Review of R and R Loop

(Jing Li, Miami University)

Big Picture

1. R is a free open-source software downloadable at `https://www.r-project.org/`; Online courses about R can be found at `https://stats.idre.ucla.edu/r/`
2. R is case sensitive
3. R offers a variety of built-in functions (commands), along with timely-updated expert-defined packages
4. R functions can be nested
5. R handles different objects (data series, matrix, list...) easily
6. R graph function is powerful

R is case sensitive

```
> A = 2
> b = 3
> a+b
Error: object 'a' not found
> A+b
[1] 5
> A==3
[1] FALSE
```

Single equal sign does assignment; double equal sign does evaluation.

R functions

```
> sqrt(A)+log(3)
[1] 2.512826
> (A<5)
[1] TRUE
> (A<5)+1
[1] 2
> (A>6)
[1] FALSE
> (A>6)-1
[1] -1
```

1. `sqrt` computes the square root; `log` computes the natural log.
2. All functions require inputs that you put inside the parentheses. If there are several inputs, they are separated by comma.
3. Parentheses alone do logic evaluation, and return Boolean values of 1 (True) and 0 (False)

R functions can be nested

Basically, we can do composite functions

$$f(g(x))$$

That is, the output of $g(x)$ is the input for $f()$. For instance, below we compute

$$\sqrt{|-4|}$$

```
> sqrt(abs(-4))
[1] 2
```

The absolute value function `abs` is substituted into the square root function `sqrt`. There can be multiple level of nesting.

c function and indexing

```
> 4:6
[1] 4 5 6
> c(1,4:6)
[1] 1 4 5 6
> avector = seq(2,10,2)
> avector
[1]  2  4  6  8 10
> avector[3]
[1] 6
> which(avector==6)
[1] 3
> avector[c(1,4:6)]
[1]  2  8 10 NA
```

Remarks: (1) The colon can create a sequence of integers, and the `seq` function is more flexible; (2) the concatenate function c can combine things together; (3) bracket is used for indexing; (4) missing value is NA

Matrix

```
> a = matrix(c(1,2,3,4), nrow=2, byrow=T)
> a
[,1] [,2]
[1,]    1    2
[2,]    3    4
> a[1,2]
[1] 2
> a[,2]
[1] 2 4
> a[1,]
[1] 1 2
> a[,1]+a[,2]
[1] 3 7
> a[,1]*a[,2]
[1]  2 12
```

Matrix operation

$\%*\%$ does matrix multiplication; while $*$ does element-by-element multiplication

```
> a = matrix(c(1,2,3,4), nrow=2, byrow=T)
> a[,1]*a[,2]
[1]  2 12
> a[,1]%*%a[,2]
[,1]
[1,]   14
> t(a[,1])%*%a[,2]
[,1]
[1,]   14
> a[,1]%*%t(a[,2])
[,1] [,2]
[1,]    2    4
[2,]    6   12
```

R plot

```
> x = seq(1,10)
> set.seed(12345)
> u = rnorm(10)
> y = x + u
> cbind(x,u,y)[1:5,]
     x          u        y
[1,] 1  0.5855288 1.585529
[2,] 2  0.7094660 2.709466
[3,] 3 -0.1093033 2.890697
[4,] 4 -0.4534972 3.546503
[5,] 5  0.6058875 5.605887
> plot(x,y)
> abline(lm(y~x))
```

The last two functions produce a scatter plot of *y* against *x* with the OLS line superimposed

Want to know more about the plot function?

```
? plot
```

Even better, google “R, plot” or chatgpt “please explains R plot function”

R Loop—Doing Repetitive Jobs Easily

Example 1

Suppose there is a crazy bank that offers 100% annual interest rate. After one year, one dollar deposit becomes

$$\left(1+\frac{1}{1}\right)^1 = 2.$$

Or, the interest can be paid twice a year (50% each time). Suppose the interest is not withdrawn, then after one year, the money becomes

$$\left(1+\frac{1}{2}\right)\left(1+\frac{1}{2}\right) = \left(1+\frac{1}{2}\right)^2.$$

Even better, how about the interest is earned every quarter? We have

$$\left(1+\frac{1}{4}\right)^4.$$

In the limit, if the interest is <u>continuously compounded</u>, we have

$$\lim_{k\to\infty}\left(1+\frac{1}{k}\right)^k = e.$$

A repetitive job

```
> (1+1/1)^1
[1] 2
> (1+1/2)^2
[1] 2.25
> (1+1/4)^4
[1] 2.441406
> (1+1/1000)^1000
[1] 2.716924
```

We see that as the number of compounding rises, the result converges to the true value of $e = 2.71828...$

A more efficient way of using R while loop

We are doing some calculation repeatedly, and that is when a loop becomes handy

```
k = 1
while (k < 1001) {
cat("-------------------------", "\n")
cat("k is ", k, "\n")
cat("Compounded Interest is ", (1+1/k)^k, "\n")
k = k*10
}
```

At beginning, $k = 1$. Then k is multiplied by 10 in each iteration, as long as k is less than 1001. In other words, the while loop does not stop until $k = 10000$. For each k the compounded interest $\left(1+\frac{1}{k}\right)^k$ is displayed.

While loop results

```
--------------------------
k is  1
Compounded Interest is  2
--------------------------
k is  10
Compounded Interest is  2.593742
--------------------------
k is  100
Compounded Interest is  2.704814
--------------------------
k is  1000
Compounded Interest is  2.716924
```

Exercises

1. what if we drop $k = k * 10$
2. what if we replace $k = k * 10$ with $k = k + 10$

Example 2

Hopefully, this simulation will amaze you

```
set.seed(12345)
k = 10
while (k < 100000001) {
point = cbind(runif(k, min = -1, max = 1),runif(k, min = -1, max = 1))
inside_circle = sum(((point[,1]^2+point[,2]^2)<1))
cat("--------------------------", "\n")
cat("Total number of points is ", k, "\n")
cat("magical number is ", inside_circle/k*4, "\n")
k = k*10
}
```

Remarks

1. `set.seed` function sets the seed number for the random number generator
2. while loop won't stop until $k \geq 100000001$. At beginning $k = 10$
3. `runif(k, min = -1, max = 1)` generates k random numbers that follow uniform distribution between -1 and 1. Those numbers are the x-coordinates and y-coordinates of n points. Note we use comma to separate function inputs.
4. `sum` function counts the number of points that lie inside a circle with radius of one. That is
$$\sqrt{x^2+y^2} < 1 \Rightarrow x^2+y^2 < 1$$
5. We compute a magical number, which is the ratio of number of points inside the circle over the total number of points times 4
6. After each iteration, k is multiplied by 10

Results

Can you see what we are getting? Why does this work?

```
--------------------------
Total number of points is  10
magical number is  2.4
--------------------------
Total number of points is  100
magical number is  3.12
--------------------------
Total number of points is  1000
magical number is  3.132
...
...
--------------------------
Total number of points is  1e+08
magical number is  3.141403
```

Exercise

You may find this line is hard to understand

```
inside_circle = sum(((point[,1]^2+point[,2]^2)<1))
```

Instead, please write down the R code that does a while loop to count the number of points inside the circle. More explicitly, for each point we evaluate whether its distance from the origin is less than one. If it is, the point counts! We do this repeatedly for all points