

## Lecture A4. Simulation 1

Sim, Min Kyu, Ph.D., [mksim@seoultech.ac.kr](mailto:mksim@seoultech.ac.kr)



서울과학기술대학교 데이터사이언스학과

1 I. Motivation - estimation of  $\pi$

2 II. Simulation approach

3 III. Discussion

4 IV. Confidence interval

# I. Motivation - estimation of $\pi$

# What is $\pi$ ?

- $\pi$  is defined as

$$\pi = \frac{\text{a circle's circumference}}{\text{a circle's diameter}}$$

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

## How to estimate?

- In your elementary school
- From your high school, you learned that

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

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

## II. Simulation approach

# Design

- Step 1.
  - Let  $X \sim U(-1, 1)$  and  $Y \sim U(-1, 1)$ .
  - Generate two vectors length of  $N$ , i.e.  $\mathbf{x} = (x_1, x_2, \dots, x_N)$  and  $\mathbf{y} = (y_1, y_2, \dots, y_N)$ , 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,

$$t_1 := \sqrt{x_1^2 + y_1^2}$$

$$t_2 := \sqrt{x_2^2 + y_2^2}$$

$$\dots \quad \dots$$

$$t_N := \sqrt{x_N^2 + y_N^2}$$

- Step 3.

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



## Remark 1

The function  $I_{\{.\}}$  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

- I am a firm believer of that you should comfortably interchange between R and python.
- Resources 1 - 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](https://www.datacamp.com/groups/shared_links/b3b5fc6f798aaf54ada0c03cee875c0099c34e300f5be6b8375e4850646b0b59)
  - The above link expires at March 2021, but I always renew it.
  - You can always request me for an invitation link if yours is expired.
- Resources 2 - 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

- Implementation with 1000 repetitions.

```
set.seed(1234) # fix the random seed
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!
```

```
##           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
```

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

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

- 1 `set.seed()` fixes randomization, which is often convenient to get consistent outcome.
- 2 `runif(N)` generates a vector of length  $N$ , where each element follows  $U(0, 1)$ .
- 3 `runif(N)*2` follows  $U(0, 2)$ , and `runif(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

```
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)
```

## Time difference of 0.1346781 secs

- What first-timer would write.

```
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)
```

## Time difference of 6.264948 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).

## Implementation - varying number of trials

- Approach with a custom function

```
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)  
}
```

```
pi_simulator(100)
```

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

```
pi_simulator(1000)
```

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

```
pi_simulator(10000)
```

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

```
pi_simulator(100000)
```

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

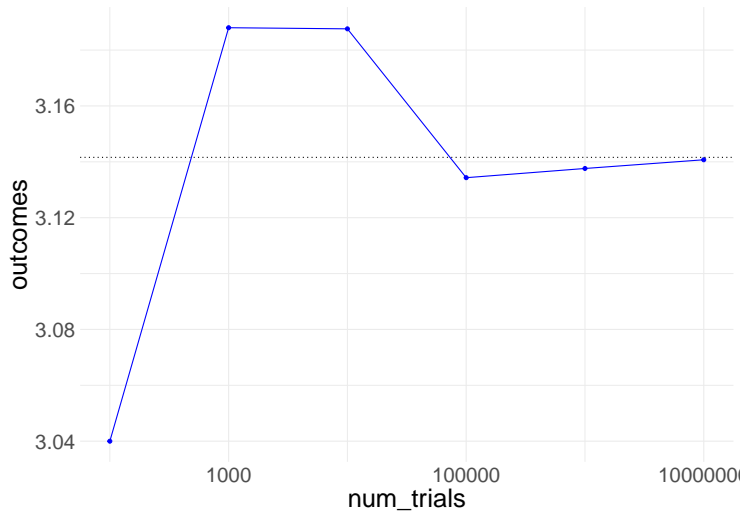
- How many repetition is necessary to get closer?

```
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?



The previous figure was plotted by the following 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))
```

## 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.

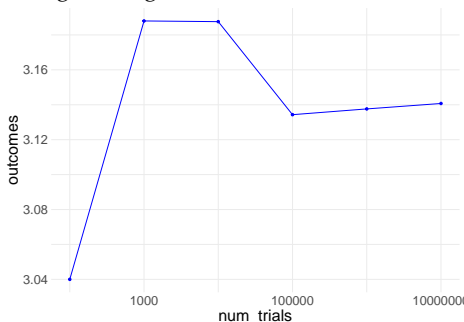
```
pi_simulator2 <- function(N) { # name change
  beg_time <- Sys.time() # newly added
  set.seed(1234)
  x <- runif(N)*2-1
  y <- runif(N)*2-1
  t <- sqrt(x^2+y^2)
  pi_hat <- 4*sum(t<=1)/N
  end_time <- Sys.time() # newly added
  print(N)
  print(end_time-beg_time) # newly added
  return(pi_hat)
}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

```
apply(num_trials, pi_simulator2)
```

```
## [1] 100
## Time difference of 0 secs
## [1] 1000
## Time difference of 0.0009970665 secs
## [1] 10000
## Time difference of 0.0009970665 secs
## [1] 100000
## Time difference of 0.009316921 secs
## [1] 1000000
## Time difference of 0.118423 secs
## [1] 10000000
## Time difference of 1.416709 secs
## [1] 3.040000 3.188000 3.187600 3.134320 3.137616 3
```

## Confidence on the result

- In estimation of  $\pi$  example, we were in the luxurious situation because we already knew the correct value of  $\pi$ , 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 legitimate 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

- 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  $N = 1000$  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  $N = 1,000$  for a simulation experiment and do this for  $n = 100$  times.

## Repetitive simulation experiments

- Let's set  $N = 1,000$  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(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)
```

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

- From LN.A4.p13,

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

- 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
```

Thus,

$$\mathbb{P}[3.137 - 1.984 \cdot 0.0519/\sqrt{100} \leq \mu \leq 3.137 + 1.984 \cdot 0.0519/\sqrt{100}] = 0.95$$

$$\Downarrow$$

$$\mathbb{P}[3.127 \leq \mu \leq 3.147] = 0.95$$

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

### Exercise 1

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

```
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
```

```
lb
## [1] 3.137164
ub
## [1] 3.143788
ub-lb
## [1] 0.006624215
```



## Exercise 2

*Do the Exercise 1 above with  $n$  increased by the factor of ten, and present the confidence interval. (Use `set.seed(1234)`)*

```
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
```

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

## Computation cost and The accuracy

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

- Increasing  $N$  or  $n$  gives the same effect.
  - When  $N$  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...
  - when you need confidence interval.
  - when you face memory issue that prevents increasing  $N$  any more.



"Man can learn nothing unless he proceeds from the known to the unknown. - Claude Bernard"