Lecture 1: Basic R

Zhentao Shi

Jan 10, 2018

Basic R

Introduction

One cannot acquire a new programming language without investing numerous hours. R-Introduction is an official manual maintained by the R core team. It was the first document that I perused painstakingly when I began to learn R in 2005. After so many years, this is still the best starting point for you to have a taste.

This lecture quickly sketches some key points of the manual, while you should carefully go over R-Introduction after today's lecture.

Help System

The help system is the first thing we must learn for a new language. In R, if we know the exact name of a function and want to check its usage, we can either call `help(function_name)` or a single question mark `?function_name`. If we do not know the exact function name, we can instead use the double question mark `??key_words`. It will provide a list of related function names from a fuzzy search.

Example: ?seq, ??sequence

Vector

A *vector* is a collection of elements of the same type, say, integer, logical value, real number, complex number, characters or factor. Unlike C, R does not require explicit type declaration.

`<-` assigns the value on its right-hand side to a self-defined variable name on its left-hand side. `=` is an alternative for assignment, which I prefer.

`c()` combines two or more vectors into a long vector.

Binary arithmetic operations `+`, `-`, `*` and `\` are performed element by element. So are the binary logical operations `&` `|` `!=`.

Factor is a categorical number. *Character* is text.

Missing values in R is represented as `NA` (Not Available). When some operations are not allowed, say, `log(-1)`, R returns `NaN` (Not a Number).

Vector selection is specified in square bracket `a[ ]` by either positive integer or logical vector.

Example

Logical vector operation.

```
# logical vectors
logi_1 = c(T,T,F)
logi_2 = c(F,T,T)

logi_12 = logi_1 & logi_2
print(logi_12)
```

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

Array and Matrix

An array is a table of numbers.
A matrix is a 2-dimensional array.

- array arithmetic: element-by-element. Caution must be exercised in binary operations involving two objects of different length. This is error-prone.
- `%*%`, `solve`, `eigen`

Example

OLS estimation with one x regressor and a constant. Graduate textbook expresses the OLS in matrix form

$$\hat{\beta} = (X'X)^{-1}X'y.$$

To conduct OLS estimation in R, we literally translate the mathematical expression into code.

Step 1: We need data Y and X to run OLS. We simulate an artificial dataset.

```
# simulate data
rm(list = ls( ) )
set.seed(111) # can be removed to allow the result to change

# set the parameters
n = 100
b0 = matrix(1, nrow = 2 )

# generate the data
e = rnorm(n)
X = cbind( 1, rnorm(n) )
Y = X %*% b0 + e
```

Step 2: translate the formula to code

```
# OLS estimation
bhat = solve( t(X) %*% X, t(X)%*% Y )
```

Step 3 (additional): plot the regression graph with the scatter points and the regression line. Further compare the regression line (black) with the true coefficient line (red).

```
# plot
plot( y = Y, x = X[,2], xlab = "X", ylab = "Y", main = "regression")
abline( a= bhat[1], b = bhat[2])
abline( a = b0[1], b = b0[2], col = "red")
abline( h = 0, lty = 2)
abline( v = 0, lty = 2)
```

regression
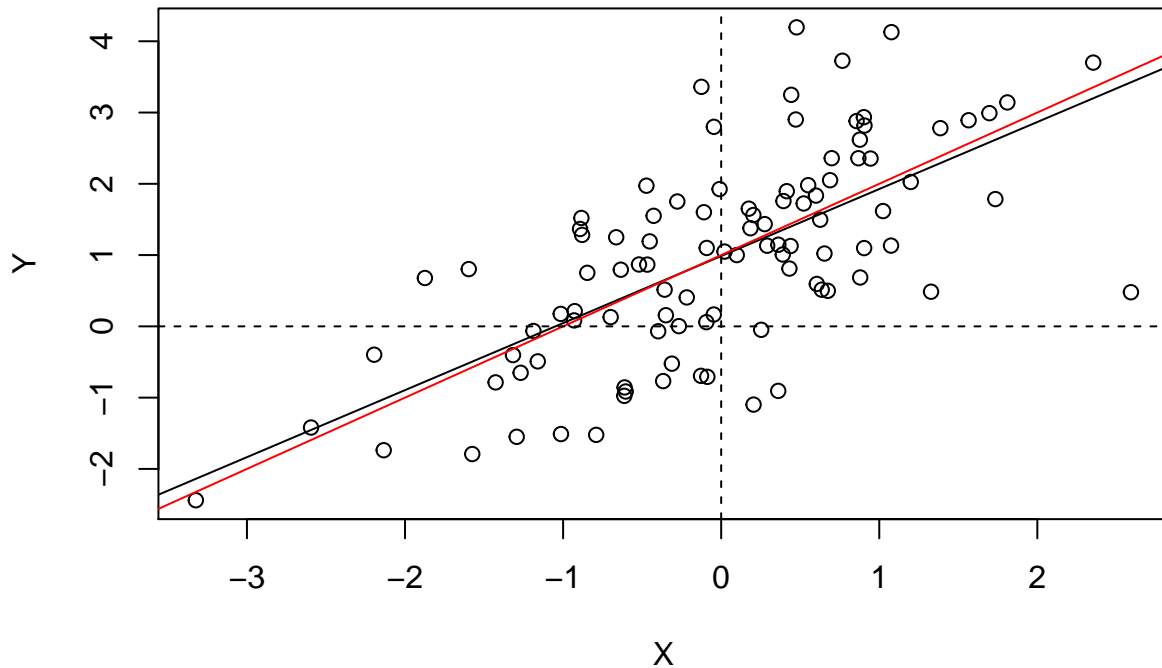
Step 4: In econometrics we are often interested in hypothesis testing. The t-statistic is widely used. To test the null $H_0 : \beta_2 = 0$, we compute the associated t-statistic. Again, this is a translation.

$$t = \frac{\hat{\beta}_2 - \beta_{02}}{\hat{\sigma}_{\hat{\beta}_2}} = \frac{\hat{\beta}_2 - \beta_{02}}{\sqrt{[(X'X)^{-1}\hat{\sigma}^2]_{22}}}.$$

where $[\cdot]_{22}$ is the (2,2)-element of a matrix.

```
# calculate the t-value
bhat2 = bhat[2] # the parameter we want to test
e_hat = Y - X %*% bhat
sigma_hat_square = sum(e_hat^2)/ (n-2)
Sigma_B = solve( t(X) %*% X  ) * sigma_hat_square
t_value_2 = ( bhat2 - b0[2]) / sqrt( Sigma_B[2,2] )
print(t_value_2)
```

```
## [1] -0.5615293
```

Package

An pure clean installation of R is small, but R has an extensive ecosystem of add-on packages. This is the unique treasure for R users, and other languages like Python or Matlab are not even close. Many of those packages are hosted on CRAN. A common practice today is that statisticians upload a package to CRAN after they write or publish a paper with a new statistical method. They promote their work via CRAN, and users have easy access to the state-of-the-art methods.

A package can be installed by `install.packages("package_name")` and invoked by `library(package_name)`.

[Applied Econometrics with R](#) by Christian Kleiber and Achim Zeileis is a useful book. It also has a companion package `AER` that contains popular econometric methods such as instrumental variable regression and robust variance.

Before we can knit the *.Rmd files in R-studio, we have to install several packages such as `knitr`.

Mixed Data Types

A vector only contains one type of elements. *list* is a basket for objects of various types. It can serve as a container when a procedure returns more than one useful object. For example, when we invoke `eigen`, we are interested in both eigenvalues and eigenvectors, which are stored into `$value` and `$vector`, respectively.

data.frame is a 2-dimensional table that stores the data, similar to a spreadsheet in Excel. A matrix is also a 2-dimensional table, but it only accommodates one type of elements. Real world data can be richer than one type. They can be a collection of integers, real numbers, characters and categorical numbers. Data frame is the best way to organize data of mixed type in R.

Example

This is a data set in a graduate-level econometrics textbook. We load the data into memory and display the first 6 records.

```
library(AER)
data("CreditCard")
head(CreditCard)
```

```
##   card reports      age income       share expenditure owner selfemp
## 1  yes       0 37.66667 4.5200 0.033269910  124.983300   yes      no
## 2  yes       0 33.25000 2.4200 0.005216942    9.854167    no      no
## 3  yes       0 33.66667 4.5000 0.004155556   15.000000   yes      no
## 4  yes       0 30.50000 2.5400 0.065213780  137.869200    no      no
## 5  yes       0 32.16667 9.7867 0.067050590  546.503300   yes      no
## 6  yes       0 23.25000 2.5000 0.044438400   91.996670    no      no
##   dependents months majorcards active
## 1          3     54          1     12
## 2          3     34          1     13
## 3          4     58          1      5
## 4          0     25          1      7
## 5          2     64          1      5
## 6          0     54          1      1
```

Input and Output

Raw data is often saved in ASCII file or Excel. I discourage the use of Excel spreadsheet in data analysis, because the underlying structure of a spreadsheet is too complicated for statistical software to read. I recommend the use of `csv` format, a plain ASCII file format.

`read.table()` or `read.csv()` imports data from a ASCII file into an R session. `write.table()` or `write.csv()` exports the data in an R session to a ASCII file.

Example

Besides loading a data file on the local hard disk, We can directly download data from internet. Here we show how to retrieve the daily stock price of *Hong Kong Exchange Limited* from *Yahoo Finance*, and save the dataset locally.

```
# Update in Jan 2018: the following link is no longer valid.
# Yahoo changes the format of the inquiry to forbid massive data downloading
# We need to figure out a new way
HEX = read.csv("http://ichart.finance.yahoo.com/table.csv?s=0388.HK")
print(head(HEX))
write.csv(HEX, file = "HEX.csv")
```

Statistics

R is a language created by statisticians. It has elegant built-in statistical functions. `p` (probability), `d` (density for a continuous random variable, or mass for a discrete random variable), `q` (quantile), `r` (random variable generator) are used ahead of the name of a probability distribution, such as `norm` (normal), `chisq` (χ^2), `t` (t), `weibull` (Weibull), `cauchy` (Cauchy), `binomial` (binomial), `pois` (Poisson), to name a few.

**Example**

This example illustrates the sampling error.

1. Plot the density of $\chi^2(3)$ over an equally spaced grid system `x_axis = seq(0.01, 15, by = 0.01)` (black line).
2. Generate 1000 observations for $\chi^2(3)$ distribution. Plot the kernel density, a nonparametric estimation of the density (red line).
3. Calculate the 95-th quantile and the empirical probability of observing a value greater than the 95-th quantile. In population, this value should be 5%. What is the number in this experiment?

```
set.seed(888)
x_axis = seq(0.01, 15, by = 0.01)

y = dchisq(x_axis, df = 3)
plot(y = y, x=x_axis, type = "l", xlab = "x", ylab = "density")
z = rchisq(1000, df = 3)
lines( density(z), col = "red")
```
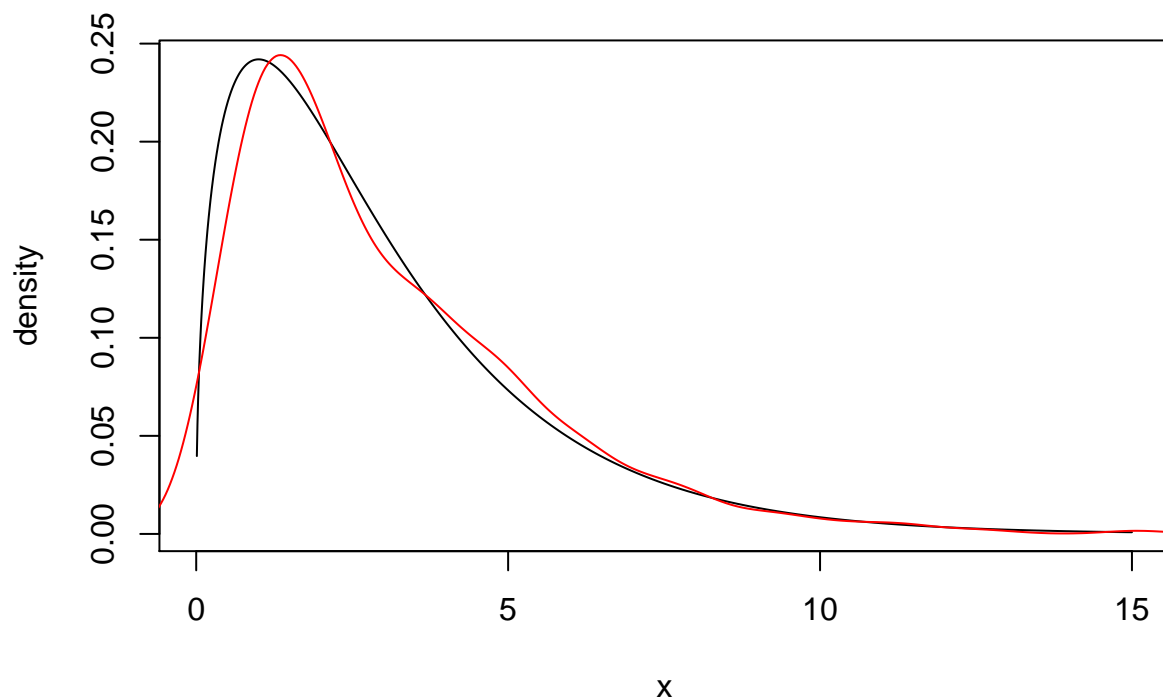
```
crit = qchisq(.95, df = 3)

mean( z > crit )
```

```
## [1] 0.047
```

User-defined Function

R has numerous built-in functions. However, in practice we will almost always have some
DIY functionality to be used repeatedly. It is highly recommended to encapsulate it into a user-defined
function. There are important advantages:

1. In the developing state, it allows us to focus on a small chunk of code. It cuts an overwhelmingly big project into manageable pieces.
2. A long script can have hundred or thousand of variables. Variables defined inside a function are local. They will not be mixed up with those outside of a function. Only the input and output have interaction with the outside world.
3. If a revision is necessary, We just need to change one place. We don't have to repeat the work in every place where it is invoked.

The format of a user-defined function in R is

```
function_name = function(input) {
  expressions
  return(output)
}
```

Example

Given a sample, if a central limit theorem is applicable, then we can calculate the 95% two-sided asymptotic confidence interval as

$$\left(\hat{\mu} - \frac{1.96}{\sqrt{n}} \hat{\sigma}, \hat{\mu} + \frac{1.96}{\sqrt{n}} \hat{\sigma} \right).$$

It is an easy job, but I do not think there is a built-in function in R to do this.

```
# construct confidence interval

CI = function(x){
  # x is a vector of random variables

  n = length(x)
  mu = mean(x)
  sig = sd(x)
  upper = mu + 1.96/sqrt(n) * sig
  lower = mu - 1.96/sqrt(n) * sig
  return( list( lower = lower, upper = upper) )
}
```

Flow Control

Flow control is common in all programming languages. `if` is used for choice, and `for` or `while` is used for loops.

Example

Calculate the empirical coverage probability of a Poisson distribution of degrees of freedom 2. We conduct this experiment for 1000 times.

```
Rep = 1000
sample_size = 100
capture = rep(0, Rep)


pts0 = Sys.time() # check time
for (i in 1:Rep){
  mu = 2
  x = rpois(sample_size, mu)
  bounds = CI(x)
  capture[i] = ( ( bounds$lower <= mu  ) & (mu <= bounds$upper) )
}
mean(capture) # empirical size
```

```
## [1] 0.938
```

```
pts1 = Sys.time() - pts0 # check time elapse
print(pts1)
```

```
## Time difference of 0.12834 secs
```

Statistical Model

Statistical models are formulated as `y~x`, where `y` on the left-hand side is the dependent variable, and `x` on the right-hand side is the explanatory variable. The built-in OLS function is `lm`. It is called by `lm(y~x, data = data_frame)`.

All built-in regression functions in R share the same structure. Only you understand one type of regression, it is easy to extend to other type of regressions.