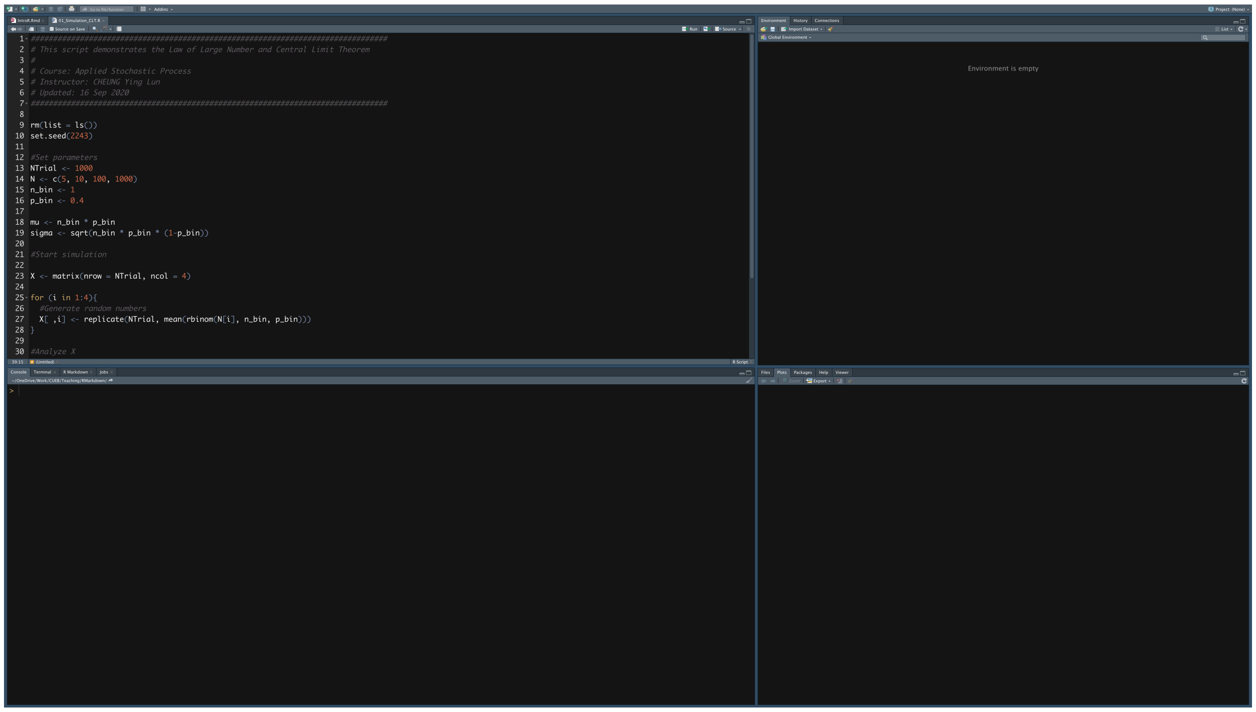
Applied Stochastic Process

Introduction to R

CHEUNG Ying Lun

1 R

R is a language and environment for statistical computing and graphics. It is an integrated suite of software facilities for data manipulation, calculation and graphical display. You can freely download R at the CRAN mirror available at https://cran.r-project.org/mirrors.html. In this course, we will be using RStudio as the integrated development environment (IDE) for R. RStudio is freely available at https://rstudio.com/products/rstudio/.



```
################################################################################
# This script demonstrates the Law of Large Number and Central Limit Theorem
#
# Course: Applied Stochastic Process
# Instructor: CHEUNG Ying Lun
# Updated: 16 Sep 2020
################################################################################

rm(list = ls())
set.seed(2243)

#Set parameters
NTrial <- 1000
N <- c(5, 10, 100, 1000)
n_bin <- 1
p_bin <- 0.4

mu <- n_bin * p_bin
sigma <- sqrt(n_bin * p_bin * (1-p_bin))

#Start simulation

X <- matrix(nrow = NTrial, ncol = 4)

for (i in 1:4){
  #Generate random numbers
  X[ ,i] <- replicate(NTrial, mean(rbinom(N[i], n_bin, p_bin)))
}

#Analyze X
```

```
~/OneDrive/Work/CUEB/Teaching/RMarkdown/
> 
```

Figure 1: Screenshot of RStudio

There are four main windows in RStudio: script editor (top left), console (bottom left), environment (top right) and plots (bottom right). Usually, we write a script of R codes in the editor, and then run the codes in the console. When we assign new variables, they will show up in the environment window. If we plot a figure, it will show up in the plots window.

2 Syntax

When you are not familiar with a function, you can use `?` to get the description and the usage of the specific function. For example, `?sum` gets you to the help file of the function `sum()`.

2.1 Creating a variable

To assign value to a variable, we can use `<-` or `=`. There is a slight difference between the two function: `<-` always create a variable or assign the value to the variable in the current environment, while `=` can also be used to assign values to input arguments of a function. It is recommended to use `<-` when creating new variables or assigning values to a variable. There are several data types in R. Most of the variables we will be dealing with are either *numeric*, *character* or *logical*.

```
x <- 100                    # Creating a numeric variable
s <- "The value of x is"   # Creating a string variable
b <- FALSE                  # Creating a logical variable
print(paste(s, x))          # Pasting x and s together, and then printing them out
```

```
## [1] "The value of x is 100"
```

2.2 Vectors and matrices

The use of vectors and matrices are also very common in R. A vector can be created by the `c()` function, while a matrix can be created with the `matrix()` function. The `i`-th element of a vector or matrix can be obtained by the square brackets `[]`.

```
v <- c(1, 2, 3)
print(v)
```

```
## [1] 1 2 3
```

```
M1 <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3, byrow = TRUE)
print(M1)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```
M2 <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3, byrow = FALSE)
print(M2)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
M1[1, ]
```

```
## [1] 1 2 3
```

```
M2[, 2]
```

```
## [1] 3 4
```

A special object in R is *list*. A list is a generic vector containing other objects. Suppose that `l` is a list variable, in which the first object contained is `v`, a vector. Then `l[1]` will be a slice of the list, i.e., it returns a list variable of size one, which contains `v` in the list. If we want to access `v` instead, then we have to use `l[[1]]`, the double squared bracket.

```
l = list(v, x, s, b)
l[c(2, 3)]
```

```
## [[1]]
## [1] 100
##
## [[2]]
## [1] "The value of x is"
```

```
l[[1]]
```

```
## [1] 1 2 3
```

```
l[[1]][2]
```

```
## [1] 2
```

2.3 Operators

2.3.1 Arithmetic operators

```
M1 + M2                           # Summation
```

```
##      [,1] [,2] [,3]
## [1,]    2    5    8
## [2,]    6    9   12
```

```
M1 - M2                           # Subtraction
```

```
##      [,1] [,2] [,3]
## [1,]    0   -1   -2
## [2,]    2    1    0
```

```
x * 2                             # Multiplication
```

```
## [1] 200
```

```
x / 2                             # Division
```

```
## [1] 50
```

```
c(1, 2, 3) ^ c(3, 2, 1)           # Exponent
```

```
## [1] 1 4 3
```

```
M1 %*% v                          # Matrix multiplication
```

```
##      [,1]
## [1,]   14
## [2,]   32
```

2.3.2 Relatonal operators

```
M1 > M2                           # Larger than
```

```
##       [,1]  [,2]  [,3]
## [1,] FALSE FALSE FALSE
## [2,]  TRUE  TRUE FALSE
```

```
M1 < M2                           # Smaller than
```

```
##       [,1]  [,2]  [,3]
## [1,] FALSE  TRUE  TRUE
## [2,] FALSE FALSE FALSE
```

```
M1 == M2                          # Equal to
```

```
##       [,1]  [,2]  [,3]
## [1,]  TRUE FALSE FALSE
## [2,] FALSE FALSE  TRUE
```

```
M1 >= M2                          # Larger than or equal to
```

```
##      [,1]  [,2]  [,3]
## [1,] TRUE FALSE FALSE
## [2,] TRUE  TRUE  TRUE
```

```
M1 <= M2                          # Smaller than or equal to
```

```
##       [,1]  [,2] [,3]
## [1,]  TRUE  TRUE TRUE
## [2,] FALSE FALSE TRUE
```

```
M1 != M2                          # Not equal to
```

```
##       [,1] [,2]  [,3]
## [1,] FALSE TRUE  TRUE
## [2,]  TRUE TRUE FALSE
```

2.3.3 Logical operators

```
c(TRUE, FALSE) & c(TRUE, TRUE)    # And
```

```
## [1]  TRUE FALSE
```

```
c(TRUE, FALSE) | c(TRUE, TRUE)    # Or
```

```
## [1] TRUE TRUE
```

2.4 Loops

2.4.1 For loop

A For loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times or over a specifc vector.

```
for (i in seq(from = 1, to = 5, by = 2)) {
  print(i)
}
```

```
## [1] 1
## [1] 3
## [1] 5
```

In the code above, `seq(from = 1, to = 5, by = 2)` creates a vector `c(1, 3, 5)`. The code within the For loop is executed three times, with `i = 1`, `i = 3` and `i = 5` respectively.

2.4.2 While loop

A While loop repeats the code within the loop while the specified condition is satisfied.

```
i <- 1
while(i < 5) {
  print(paste("i =", i))
  i <- i + 2
}
```

```
## [1] "i = 1"
## [1] "i = 3"
```

2.5 Random variables

The density (or probability mass function), distribution function, quantile function and the random generation for many standard distributions can be accessed easily with R. The function name follows the syntax 'Quantity prefix + Distribution suffix'.

Quantity prefix	Meaning
d	Density function
p	Cumulative distribution function
q	Quantile function
r	Random number generation

Table 1: Quantity prefix

Distribution suffix	Meaning
binom	Binomial distribution
pois	Poisson distribution
geom	Geometric distribution
exp	Exponential distribution
norm	Normal distribution
unif	Uniform distribution

Table 2: Distribution suffix

Before one starts generating random numbers with R, it is good practice to set the seed. Note that the random numbers generated by R are not really random. They are generated with a set of *deterministic* algorithm. Therefore, we can make sure that we get the exact same *random* numbers every time we run the codes by specifying the starting point of the algorithm. It can be done by the function `set.seed()`. Remember that you should set the seed *randomly*, i.e., you should avoid using `set.seed(1)` every time.

```r
set.seed(1651)
NTrial <- 1000
X_Unif <- runif(NTrial, min = -1, max = 1)
print(paste("Empirical probability of X < 0 is", sum(X_Unif < 0)/NTrial))
```

```
## [1] "Empirical probability of X < 0 is 0.498"
```

```r
print(paste("Theoretical probability of X < 0 is ", punif(0, min = -1, max = 1)))
```

```
## [1] "Theoretical probability of X < 0 is  0.5"
```

2.6 Useful functions

Two functions are very useful for our purpose, simulation. They are `replicate()` and `apply()`. The syntax of `replicate()` is `replicate(n, expr)`. The function repeatedly evaluates the expression `expr` for `n` times and output the results in a vector or matrix. For example, in the following code, `rnorm(10, mean = 2, sd = 2)` creates a vector of ten random numbers, drawn from the normal distribution with both mean and standard deviation being 2. Therefore, the whole command creates an 10 times `NTrial` matrix containing the normally distributed random numbers.

```r
X_Norm <- replicate(NTrial, rnorm(10, mean = 2, sd = 2))
```

The syntax of `apply()` is `apply(X, MARGIN, FUN)`. It means that we apply the same function `FUN` over the `MARGIN`-th dimension of `X`. In the following example, we are applying the function `mean()` over the 2nd dimension (i.e., over each column) of `X_Norm`. In other words, we first get each column out of `X`. Then, we apply the function `mean()` to that column and output the result. Therefore, the result `X_bar` is a vector of length `NTrial`, in which each element in the vector is the mean of the respective column of `X_Norm`.

```
X_bar <- apply(X_Norm, 2, mean)
```

2.7 Plots

The main function for plotting graphs is `plot(x, y, ...)`. Besides the `x` and `y` coordinates of points in the plot, there are many arugments one can pass to the method. One of the most important arguments is `type`, which indicates which type of plot should be drawn. For our purpose, the most common types will be `p` for points, `l` for lines or `b` for both. If one want another plot to overlay the existing one, one can use the function `lines(x, y)`.

Another type of plot we will be using quite often is the histogram, which can be plotted with command `hist(x, ...)`. An important argument is `freq`, which if set as `TRUE`, then the histogram shows the *frequencies* of the data in each bin. If it is set as `FALSE`, then the probability density will be plotted, so that the histogram has a total area of one. An example will be given below.

3 An Example: Law of Large Number and Central Limit Theorem

In this example, we demonstrate convergence in probability and distribution of the sample mean suggested by the Law of Large Number (LLN) and Central Limit Theorem (CLT), stated as follow:

Theorem 1 *Let $X_1, X_2, \ldots, X_N$ be a sequence of indpendent random variables with a common distribution and $\mathbb{E}(X_i) = \mu$ for all i. Then, with probability 1,*

$$\bar{X} = N^{-1} \sum_{i=1}^{N} X_i \xrightarrow{p} \mu \qquad \text{as } N \to \infty.$$

Theorem 2 *Let $X_1, X_2, \ldots, X_N$ be a sequence of indpendent identically distributed random variables with mean μ and variance σ^2 for all i. Then as $N \to \infty$,*

$$N^{-1/2} \sum_{i=1}^{N} \frac{X_i - \mu}{\sigma} = \frac{\sqrt{N}(\bar{X} - \mu)}{\sigma} \xrightarrow{d} \mathcal{N}(0, 1)$$

To begin with, we clear the environment and set the seed.

```
rm(list = ls())
set.seed(2243)
```

To study the LLN and CLT, we let $X_i \overset{\text{iid}}{\sim} Bin(n, p)$, $i = 1, \ldots, N$, and check whether the sample mean converges to the expected value of X_i as N increases. Recall that $\mathbb{E}(X_i) = np$ and $\text{var}(X_i) = np(1 - p)$. The parameters are set as follow:

```
NTrial <- 1000
N <- c(5, 10, 100, 1000)
n_bin <- 5
p_bin <- 0.4
mu <- n_bin * p_bin
sigma <- sqrt(n_bin * p_bin * (1-p_bin))
```

After assigning the values of the parameters, we can start our simulation. We first generate an empty matrix of dimension `NTrial` times 4. It is used to store the simulated values of $\bar{X}$ for the `NTrial` simulations and for the four different `N`.

```
X <- matrix(nrow = NTrial, ncol = 4)
for (i in 1:4){
  X[ ,i] <- replicate(NTrial, mean(rbinom(N[i], n_bin, p_bin)))
}
```

In the above, `mean(rbinom(N[i], n_bin, p_bin))` generates `N[i]` replicates of $X_i \overset{\text{iid}}{\sim} Bin(5, 0.4)$ and take the sample mean. This expression is repeated `NTrial` times and the output is a vector of length `NTrial`. It is stored in the `i`-th row of `X`. `X` is summarized as follow:

```
summary(X)
```

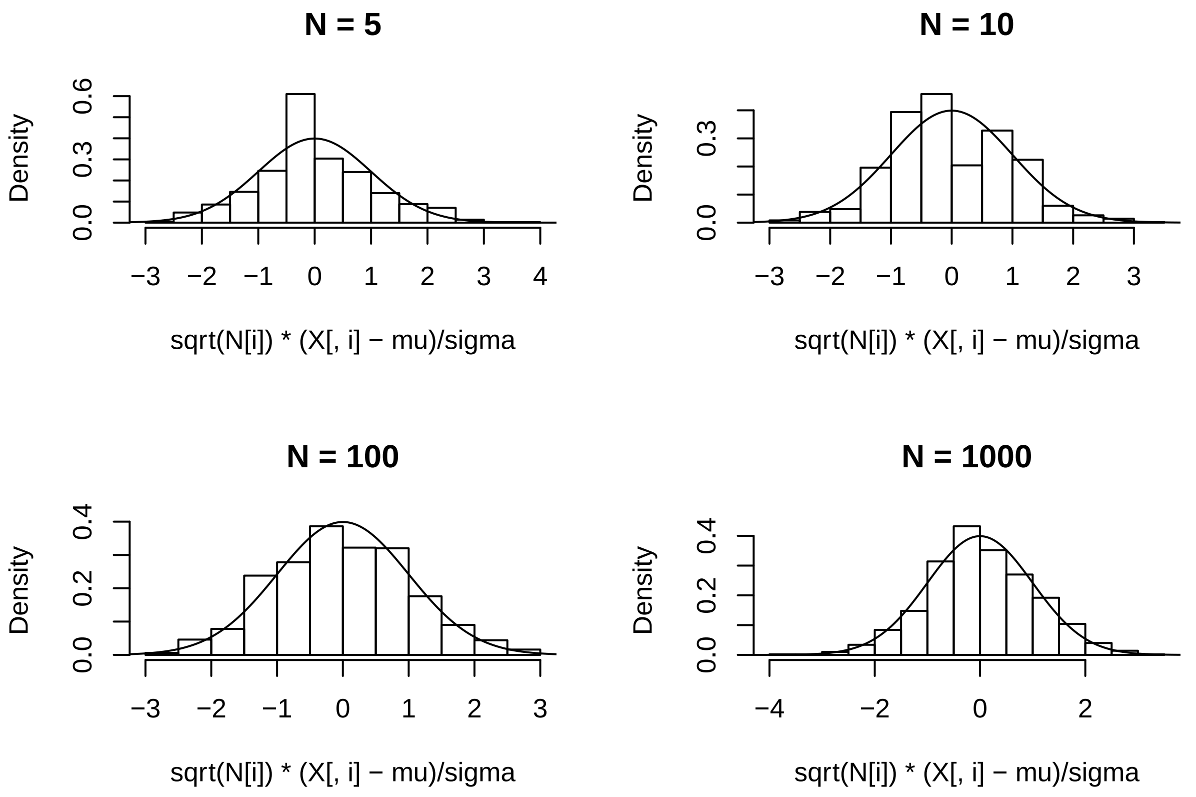
```
##        V1              V2              V3             V4       
##  Min.   :0.600   Min.   :1.000   Min.   :1.71   Min.   :1.875  
##  1st Qu.:1.600   1st Qu.:1.800   1st Qu.:1.92   1st Qu.:1.978  
##  Median :2.000   Median :2.000   Median :2.00   Median :1.999  
##  Mean   :2.023   Mean   :2.001   Mean   :2.00   Mean   :2.001  
##  3rd Qu.:2.400   3rd Qu.:2.200   3rd Qu.:2.08   3rd Qu.:2.024  
##  Max.   :3.800   Max.   :3.200   Max.   :2.31   Max.   :2.112
```

```
apply(X, 2, sd)
```

```
## [1] 0.49691759 0.33716891 0.11143578 0.03499827
```

We can see that both the range and the standard deviation of `X` shrinks as N increases. In the last column, the value of $\bar{X}$ is between 1.875 and 2.112, very close to the theoretical value 2. The decrease in SD also demonstrate the LLN. To demonstrate the CLT, we plot the histogram of the scaled value of $\bar{X}$.

```
x <- seq(-5, 5, 0.01)
par(mfrow = c(2,2))
for (i in 1:4) {
  hist(sqrt(N[i]) * (X[ , i] - mu) / sigma, freq = FALSE, main = paste0("N = ",N[i]))
  lines(x, dnorm(x))
}
```



The histogram above represents the empirical density of $\bar{X}$, while the curve plots the density function of the standard normal. We see that the empirical density converges to that of the standard normal when N increases, agreeing with the CLT.