')
plt.ylabel('outcomes')
plt.show()
```


3.150
3.125
3.100
3.075
3.050
3.025
3.000
2.975
outcomes
10^2
10^3
10^4
10^5
10^6
10^7
num_trials

p.17 Computation Time

R code

```
pi_simulator2 <- function(N) { # name change
beg_time <- Sys.time() # newly added
set.seed(1234)
x <- runif(N)*2-1
y <- runif(N)*2-1
t <- sqrt(x^2+y^2)
pi_hat <- 4*sum(t<=1)/N
end_time <- Sys.time() # newly added
print(N)
print(end_time-beg_time) # newly added
return(pi_hat)
}
sapply(num_trials, pi_simulator2)
```

Python code

대체적으로 결과값이 비슷하나 시간차이가 크게나는 학생이 3명 있었습니다.

보통 0.5초 이내 완료되나 20~30초까지 찍히는 코드가 있습니다

Uniform 함수 자체에서 바로난수를 생성하는 것과, U(0,1) 생성 후 연산하는 방법에서 시간차이가 발생하는 것 같습니다

-시간차이가 얼마 나지 않는 코드

```
import time
def pi_simulator2(MC_N): # name change
  beg_time=time.time() # newly added
  np.random.seed(seed=1234)
  x= np.random.uniform(low=-1, high=1, size=MC_N)
  y= np.random.uniform(low=-1, high=1, size=MC_N)
  t=np.sqrt(x**2+y**2)
  pi_hat=4*np.sum(t<=1)/MC_N
  end_time=time.time() # newly added
  print(MC_N)
  print("Time diffrence of",end_time-beg_time,"secs") # newly added
  return(pi_hat)
```

```
#1.
num_trials = [10**i for i in range(2,8)]
list(map(pi_simulator2, num_trials))
#2.
#print(*list(map(pi_simulator2,num_trials)))
```

```
## 100
## Time diffrence of 0.0009965896606445312 secs
## 1000
## Time diffrence of 0.0 secs
## 10000
## Time diffrence of 0.0 secs
## 100000
## Time diffrence of 0.004987001419067383 secs
## 1000000
## Time diffrence of 0.03989291191101074 secs
## 10000000
## Time diffrence of 0.4129014015197754 secs
## [2.96, 3.06, 3.1352, 3.13976, 3.142876, 3.1422888]
```

-시간차이가 많이나는 코드

```
def pi_simulator2(N):
  beg_time=time.time()
  np.random.seed(1234)
  x=np.random.uniform(0,1,size=N)*2-1
  y=np.random.uniform(0,1,size=N)*2-1
  t=np.sqrt(x**2+y**2)
  pi_hat=4*sum(t<=1)/N
  end_time=time.time()
  print(N)
  print('Time difference of ',end_time-beg_time, 'secs')
  return pi_hat
#1.
num_trials=10**np.arange(2,8)
[pi_simulator2(i) for i in num_trials]
#2.
#outcomes = np.vectorize(pi_simulator2)(num_trials)
```

```
## 100
## Time difference of  0.0 secs
## 1000
## Time difference of  0.003958463668823242 secs
## 10000
## Time difference of  0.03293800354003906 secs
## 100000
## Time difference of  0.31316232681274414 secs
## 1000000
## Time difference of  3.043837070465088 secs
## 10000000
## Time difference of  30.630128145217896 secs
## [2.96, 3.06, 3.1352, 3.13976, 3.142876, 3.1422888]
```

p.22 Repetitive simulation experiments

R code

```r
pi_simulator3 <- function(N) { # name change
# set.seed(1234) # seed must not be fixed
x <- runif(N)*2-1
y <- runif(N)*2-1
t <- sqrt(x^2+y^2)
pi_hat <- 4*sum(t<=1)/N
return(pi_hat)
}
n <- 100 # number of experiments to repeat
N <- 1000 # number of simulation repetition in a single experiment
set.seed(1234)
samples <- rep(0, n) # create an empty zero vector
for (i in 1:n) { # do this for n times
samples[i] <- pi_simulator3(N)
}
head(samples)
```

```python
def pi_simulator3(MC_N):
    x = np.random.uniform(-1,1,MC_N)
    y = np.random.uniform(-1,1,MC_N)
    t = np.sqrt(x**2 +y**2)
    pi_hat =4*sum(t<=1) / MC_N

    return pi_hat

n = 100
N = 1000

samples = np.zeros(n)
for i in range(n):
    samples[i] = pi_simulator3(N)

print(samples[:6])
```

```
## [3.088 3.188 3.096 3.112 3.144 3.176]
```

R code

```
X_bar <- mean(samples)
s <- sqrt(sum((X_bar-samples)^2)/(n-1))
t <- qt(p=0.975, df = n-1)
```

Python code

```
import numpy as np
from scipy import stats

x_bar=np.mean(samples)
s=np.sqrt(sum((x_bar-samples)**2)/(n-1))
t=stats.t(df=n-1).ppf((0.975)) #2. stats.t.ppf(0.975, n-1)

print(x_bar, s, t)
```

```
## 3.13936 0.06076075288300081 1.9842169515086827
```

p.24 Confidence interval Ex.1

R code

```
n <- 100 # number of exp. to rep.
N <- 10000 # number of sim. rep. in a single exp.
set.seed(1234)
samples <- rep(0, n)
for (i in 1:n) {
samples[i] <- pi_simulator3(N)
}
X_bar <- mean(samples)
s <- sqrt(sum((X_bar-samples)^2)/(n-1))
t <- qt(p=0.975, df = n-1)
lb <- X_bar-t*s/sqrt(n) # lower bound
ub <- X_bar+t*s/sqrt(n) # upper bound
```

Python code

```
import numpy as np
from scipy import stats
n = 100
N = 10000

samples=np.zeros(n)
for i in range(n):
    samples[i]=pi_simulator3(N)

x_bar=np.mean(samples)
s=np.sqrt(sum((x_bar-samples)**2)/(n-1))
t=stats.t(df=n-1).ppf((0.975))
lb=x_bar-t*s/np.sqrt(n)
ub=x_bar+t*s/np.sqrt(n)
print(lb,ub,ub-lb)
```

```
## 3.135871316902861 3.1426006830971396 0.006729366194278441
```

p.25 Confidence interval Ex.2

R code

```
n <- 1000 # number of exp. to rep.
N <- 10000 # number of sim. rep. in a single exp.
set.seed(1234)
samples <- rep(0, n)
for (i in 1:n) {
samples[i] <- pi_simulator3(N)
}
X_bar <- mean(samples)
s <- sqrt(sum((X_bar-samples)^2)/(n-1))
t <- qt(p=0.975, df = n-1)
lb <- X_bar-t*s/sqrt(n) # lower bound
ub <- X_bar+t*s/sqrt(n) # upper bound
```

Python code

```
import numpy as np
from scipy import stats
n = 1000
N = 10000

samples=np.zeros(n)
for i in range(n):
    samples[i]=pi_simulator3(N)

x_bar=np.mean(samples)
s=np.sqrt(sum((x_bar-samples)**2)/(n-1))
t=stats.t(df=n-1).ppf((0.975))
lb=x_bar-t*s/np.sqrt(n)
ub=x_bar+t*s/np.sqrt(n)
print(lb,ub,ub-lb)
```

```
## 3.140481280182024 3.1424987198179752 0.002017439635951135
```