Lecture A4. Simulation 1

Sim, Min Kyu, Ph.D., mksim@seoultech.ac.kr

서울과학기술대학교 데이터사이언스학과

I. Motivation - estimation of π

원주율

3.1415...

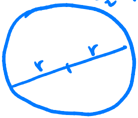
What is π?

- π is defined as

$$\pi = \frac{\text{a circle's circumference}}{\text{a circle's diameter}} \qquad \frac{C}{2r}$$

- To list a few reasons why π is such an important quantity:
  - In Architecture
  - In Construction
  - In Art
  - In Military operation
  - so many $\cdots$

둘레: C



$C = 2\pi r$

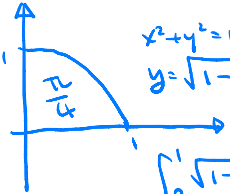
$\pi = \frac{C}{2r}$

How to estimate?

- In your elemetary school

- From your high school, you learned that

$$
\begin{aligned}
\text{(quarter size of a unit circle)} &= \int_0^1 \sqrt{1-x^2}\,dx \\
&= \pi/4
\end{aligned}
$$

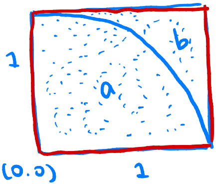


$x^2+y^2=1$

$y=\sqrt{1-x^2}$

$\int_0^1 \sqrt{1-x^2}\,dx = \frac{\pi}{4}$

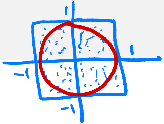
- In ancient days, people used $\pi/4 = \int_0^1 \sqrt{1-x^2}dx$ to estimate π.



$a+b : a = 1 : \frac{\pi}{4}$

II. Simulation approach

Design



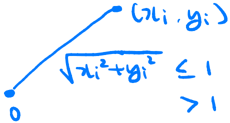
X : a random variable

$\mathbf{x}$: a vector

x_i : a number

- Step 1.
  - Let $X \sim U(-1, 1)$ and $Y \sim U(-1, 1)$.
  - Generate two vectors length of N, i.e. $\mathbf{x} = (x_1, x_2, ..., x_N)$ and $\mathbf{y} = (y_1, y_2, ..., y_N)$, where x_i is a sample of X and y_i is a sample of Y for all i.
- Step 2.
  - Let $t_i := \sqrt{x_i^2 + y_i^2}$ for all i, that is,

$$
\begin{aligned}
t_1 &:= \sqrt{x_1^2 + y_1^2} \\
t_2 &:= \sqrt{x_2^2 + y_2^2} \\
\cdots & \quad\ \ \cdots \\
t_N &:= \sqrt{x_N^2 + y_N^2}
\end{aligned}
$$



	면적	개수
□	4	N
○	$\pi \cdot 1^2$	number of $\{t_i \le 1\}$

$4 : N = \pi : \#\text{ of } \{t_i \le 1\}$

$\Rightarrow \pi = 4 \times \dfrac{\#\text{ of } \{t_i \le 1\}}{N}$

- Step 3.

$$
\hat{\pi} = 4 \times \frac{\text{number of } \{t_i \le 1\}}{N} = 4 \times \frac{\sum_{i=1}^{N} I_{\{t_i \le 1\}}}{N}
$$

$I_{\{t_i \le 1\}} = \begin{cases} 1 & \text{if } t_i \le 1 \\ 0 & \text{otherwise} \end{cases}$

(= binary fn)

Remark 1

The function $I_{\{\cdot\}}$ is called an *indicator function* that returns 1 if the statement is true and 0 if false.

- $\frac{\sum_{i=1}^{N} I_{\{t_i \leq 1\}}}{N}$ counts the number of t_i that is less than or equal to 1, among all i.

A statistical software, R

- You should comfortably interchange between R and python.
- Resources 1 - 'R for Python user'
  - https://medium.com/@nawazahmad20/r-for-python-programmers-part-1-ca4eab668b8c
  - http://ramnathv.github.io/pycon2014-r/
- Resources 2 - `datacamp.com`
  - `datacamp.com` allows access for >100 courses for R and python.
  - I suggest you at least do 'Introduction to R' and 'Intermediate R'.
  - You can subscribe for free using your `@seoultech.ac.kr` email with the following link.
  - https://www.datacamp.com/groups/shared_links/b3b5fc6f798aaf54ada0c03cee875c0099c34e300f5be6b8375e4850646b0b59
  - The above link expires every March and September, but I always renew it.
  - You can visit my github and find the link https://github.com/aceMKSim/teaching/
  - You can always request me for an invitation link if yours is expired.
- Resources 3 - lecture material for data visualization
  - From the following repository, study L01-L03 for installation and basic usage.
  - https://github.com/aceMKSim/teaching/tree/master/Data%20Visualization/Lecture%20Notes

Implementation - basic

~~N~~ MC_N

Monte Carlo

- Implementation with 1000 repetitions.

runif(MC_N, -1, 1)

U(-1,1)

-1

U(0,2)

×2

```
set.seed(1234) # fix the random seed
MC_N <- 10^3
x <- runif(MC_N)*2-1 # runif() generates U(0,1)
y <- runif(MC_N)*2-1 # this code generates U(-1,1)
t <- sqrt(x^2+y^2)
head(cbind(x,y,t)) # always display and check!
```

```
##               x          y         t
## [1,] -0.7725932  0.6752678 1.0261028
## [2,]  0.2445988 -0.0250675 0.2458800
## [3,]  0.2185495 -0.7793260 0.8093904
## [4,]  0.2467589 -0.2972740 0.3863441
## [5,]  0.7218308  0.5221261 0.8908733
## [6,]  0.2806212 -0.2206703 0.3569925
```

$\sqrt{(-0.77)^2+(0.67)^2}$

원 밖에 있음

원 안에 있음

```
pi_hat <- 4*sum(t<=1)/MC_N
pi_hat
```

```
## [1] 3.188
```

$\hat{\pi} = 4 \times \frac{\sum I\{t_i \le 1\}}{n}$

1. `set.seed()` fixes randomization, which is often convenient to get consistent outcome.
2. `runif(MC_N)` generates a vector of length `MC_N`, where each element follows $U(0,1)$.
3. `runif(MC_N)*2` follows $U(0,2)$, and `runif(MC_N)*2-1` follows $U(-1,1)$.
4. `t<=1` returns the 0-1 vector of length same as `t`, where an element is 1 if corresponding element in `t` is less than or equal to 1, and 0 otherwise.
5. `cbind()` combines (column) vectors into a matrix.
6. `head()` displays the first six observations.

Vectorized programming

- From the previous slide

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

백만번

$x, y \to t$

$\downarrow$

$\hat{\pi}$

```
## Time difference of 0.07878995 secs
```

0.08

- What a first-timer would write.

```r
beg_time <- Sys.time()
set.seed(1234)
MC_N <- 10^6
count <- 0
for (MC_i in 1:MC_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/MC_N
end_time <- Sys.time()
print(end_time-beg_time)
```

2.85

```
## Time difference of 2.858543 secs
```

- The style of the code on the left is called *vectorized programming*.
- It is elegant, economic, and efficient.
- You must be able to *write as the left* side and *communicate as the right* side (to non-expert).

Exercise 1

Write the two programs in the previous page with python and compare the computation time.

Implementation - varying number of trials

- Approach with a custom function

```
pi_simulator <- function(MC_N) {
  set.seed(1234)
  x <- runif(MC_N)*2-1
  y <- runif(MC_N)*2-1
  t <- sqrt(x^2+y^2)
  pi_hat <- 4*sum(t<=1)/MC_N
  return(pi_hat)
}
```

```
pi_simulator(100)
```

```
## [1] 3.04
```

```
pi_simulator(1000)
```

```
## [1] 3.188
```

```
pi_simulator(10000)
```

```
## [1] 3.1876
```

```
pi_simulator(100000)
```

```
## [1] 3.13432
```

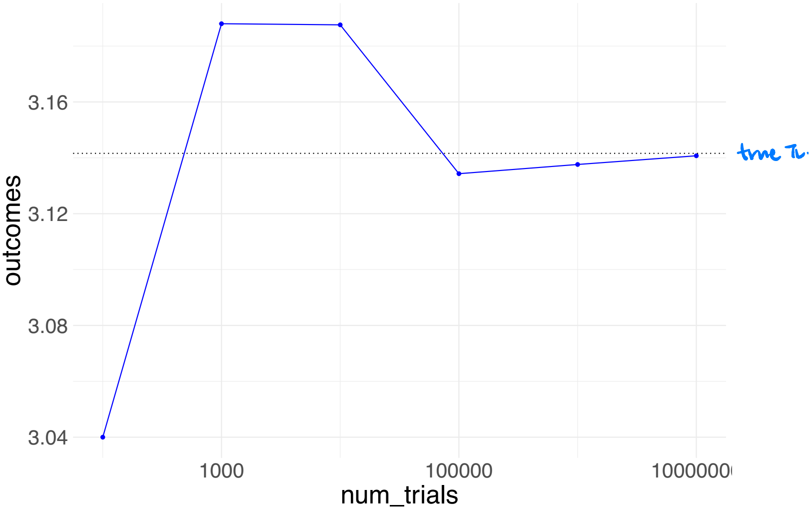
- How many repetition is necessary to get closer?

```
num_trials <- 10^(2:7)
outcomes <- sapply(num_trials, pi_simulator)
results <- cbind(num_trials, outcomes)
results
```

```
##      num_trials outcomes
## [1,]        100 3.040000
## [2,]       1000 3.188000
## [3,]      10000 3.187600
## [4,]     100000 3.134320
## [5,]    1000000 3.137616
## [6,]   10000000 3.140733
```

- `sapply(num_trials, pi_simulator)` *applies* the function `pi_simulator()` to *each element* of `num_trials`.

- How many repetition is necessary to get closer?



- (optional) The previous figure was plotted by the following code.

```r
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))
```

III. Discussion

Computation time

- In this implementation, number of trials were increased from 10^2 to 10^7. No wonder that this increases the computational time.
- Following modified function displays the elapsed time.

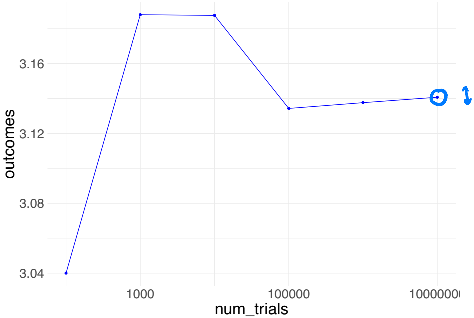
```
pi_simulator2 <- function(MC_N) { # name change
  beg_time <- Sys.time() # newly added
  set.seed(1234)
  x <- runif(MC_N)*2-1
  y <- runif(MC_N)*2-1
  t <- sqrt(x^2+y^2)
  pi_hat <- 4*sum(t<=1)/MC_N
  end_time <- Sys.time() # newly added
  print(MC_N)
  print(end_time-beg_time) # newly added
  return(pi_hat)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

```
sapply(10^(2:6), pi_simulator2)
```

```
## [1] 100
## Time difference of 0 secs
## [1] 1000
## Time difference of 0 secs
## [1] 10000
## Time difference of 0.0009980202 secs
## [1] 100000
## Time difference of 0.005984068 secs
## [1] 1000000
## Time difference of 0.07583094 secs
## [1] 3.040000 3.188000 3.187600 3.134320 3.137616
```

Confidence on the result

- In estimation of π example, we were in the luxurious situation because we already knew the correct value of π, 3.14159.
- In reality, this situation is rare. Rather, you shouldn't be in need of doing simulation after all if you already know the exact value.
- In reality, following figure is what you would normally face. Notice that the correct value indicating line is gone.



(± ·····)

- One legimate way to have some confidence is to observe the plot and say 'It seems converging with good degree'. Then, present a number that seems to be within the tolerance.
- Are there any way to present a confidence interval, just as good statistical estimations should present?

IV. Confidence interval

Motivation

A3. $\cdots$ $\mathbb{P}\left[\mu \in \left[\bar{x} - 1.96\frac{s}{\sqrt{n}}, \bar{x} + 1.96\frac{s}{\sqrt{n}}\right]\right] = 0.95$

- Building a confidence interval from experiment generally involves repetitive experiments. But in the simulation approach, we already do have the repetitive simulation experiments? Isn't this enough?
- Not really so. In order to build a confidence interval, we should treat *one entire simulation experiment* as *one observation*. For example, we treat the result from a `MC_N=1000` simulation experiment as a single observation on the true value. Then repeat this simulation experiment, say, n times, to build a confidence interval.
- Let's set `MC_N=1000` for a simulation experiment and do this for `n=100` times.

MC_N = 1000 $\rightarrow$

X_1
X_2
X_3
X_4
$\vdots$
X_n

$\rightarrow$ C.I.

Repetitive simulation experiments

- Let's set `MC_N=1,000` for a simulation experiment and do this for `n=100` times.
- For each experiment, record the result to collect `n=100` samples.

```
pi_simulator3 <- function(MC_N) { # name change
  # set.seed(1234) # seed must not be fixed
  x <- runif(MC_N)*2-1
  y <- runif(MC_N)*2-1
  t <- sqrt(x^2+y^2)
  pi_hat <- 4*sum(t<=1)/MC_N
  return(pi_hat)
}
```

```
n <- 100 # number of experiments to repeat
MC_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(MC_N)
}
head(samples)
```

$x_1 \ldots x_{100}$

of length n

100×1 vector

```
## [1] 3.188 3.144 3.060 3.240 3.148 3.172
```

- From LN.A~~4~~3.p~~14~~15,

$$\mathbb{P}[\overline{X} - \underline{t_{0.975,n-1}} \cdot s/\sqrt{n} \le \mu \le \overline{X} + t_{0.975,n-1} \cdot s/\sqrt{n}] = 0.95$$

- Obtain the numbers as follows:

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

```
X_bar
## [1] 3.137
s
## [1] 0.05186579
t
## [1] 1.984217
```

MC-N=1000
n= 100
⇓
length of CI=0.020

- Thus,

$$\mathbb{P}[3.137 - 1.984 \cdot 0.0519/\sqrt{100} \;\le\; \mu \;\le\; 3.137 + 1.984 \cdot 0.0519/\sqrt{100}] = 0.95$$
$$\Downarrow$$
$$\mathbb{P}[3.127 \;\le\; \mu \;\le\; 3.147] = 0.95$$

- Note that the length of interval was 0.020 (=3.147-3.127)
- Obviously, increasing MC_N and/or increasing n should narrow the confidence interval.

Exercise 2

Do the above experiment with `MC_N` *increased by the factor of ten, and present the confidence interval. (Use* `set.seed(1234)`*)*

```
n <- 100 # number of exp. to rep.
MC_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(MC_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
```

```
lb
## [1] 3.137164
ub
## [1] 3.143788
ub-lb # length of CI
## [1] 0.006624215
```

$\frac{1}{\sqrt{10}}$ 배가 된다.

Exercise 3

Do the Exercise above with n *increased by the factor of ten, and present the confidence interval. (Use* `set.seed(1234)`*)*

```r
n <- 1000 # number of exp. to rep.
MC_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(MC_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
```

```r
lb
## [1] 3.139777
ub
## [1] 3.141834
ub-lb
## [1] 0.002057237
```

MC-N	n	
1000	100	0.02
10000	100	0.0066
10000	1000	0.002

Computation cost and The accuracy

MC_N	n	length of CI
1,000	100	0.020
1,0000	100	0.0066
1,0000	1000	0.00205

- Increasing `MC_N` or `n` gives the same effect.
    - When `MC_N` was increased by the factor of 10, the length of CI was decreased by the factor of $\sqrt{10}$.
    - When `n` was increased by the factor of 10, the length of CI was decreased by the factor of $\sqrt{10}$.
- Repetitive simulation experiments is beneficial if $\cdots$
    - when you need confidence interval.
    - when you face memory issue that prevents increasing `MC_N` any more.

Exercise 4

Present a previous page's table by writing a neat python code block.

```
"If I only had an hour to chop down a tree, I would spend the first 45 minutes sharpening my axe.
- A. Lincoln"
```