B1_Newsvendor Exercise

Jaemin Park

2021-01-03

차 례

Exercises

p.11 Implementation - basic

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!
```

Python code

```
MC_N = 1000
x = np.random.uniform(-1,1,MC_N)
y = np.random.uniform(-1,1,MC_N)
z = x*x +y*y
t = np.sqrt(z)

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

```
## 3.076
```

p.12 Vectorized programming

vectorized programming - Python

```
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)
```

vectorized programming - Python

```
import timeit
begin = timeit.default_timer()
MC_N = 100000
x = np.random.uniform(-1,1,MC_N)
y = np.random.uniform(-1,1,MC_N)
z = x**2 +y**2
t = np.sqrt(z)

num = (t<=1).sum()
pi_hat = num * 4 / MC_N
end = timeit.default_timer()
print(end-begin)
```

```
## 0.0898071
```

with Iterator - R code

```
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)
```

with Iterator - 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)[0]
    y_i = np.random.uniform(-1,1,1)[0]
    z_i = x_i**2 + y_i**2
    t_i = np.sqrt(z_i)
    if (t_i <= 1):
        count += 1
pi_hat_i = 4*count/MC_N
end_i = timeit.default_timer()
print(end_i - begin_i)
```

```
## 1.3863726
```

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

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

print('Trial: %i, Outcome: %f' % (100, pi_simulator(100)))
```

```
## Trial: 100, Outcome: 2.880000
```

```
print('Trial: %i, Outcome: %f' % (1000, pi_simulator(1000)))
```

```
## Trial: 1000, Outcome: 3.176000
```

```
print('Trial: %i, Outcome: %f' % (10000, pi_simulator(10000)))
```

```
## Trial: 10000, Outcome: 3.140400
```

```
print('Trial: %i, Outcome: %f' % (100000, pi_simulator(100000)))
```

```
## Trial: 100000, Outcome: 3.143040
```

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

```
import timeit
def pi_simulator2(MC_N):
    begin = timeit.default_timer()
    x = np.random.uniform(-1,1,MC_N)
    y = np.random.uniform(-1,1,MC_N)
    z = x**2 +y**2
    t = np.sqrt(z)

    num = (t<=1).sum()
    pi_hat = num * 4 / MC_N
    end = timeit.default_timer()
    print('N: %i, Time difference: %f secs' %(MC_N,end-begin))

MC_N = 10**np.arange(2,8)
for i in MC_N:
    pi_simulator2(i)
```

```
## N: 100, Time difference: 0.000117 secs
```

```
## N: 1000, Time difference: 0.000097 secs
## N: 10000, Time difference: 0.000565 secs
## N: 100000, Time difference: 0.006519 secs
## N: 1000000, Time difference: 0.059636 secs
## N: 10000000, Time difference: 0.443720 secs
```

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)
    z = x**2 +y**2
    t = np.sqrt(z)
    num = (t<=1).sum()
    pi_hat = num * 4 / MC_N

    return pi_hat

n = 100
N = 1000

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

print(samples[:6])
```

```
## [3.176, 3.188, 3.1, 3.208, 3.172, 3.168]
```

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

```
## [3.176, 3.22, 3.184, 3.168, 3.168, 3.06]
```

```
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))

print(x_bar, s, t)
```

```
## 3.1521200000000005 0.05211783075712463 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

```
## [3.052, 3.212, 3.168, 3.124, 3.032, 3.184]
```

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

samples=[]
for i in range(n):
    samples.append(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)
```

```
## 3.140496366152456 3.146951633847545 0.006455267695089084
```

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

```
## [3.24, 3.168, 3.144, 3.18, 3.08, 3.068]
```

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

samples=[]
for i in range(n):
    samples.append(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)
```

```
## 3.1395262637788193 3.141576936221181 0.0020506724423619005
```