Getting Started With R

Yashin Hsiao (cocacatyaya@gmail.com)

- For more information, please visit :
  http://cran.r-project.org/doc/manuals/R-lang.html
- How to become R people easily and cheerfully?
  ➔ **Google** + **? function_name** in R console is the only way out!

【Content】

1. Basic Operators

<-	Left assignment
->	Right assignment
-	Minus
+	Plus
*	Multiplication
/	Division
^	Exponentiation
%/%	Integer divide
%*%	Matrix product
%o%	Outer product
%x%	Kronecker product
<	Less than
>	Greater than
==	Equal to
>=	Greater than or equal to
<=	Less than or equal to
&	And
&&	And
|	Or
||	Or
:	Sequence
?	Help

```
ls()        # list all variables
ls.str()    # list details on variables and their types
rm(list=ls(all=TRUE)) # delete all variables in memory
rm(a_var) #delete variable a_var
```

- R code

```
x <- 10     # x eqauls to 5
10 -> x     # x equals to 5
y <- 5      # y equals to 10
x + y
x - y
x * y
x / y
x^y
```

2. Matrix Algebra in R

2.1 Create Vectors and basic calculations

- `as.vector` produces a vector of the given length and mode. The atomic modes are `"logical"`, `"integer"`, `"numeric"`, `"complex"`, `"character"` and `"raw"`.

- `seq` is a standard generic with a default method. `seq(from, to, by= )`

◆ R code

```
#First, we consider two vectors, vec1 and vec2.
vec1 <- as.vector(seq(from = 1, to = 20, 2))
# vec1 <- seq(from = 1, to = 20, by = 2)
> vec1
 [1]  1  3  5  7  9  11  13  15  17  19
> vec2 <- as.vector(seq(from = 2, to = 20, 2))
> vec2
 [1]  2  4  6  8  10  12  14  16  18  20

# replace particular one element in vector by other value
# > vec1[n] <- value1

# add a constant to vec1
> vec3 <- vec1 + 1
> vec3
 [1]  2  4  6  8  10  12  14  16  18  20

# addition
> vec4 <- vec2 + vec3
> vec4
 [1]  4  8  12  16  20  24  28  32  36  40

# multiply by a constant
> vec5 <- vec2 *2
> vec5
 [1]  4  8  12  16  20  24  28  32  36  40
# Simple multiplication;
# 1. Same length 2. element times element
> vec1
 [1]  1  3  5  7  9  11  13  15  17  19
> vec2
 [1]  2  4  6  8  10  12  14  16  18  20
> vec5 <- vec1 * vec2
```

```
> vec5
 [1]   2  12  30  56  90 132 182 240 306 380
```

Inner product

```
> vec5 <- vec1 %*% vec2
> vec5
     [,1]
[1,] 1430
```

Outer product

```
> vec5 <- vec1 %o% vec2
> vec5
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    2    4    6    8   10   12   14   16   18    20
 [2,]    6   12   18   24   30   36   42   48   54    60
 [3,]   10   20   30   40   50   60   70   80   90   100
 [4,]   14   28   42   56   70   84   98  112  126   140
 [5,]   18   36   54   72   90  108  126  144  162   180
 [6,]   22   44   66   88  110  132  154  176  198   220
 [7,]   26   52   78  104  130  156  182  208  234   260
 [8,]   30   60   90  120  150  180  210  240  270   300
 [9,]   34   68  102  136  170  204  238  272  306   340
[10,]   38   76  114  152  190  228  266  304  342   380
```

```
# create another vector, vec6, with negative values
> vec6 <- seq(from = -10, to = -1, by = 1)
> vec6
 [1] -10  -9  -8  -7  -6  -5  -4  -3  -2  -1

# get absolutely value of vec6
> vec6 <- abs(vec6)
> vec6
 [1] 10  9  8  7  6  5  4  3  2  1

# other common functions in R
> x <- c(1,2,3,4,5,6)
> sum(x)
[1] 21

> prod(x)
[1] 720
```

```
> max(x)
[1] 6

> min(x)
[1] 1

> which.max(x)        #return the index of the max element
[1] 6

> which.min(x)        #return the index
[1] 1

> range(x)
[1] 1 6

> length(x)
[1] 6

> mean(x)
[1] 3.5

> median(x)
[1] 3.5

> var(x)
[1] 3.5


> rev(x)
[1] 6 5 4 3 2 1

> sort(x)
[1] 1 2 3 4 5 6

> x <- x/54
> x
[1] 0.003703704 0.007407407 0.011111111 0.014814815 0.018518519 0.022222222
> round(x,2)        # round to
[1] 0.00 0.01 0.01 0.01 0.02 0.02
```

```
> ceiling(x)
> floor(x)
> trunc(x, ...)
```

2.2 Combine vectors

◆ R code

```
> M1 <- c(vec1, vec2)        # combine values into a vector
> M1
 [1]  1  3  5  7  9 11 13 15 17 19  2  4  6  8 10 12 14 16 18 20

> M1 <- cbind(vec1, vec2)        # combine by columns
> M1
      vec1 vec2
 [1,]    1    2
 [2,]    3    4
 [3,]    5    6
 [4,]    7    8
 [5,]    9   10
 [6,]   11   12
 [7,]   13   14
 [8,]   15   16
 [9,]   17   18
[10,]   19   20

> M1 <- rbind(vec1, vec2)        # combine by rows
> M1
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
vec1    1    3    5    7    9   11   13   15   17    19
vec2    2    4    6    8   10   12   14   16   18    20
```

2.3 Attributes of a vector

- `dim(x)` returns the dimension of x
- `length(x)` returns the length of x
- `t(x)` returns the transpose of x
- Note that when a vector is created, it has no dimensions.

◆ R code

```
> dim(vec1)
```

```
NULL

> length(vec1)
[1] 10

> vec7 <- t(vec1)
> vec7
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    3    5    7    9   11   13   15   17    19
> dim(vec7)
[1]  1 10

> vec7 <- t(t(vec1))
> dim(vec7)
[1] 10  1
> vec7
      [,1]
 [1,]    1
 [2,]    3
 [3,]    5
 [4,]    7
 [5,]    9
 [6,]   11
 [7,]   13
 [8,]   15
 [9,]   17
[10,]   19
```

2.4 Matrix Operations

◆ R code

```
# There are many ways to create a matrix. For example…
> Mij <- rbind(c(1, 2, 3), c(4, 5, 6))
> Mij
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> Mij <- cbind(c(1, 2, 3), c(4, 5, 6))
> Mij
     [,1] [,2]
[1,]    1    4
```

```
[2,]    2    5
[3,]    3    6
> Mij <- matrix(seq(1, 6), ncol = 2)
> Mij
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> Mij <- matrix(seq(1, 6), ncol = 3)
> Mij
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

# Identity matrix
> I5 <- diag(1, nrow = 5, ncol = 5)
> I5
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1

# Solve(a, b) : This generic function solves the equation a %*% x = b for x,
# where b can be either a vector or a matrix.

# inverse of x
> x <- matrix(c(1, 2, 3, 4), ncol = 2)
> x
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> solve(x)      # get inverse of x
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5

# solve ax = b
```

$$\begin{bmatrix} 1 & 2 & 4 \\ 2 & 4 & 6 \\ 4 & 6 & 8 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

```
> a <- matrix(c(1,2,4,2,4,6,4,6,8), ncol = 3)
> a
     [,1] [,2] [,3]
[1,]    1    2    4
[2,]    2    4    6
[3,]    4    6    8
> b <- matrix(c(1,1,-1), ncol = 1)
> b
     [,1]
[1,]    1
[2,]    1
[3,]   -1
> solve(a, b)
     [,1]
[1,] -2.0
[2,]  0.5
[3,]  0.5
```

3. Generate data from the distribution

   ◆ R code

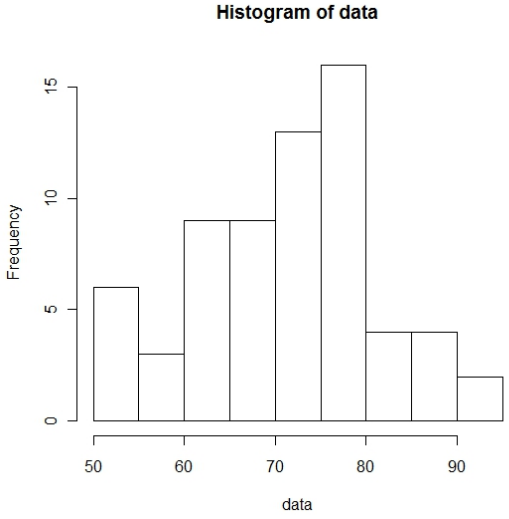
```
# n is number of data

rep(value, n)                       # repeat value n times
runif(n, min=0, max=1)              # uniform
rnorm(n, mean=0, sd=1)              # Gaussian (normal)
rexp(n, rate=1)                     # exponential
rgamma(n, shape, scale=1)           # gamma
rpois(n, lambda)                    # Poisson, "arrivals of default events"
rweibull(n, shape, scale=1)         # Weibull
rcauchy(n, location=0, scale=1)     # Cauchy, "fat tail phenomena"
rbeta(n, shape1, shape2)            # beta, "recovery rate modeling"
rt(n, df)                           # Student (t) "fat tail phenomena"
rchisq(n, df)                       # Chi-square
rbinom(n, size, prob)               # binomial "tree pricing"
rmultinom(n, size, prob)            # multinomial
rgeom(n, prob)                      # geometric
```

rhyper(nn, m, n, k)	# hyper geometric
rlogis(n, location=0, scale=1)	# logistic "transform the domain of variables"
rlnorm(n, meanlog=0, sdlog=1)	# lognormal "Geometric Brownian motion of equity price process"
rnbinom(n, size, prob)	# negative binomial

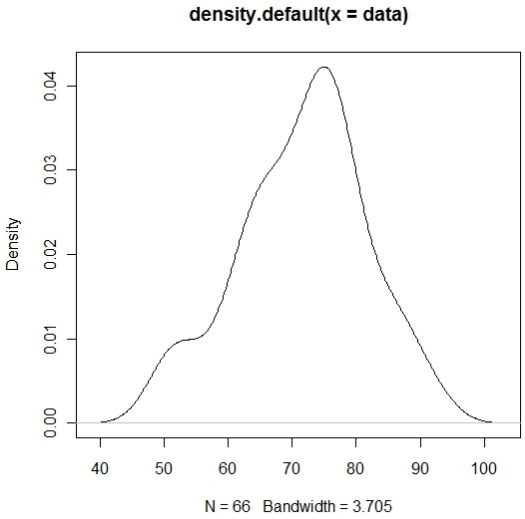
4. Plot

- R code

```
> data <-
c(50,50,52,52,54,54,60,60,60,61,63,63,63,64,65,65,65,65,66,66,67,67,67,68,70,70,70,72,72,72,72
,72,73,73,73,73,75,75,75,75,76,76,76,76,76,76,77,77,77,78,78,78,79,79,80,80,81,83,85,85,86,86,
87,88,91,92)
> hist(data)
```




```
> plot(density(data))
```



5. Control flow (loop)

if(cond)	expr
if(cond)	cons.expr else alt.expr
for(var in seq)	expr
while(cond)	expr
repeat expr	
break	
next	

◆ R code – Example 1 (Generate Fibonacci number)

```
rm(list=ls(all=TRUE));
x = c(0, 1);
for(i in 1: 10)
{
    #print(x);
    len = length(x);
    tmp = x[len]+x[len-1];
    x = c(x, tmp);
}
print(x);
```

```
> print(x);
 [1]  0  1  1  2  3  5  8  13  21  34  55  89
```

◆ R code – Example 2 (Tests of Hypothesis – t test)

For example, final scores of the Professor Kuan's Econometrics course last year are listed below. We want to test whether mean is equal to 70.

$$\mathrm{H}_0 : \mu_0 = 70$$
$$\mathrm{H}_1 : \mu_0 \neq 70$$

```
> data <-
c(50,50,52,52,54,54,60,60,60,61,63,63,63,64,65,65,65,65,66,66,67,67,67,68,70,70,70,72,72,
72,72,72,73,73,73,73,75,75,75,75,76,76,76,76,76,76,77,77,77,78,78,78,79,79,80,80,81,83,85
,85,86,86,87,88,91,92)
> length(data)
[1] 66

> t.test(data, mu = 70, conf.level = 0.95)
```

One Sample t-test

data: data

t = 1.3937, df = 65, p-value = 0.1681

alternative hypothesis: true mean is not equal to 70

95 percent confidence interval:

69.26529 74.12865

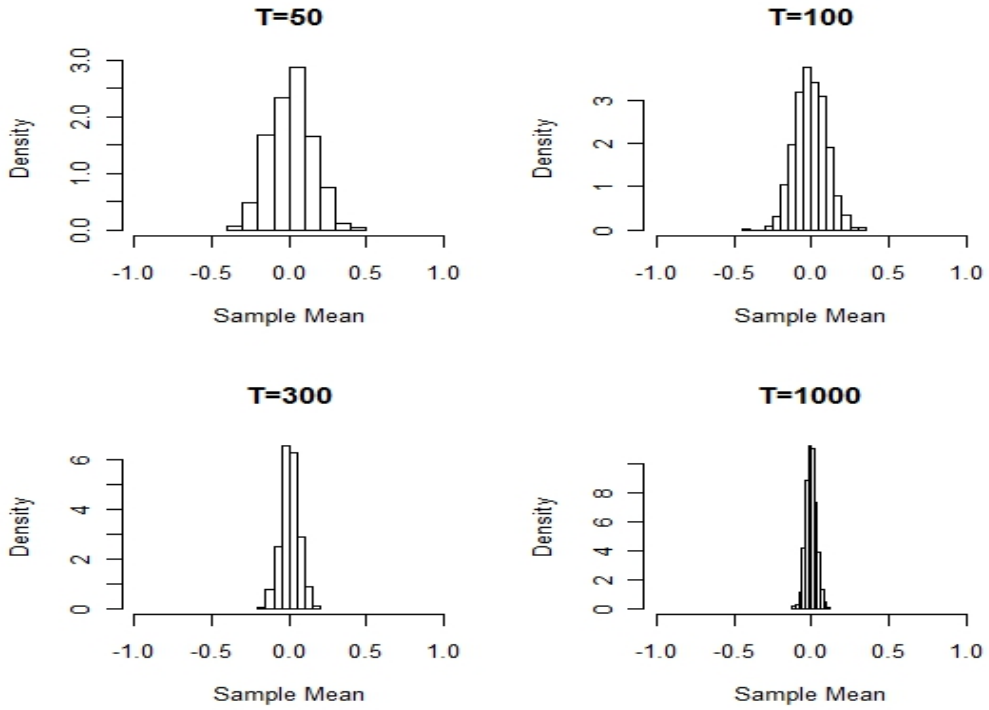
sample estimates:

mean of x

71.69697

◆ R code – Example 3 (Law of Large Number)

This example will demonstrate the law of large numbers theorem. Please generate random samples from the standard normal distribution for 1000 times with different sample size T. Let's say four cases here, T=50, T=100, T=300, and T=1000. For each different sample size T, please calculate the sample average each time and plot histogram of sample averages. You may observe the average of the results obtained from a large number of trials should be close to the expected value, and will tend to become closer as more trials are performed.




```
# t is sample size
# n is repetition time
rm(list=ls(all=TRUE));

fun_LLN<-function(t,n){
```

```
        x<-numeric(n)
        for(i in 1:n){
                x[i] <- mean(rnorm(t))
        }
        x
}

par(mfrow=c(2,2))
hist(fun_LLN(50,1000),xlim=range(-1,1),freq=FALSE,main='T=50',xlab='Sample Mean')
hist(fun_LLN(100,1000),xlim=range(-1,1),freq=FALSE,main='T=100',xlab='Sample Mean')
hist(fun_LLN(300,1000),xlim=range(-1,1),freq=FALSE,main='T=300',xlab='Sample Mean')
hist(fun_LLN(1000,1000),xlim=range(-1,1),freq=FALSE,main='T=1000',xlab='Sample Mean')
```

◆ R code – Example 4 (Simulate Chi-square distribution from normal random data)

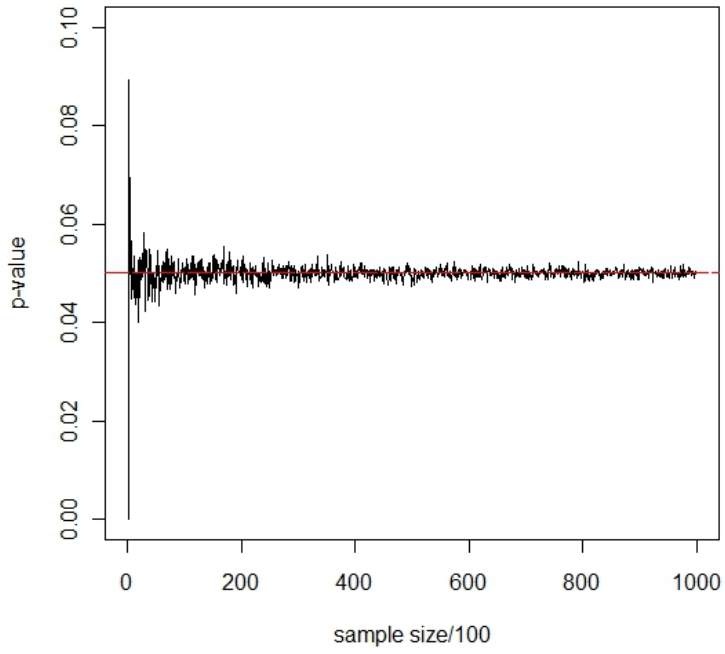
If X_1, …, X_k are independent, standard normal random variables, then the sum of their squares

$$SS = \sum_{i}^{k} X_i^2$$

is distributed according to the **chi-square distribution** with *k* degrees of freedom. This is usually denoted as

$SS \sim \chi^2(k)$.

The square of a standard normal random variable has a chi-squared distribution with one degree of freedom. An illustration is shown below.



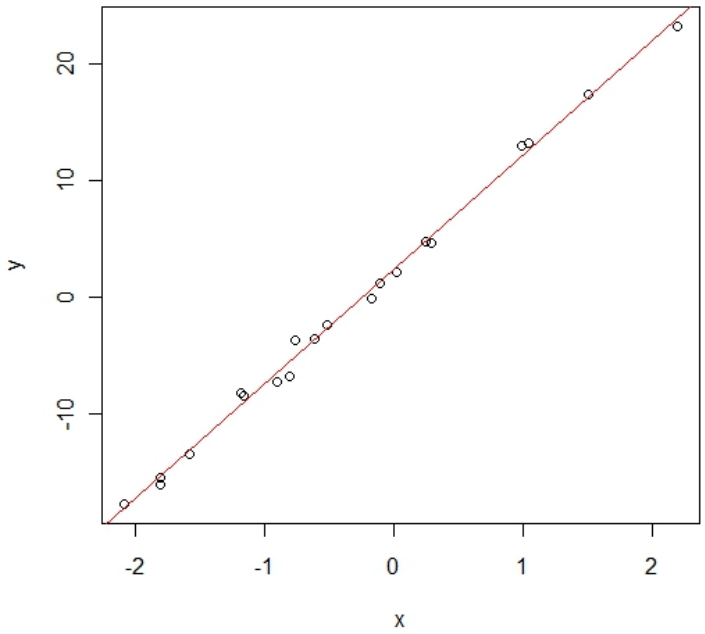
```
# Simulate Chi-square distribution from standard normal random samples

rm(list=ls(all=TRUE));

# x_chi    = chi_sq(1)
critical_value_chi <- qchisq(0.95, df=1)
count = 0    #set initial value = 0
fun_chi<-function(t){
      x_chi <- rnorm(t)^2;
      count = length( x_chi[x_chi >= critical_value_chi]);
      p_value_chi_sample = count/t
      p_value_chi_sample
}

#par(mfrow=c(2,2))

for (i in seq(from = 1, to = 100000, by = 100))
{
      tmp = fun_chi(i)
      if(i==1)
            x = tmp
      else
            x = c(x, tmp)

      x
}

plot(x, ylim=range(0,0.1), type = "l", ylab='p-value', xlab='sample size/100')
abline(a = 0.05, b = 0, col = "red", lty=5)
```

◆ R code – Example 5 (OLS)

- Use R build-in function lm()

```
# OLS
rm(list=ls(all=TRUE));
n = 20
x <- rnorm(n)
b1 <- 2
```

```
b2 <- 10
y <- b1 + b2*x + rnorm(n, 0, 1);

lm(y~x)
summary(lm(y~x))
b_hat_ols <- coef(lm(y~x))
plot(x, y)
abline(b_hat_ols[1], b_hat_ols[2], col = "red")
```




```
######### alternative way to do OLS ##########
```

- Calculate OLS coefficients by ourselves

```
b <- t(matrix(c(b1, b2), ncol = 2))
x <- matrix(c(rep(1, n), x), ncol = 2)
xx_inv <- solve((t(x)%*%x))
b_hat <- xx_inv%*%t(x)%*%y
```

◆ R code – Example 6 (Tests of Hypothesis – Type 1 error & Type 2 error)

Type 1 error means incorrect rejection of the null when the null is actually correct. **Type 2 error** means incorrect acceptance of the null when the null is false. In constructing tests, we want to control the probability of Type 1 error, which is also known as the **significance level**. One minus the probability of Type 2 error is known as **power,** which is the probability of correctly rejecting the null when the null is false. Given a significance level, test performance, such as power, may depend on several factors: sample size, standard deviation of data (σ), and the deviation from the null hypothesis (Δ).

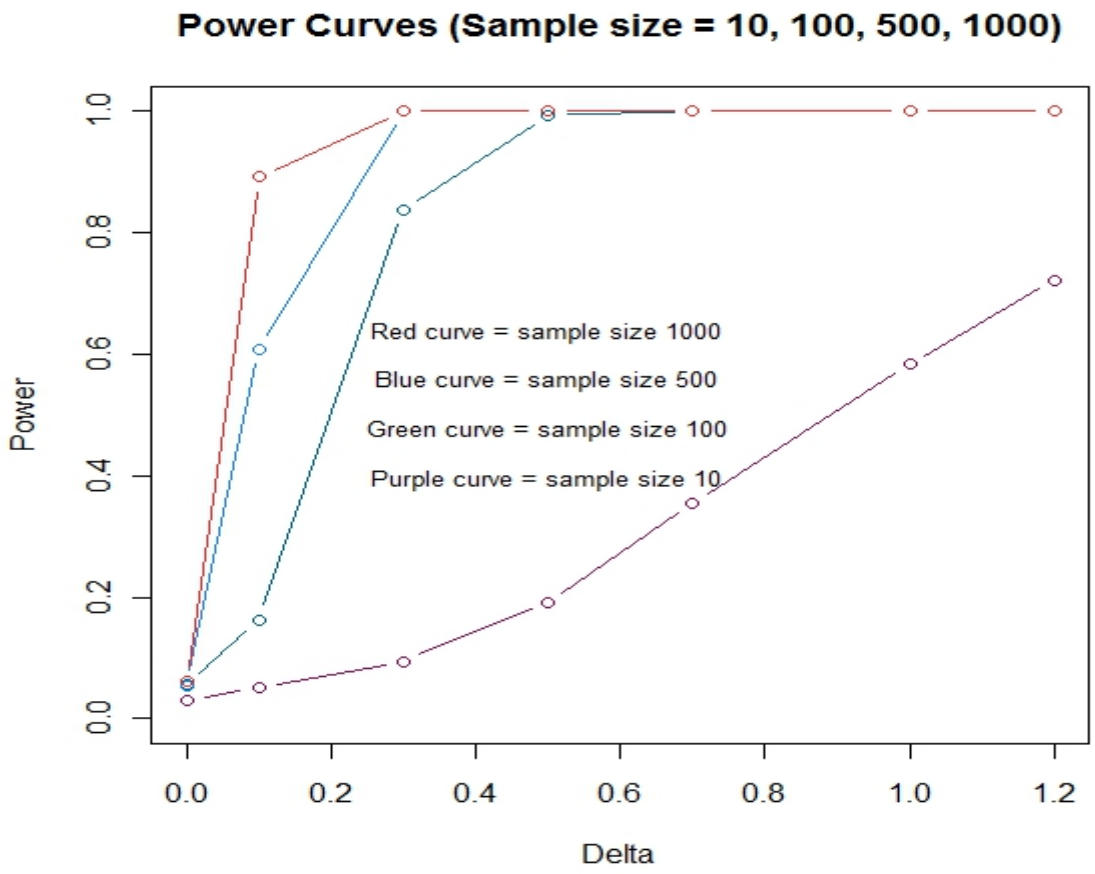
Consider the following DGP (data generation process) :

$$y_t = \beta_1 + (\beta_2 + \Delta) * x_{2t} + \beta_3 * x_{3t} + e_t$$
$$x_2 \sim N(0,1), \qquad x_3 \sim N(0,1)$$
$$where \qquad \beta_1 = 2, \qquad \beta_2 = 0.5, \qquad \beta_3 = 0.2$$
$$e_t \sim N(0, \sigma^2), \qquad \sigma = 1$$

Regressing y on x, we obtain OLS coefficients $\hat{\beta} = (\hat{\beta}_1, \hat{\beta}_2, \hat{\beta}_3)'$ and construct a t test to check whether $\hat{\beta}_2$ is "sufficiently close" to 0.5.

$$H_0 : \beta_2 = 0.5$$
$$H_1 : \beta_2 \neq 0.5$$
$$T_{\beta_2} = \frac{\hat{\beta}_2 - 0.5}{S.E.} \sim t(T-3)$$

Let $\Delta = (0, \quad 0.1, \quad 0.3, \quad 0.5, \quad 0.7, \quad 1, \quad 1.2)$. For each Δ, you can evaluate how power changes with different sample sizes, T=10, 100, 500, and 1000. For each sample size, please simulate the test at least 1000 times and evaluate the proportion of projection.



◆ R code

```
# change delta

rm(list=ls(all=TRUE));

ols_Hypo_type1 <- function(t, sigma, delta)
{
	x2 <- rnorm(t, 0, 1)
	x3 <- rnorm(t, 0, 1)
	b1 <- 2
	b2 <- 0.5 + delta
	b3 <- 0.2
	b <- t(matrix(c(b1, b2, b3), ncol = 3))
	x <- matrix(c(rep(1, t), x2, x3), ncol = 3)
	y = x %*% b + rnorm(t, 0, sigma)
	#summary(lm(y~ x2 + x3))
	c <- coef(summary(lm(y~ x2 + x3)))
	b_hat <- c[, 1]
	y_hat <- x %*% b_hat
	err <- sqrt(t(y-yhat)%*%(y-yhat)/(t-3)*diag(a)[2])
	t_stat <- (c[2, 1]- (0.5))/err
	#t_stat <- (c[2, 1])/err
	t_stat
}

exe <- function(t, n, delta_in)
{
	critical_value <- qt(0.975, t-3)
	test <- numeric(n)
	for(i in 1:n)
	{
		test[i] <- ols_Hypo_type1(t, 1, delta_in)
	}

	c <- length( test[abs(test) > critical_value] )

	print(c/n)
}

T <- c(10, 100, 500, 1000)
```

a<-solve(t(x)%*%x)

```
Del <-c(0, 0.1, 0.3, 0.5, 0.7, 1, 1.2)
y_axis <- seq(from = 0 , to = 1, by = 1/(length(Del)-1))
power<- numeric(length(Del))
plot(Del, y_axis, xlab="Delta",ylab="Power", main="Power Curves (Sample size = 10, 100,
500, 1000)", type = 'n', )


for(j in 1:length(T))
{

      print(j)
      for(i in 1:length(Del))
      {
            power[i] = exe(T[j], 1000, Del[i])
      }
      lines(Del, power, type = 'b', col =    colors()[120+5*(j-1)])
}

text(0.5, 0.5, "Red curve = sample size 1000\n
Blue curve = sample size 500\n
Green curve = sample size 100\n
Purple curve = sample size 10\n", cex = .8)
```

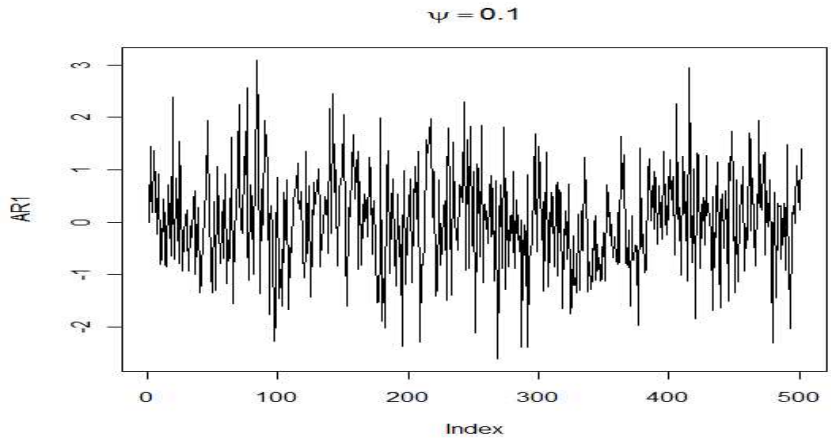
6. Data input and data output

read.table() : Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

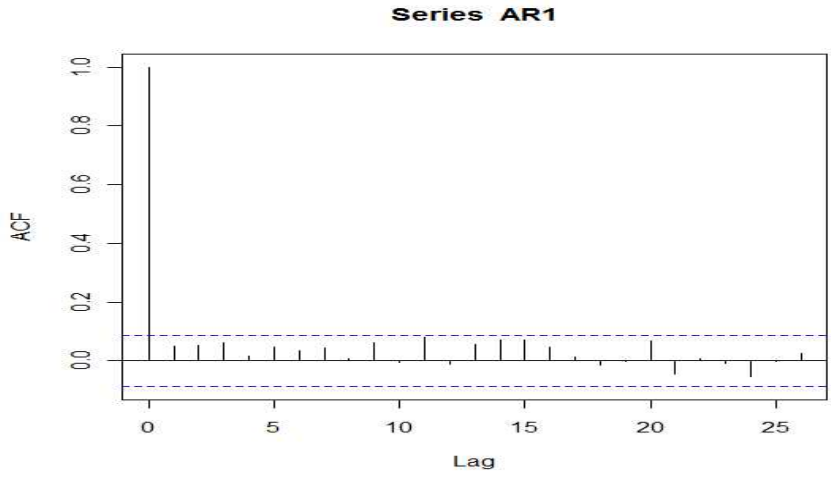
write.table() : write.table prints its required argument x (after converting it to a data frame if it is not one nor a matrix) to a file or connection.

◆ R code

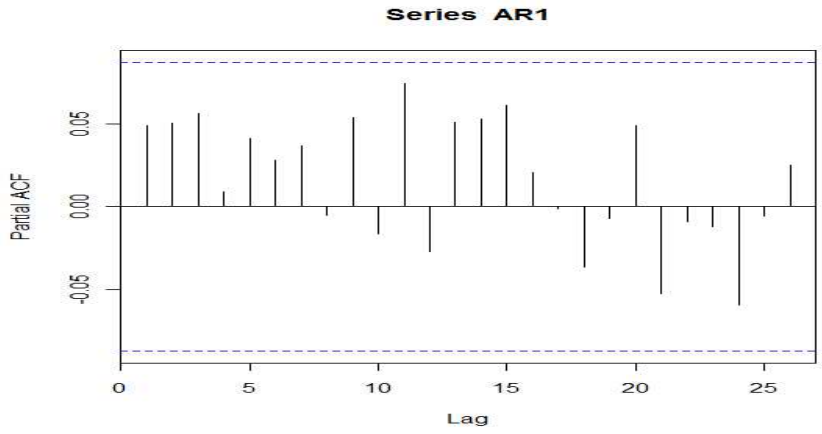
```
data<- read.table("C:/Users/oocomputer/Documents/Econometrics_Kuan/R lecture by
Yashin Hsiao/ar1_500.txt")
AR1 = data[,5]
plot(AR1, type = "l", ylab = "AR1", , main = expression(psi == 0.1))
```



acf(AR1)



pacf(AR1)



7. Exercise

i. (LLN) Generate random samples with numbers of T=50, T=100, T=300, and T =1000 form the following distributions for 1000 times, and compute their sample averages each time. Plot the histograms. Do your results obey the law of large numbers? Why?

(1) Chi-squared $\chi^2(1)$ distribution

(2) Student t(5) with zero mean
(3) Student t(1)

ii. (CLT) In this simulation, random sequences $\{x_t\}$ with sizes of T = 10, T = 50, T = 500, T= 1000 are sampled from a distribution for 1000 times and their normalized sample averages are computed each time as follows:

$$\frac{\sqrt{T}(\bar{x} - \mu)}{\sigma}$$

where x, μ, and σ are the sample average, mean, and standard deviation, respectively. Please plot the histograms of averages under different sizes of and explain the results. Moreover, there are 3 distributions we should consider in this exercise as follows.
(1) Student t (2) distribution with zero mean.
(2) Student t (4) distribution with zero mean.
(3) Lognormal distribution.

iii. The ratio of two chi-square random variables is F-distribution. Please generate random data from standard normal distribution and use those data to simulate F-distribution. Like example 4, you may verify your result by calculating p-value of F-distribution.

$$\frac{\chi_1^2 / d_1}{\chi_2^2 / d_2} \sim F(d_1, d_2)$$

iv. According to the example 6, you may try to change sigma and observe the power when sample size increases. (T = 50, 100, 200, 300, 500)
(1) Let Δ = 1 and $\sigma = (0.5, \quad 1, \quad 1.5, \quad 2, \quad 3, \quad 4, \quad 5, \quad 6, \quad 7, \quad 10)$. For each σ, you can evaluate how power changes with different sample size. Draw a power curve like example 6 and explain the result you observe.
(2) Please try to change the variance of x_2. Dose this change affect test performances and in what direction? Draw a power curve like example 6 and explain the result you observe.