STATS 406 F15: Lab 01

1 Section inforamtion

- Lab section: Thursday, 8:30-10:00 am (Yuan Zhang)
- Office hours: (Science Learning Center, Chemistry building, attend any)
  - Tuesday: 10:00a.m. – 11:30a.m. (Yuan Zhang)
  - Tuesday: 1:00p.m. – 2:30p.m. (Dao Nguyen)
  - Tuesday: 3:00p.m. – 4:30p.m. (Jun Guo)

2 Start coding in Rstudio

- New code:
  - By shortcut: (Ctrl + Shift + N).
  - By menu: File → New File → R Script.
- Save the code:
  - By shortcut: (Ctrl + S).
  - By menu: File → Save.
- Execute the code:
  - Line-by-line:
    1. Move the cursor to the line.
    2. Press (Ctrl + R) for Windows or (Command + Enter) for Mac.
  - Run script file(Recommended):
    1. Set proper working directory.

```
# Only for illustration, will not run
## Check the working directory
> getwd();
[1] "C:/Users/yzhanghf/Documents"
## Set proper working directory
> setwd('C:/Users/yzhanghf/Desktop');
```

    2. Source the file.

```
> source('Lab_1.r');
```

3 Variables and Functions

3.1 Variables

Two ways to assign a value to a variable

- (variable) = (value)
- (variable) <- (value)

```
> x = 6;
> y = x;
> print(y);
[1] 6

> x <- 8;
> y <- x^2 - 2*x + 1;
> print(y);
[1] 49
```

Do not use reserved keywords of R as variable names. Examples:

- c, function, print, return, ...
- sin, cos, sqrt, abs, ...
- plot, maplot, levelplot, hist, ...

Quiz: guess the output before running the script:

```
# What happens if you run this?
> c = 3;
> aa = c(1, 2, c);
> print(aa);
```

Run:

```
> rm(c);
```

afterwards to remove the user-specified definition for "c".

3.2 Define a user-specified function

Format: (function name) = function(argument list){function content; return(variable);}

- Simple functions can be defined in the in the main code file, while complicated functions are advised to be defined in separate files for clarity.

Example: create a file "plusthree.r" under working directory with the following content:

```
# Content of file: plusthree.r
plusthree = function(x, y=3){
  z = x + y;
  return(z);
}
```

Then in the console:

```
> source('plusthree.r');
> plusthree(3);
[1] 6
> plusthree(3, 6);
[1] 9
```

Remarks:

1. You can also run the content of plusthree.r instead of sourcing it in the console.
2. Assign *default values* to variables while defining the argument list.
3. In STATS 406, R does not pass variables by reference, so always have **return** by the end.

4 Vectors and Matrices

4.1 Create a vector

Basic ways to create vectors:

- **c()**, example **a=c(1, 2)**.
- **rep()**, example **a=rep(1, 3)**, creates an all-one vector of length 3.
- **seq()**, example **a=seq(from=0, to=2, by=0.1)** or **a=seq(0, 2, 0.1)**(not recommended). The reason that we advise writting "from", "to" and especially "by" in **seq** is **clarity**. **Quiz: what does this give you, "0,3,6" or "0,2,4,6"?**

  ```
  # Not recommended way of writing
  > print(seq(0, 6, 3));
  ```

  An alternative to "by" is "length.out".

  ```
  > print(seq(from=0, to=6, length.out=7));
  [1] 0 1 2 3 4 5 6
  ```

- Integer sequences, example **a=1:4;** equivalent to **a=c(1, 2, 3, 4);** This method is originally designed to facilitate **for** loops, but is now more widely applied.
- By random number generators. Examples: **a=rnorm(1000)**, **a=runif(100, -0.5, 0.5)** and so on. A few important notes:
  - **Always save random seed if any part of your code involves random number generation to allow others repeat your experiment and verify your results!**

    ```
    > set.seed(2015); # Use your lucky number in exams, if you have one.
    ```

  - **runif(100)** and **rnorm(100)** are OK. **runif(100, -1, 1)** is OK. But **rnorm(100, 1, 3)** should be replaced by **rnorm(100, mean=1, sd=3)**, especially "sd=" should never be omitted in general cases.
    - For all other distributions, explicitly write all parameter names.

4.2 Vector operations and manupulations

In R, ordinary operations on vectors are **element-wise**.

```
> a1 = c(1, 2, 3); a2 = c(1, 0, -1);
> print(a1*a2);
[1] 1 0 -3
```

Operations between vectors of different lengths is ...

1. allowed, if one's length is multiple to the other's

```
> a1 = c(1, 2, 3); a2 = c(1, 0, -1);
> a3 = c(-1, 0, 1, a2);
# Element-wise product of vectors with different lengths
> print(a1*a3);
[1] -1  0  3  1  0 -3
```

2. but should be avoided, to reduce bugs that do not trigger error messages.
3. The only exception is an operation between a vector and a scalar.

```
> a1 = c(1, 2, 3);
> print(a1+3);
[1] 4 5 6
```

Such operations are fully acceptable and widely-used. In R, scalars are treated as length-one vectors:

```
> a=1;
> is.vector(a);
[1] TRUE
> length(a);
[1] 1
```

Notice: an operation alone does not change the vector itself.

```
> a1 = c(1, 2, 3);
> append(a1, 3);
[1] 1 2 3 3
> print(a1);
[1] 1 2 3

# If you want to permanently append 3 to a1, then use ''=''.
> a1 = append(a1, 3);
> print(a1);
[1] 1 2 3 3
```

Two basic ways to extract subvectors:

1. By integer indexing: **a1[2:3]**, **a1[c(1, 3)]** and so on.
2. By logical indexing: **a1[c(TRUE, FALSE, FALSE)]**. Accessing by logical expressions are essentially the same.

```
> a1 = c(1, 2, 3); a2 = c(1, 0, -1);
> print( a1[a2<0.5] );
[1] 2 3

> indexvector = a2>0.5;
> print(indexvector);
[1] TRUE FALSE FALSE
> print( a1[indexvector] );
[1] 1
```

Quiz(seen in lecture): what will be the output?

```
> a1=c(1, 2, 3);
> indexvector=c(TRUE, FALSE); # Incorrectly short in length.
# Only for illustration, avoid in practical programming
> print( a1[indexvector] );
```

This is an example of a potential mistake that does not trigger error message.

Many functions defined for scalars accept vectors as input:

```
> a=seq(from=0, to=pi, by=0.25*pi);
> print(round(sin(a), 3));
[1] 0.000 0.707 1.000 0.707 0.000
```

The function **sin** was applied element-wise.
Many other useful operations, including **sort()**, **order()**, **min()**, **median()** and so on. Learn their usage from their manuals.

```
# Toggle documentation, press ``q'' to quit when finished.
> ?sort
```

4.3 Create a matrix

Useful commands:

- **matrix()**, first specify #rows, then #columns. Example:

```
> a = matrix(1:6, c(2, 3));
> print(a);
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

- **array()**, similar to **matrix**, but can easily generate matrices of identical elements:

```
> a = array(0, c(2,2));
> print(a);
     [,1] [,2]
[1,]    0    0
[2,]    0    0
```

Quiz: how to generate an all-one matrix of 6×6 dimensions with "matrix()"?

- **cbind()** and **rbind()**, combine vectors and/or matrices of proper dimensions. Example:

```
> a1 = 1:3; a2 = 2:4;
> a12 = cbind(a1, a2);
> print(a12);
     a1 a2
[1,]  1  2
[2,]  2  3
[3,]  3  4
> print(cbind(a12, a1));
     a1 a2 a1
[1,]  1  2  1
[2,]  2  3  2
[3,]  3  4  3
```

4.4 Matrix operations and manipulations

Like vectors, operations between scalars applied to matrices become element-wise.
The symbol for matrix product is **%*%**, example: **A%*%B**.
Two important remarks:

1. Non-degenerate submatrix views produce matrix objects, but degenerate submatrix views produce vectors. To prevent this, use **drop=FALSE**.

```
> A = array(1:9, c(3,3));
> is.matrix(A[1:2, 1:2]);
[1] TRUE
> is.matrix(A[, 1]);
[1] FALSE
> is.matrix(A[, 1, drop=FALSE]);
[1] TRUE
```

2. R treat all vectors as column vectors, but it does not differentiate between row- and column- vectors.

```
# Example 1
> a1 = 1:3;
> a2 = t(a1); # transpose
> print(a2);
          [,1] [,2] [,3]
     [1,]    1    2    3
> is.vector(a2); # After transpose, a2 becomes a matrix.
[1] FALSE

# Example 2
> A = diag(1:3);
> print( a1 %*% A %*% a1 );
          [,1]
     [1,]   36
```

No need to transpose the first "a1" in the sandwich product.

4.5 Data frames

Data frame is an extension to matrices, allowing columns to take different types.

```
> A1 = (1:5)/10; A2 = letters[1:5]; A3 = round(sin(1:5), 3);
# Numerical objects are converted to characters when concatenating A1 to A3.
> A = cbind(A1, A2, A3);
> print(A);
          A1    A2  A3
     [1,] "0.1" "a" "0.841"
     [2,] "0.2" "b" "0.909"
     [3,] "0.3" "c" "0.141"
     [4,] "0.4" "d" "-0.757"
     [5,] "0.5" "e" "-0.959"
# Data frames keep numerical variables numerical, and convert characters to factors.
> Adataframe = data.frame(A1, A2, A3);
> print(Adataframe);
    A1 A2     A3
1 0.1  a  0.841
2 0.2  b  0.909
3 0.3  c  0.141
4 0.4  d -0.757
5 0.5  e -0.959
```

Variables in a data frame can be conveniently referred by their names, like **A$A2**.
Setting input type to data.frame is a requirement in many packages for regression and other purposes.