5
Dùng R cho các phép tính đơn giản và ma trận

Một trong những lợi thế của R là có thể sử dụng như một … máy tính cầm tay. Thật ra, hơn thế nữa, R có thể sử dụng cho các phép tính ma trận và lập chương. Trong chương này tôi chỉ trình bày một số phép tính đơn giản mà học sinh hay sinh viên có thể sử dụng lập tức trong khi đọc những dòng chữ này.

5.1 Tính toán đơn giản

Cộng hai số hay nhiều số với nhau:
```
> 15+2997
[1] 3012
```

Cộng và trừ:
```
> 15+2997-9768
[1] -6756
```

Nhân và chia
```
> -27*12/21
[1] -15.42857
```

Số lũy thừa: $(25 - 5)^3$
```
> (25 - 5)^3
[1] 8000
```

Căn số bậc hai: $\sqrt{10}$
```
> sqrt(10)
[1] 3.162278
```

Số pi (π)
```
> pi
[1] 3.141593
> 2+3*pi
[1] 11.42478
```

Logarit: $\log_e$
```
> log(10)
[1] 2.302585
```

Logarit: $\log_{10}$
```
> log10(100)
[1] 2
```

Số mũ: $e^{2.7689}$
```
> exp(2.7689)
[1] 15.94109

> log10(2+3*pi)
[1] 1.057848
```

Hàm số lượng giác
```
> cos(pi)
[1] -1
```

Vector
```
> x <- c(2,3,1,5,4,6,7,6,8)
> x
[1] 2 3 1 5 4 6 7 6 8

> sum(x)
[1] 42

> x*2
```

```
> exp(x/10)
[1] 1.221403 1.349859 1.105171 1.648
1.491825 1.822119 2.013753 1.822119
[9] 2.225541

> exp(cos(x/10))
[1] 2.664634 2.599545 2.704736 2.405
2.511954 2.282647 2.148655 2.282647
[9] 2.007132
```

`[1]  4  6  2 10  8 12 14 12 16`	

Tính tổng bình phương (sum of squares): $1^2 + 2^2 + 3^2 + 4^2 + 5^2 = ?$
`> x <- c(1,2,3,4,5)`
`> sum(x^2)`
`[1] 55`
	Tính tổng bình phương điều chỉnh (adjusted sum of squares): $\sum_{i=1}^{n}(x_i - \bar{x})^2 = ?$
	`> x <- c(1,2,3,4,5)`
	`> sum((x-mean(x))^2)`
	`[1] 10`
	
	Trong công thức trên `mean(x)` là số trung bình của vector `x`.

Tính sai số bình phương (mean square): $\sum_{i=1}^{n}(x_i - \bar{x})^2 / n = ?$
`> x <- c(1,2,3,4,5)`
`> sum((x-mean(x))^2)/length(x)`
`[1] 2`
 	
Trong công thức trên, `length(x)` có nghĩa là tổng số phần tử (elements) trong vector `x`.
	Tính phương sai (variance) và độ lệch chuẩn (standard deviation):
	Phương sai: $s^2 = \sum_{i=1}^{n}(x_i - \bar{x})^2 / (n-1) = ?$
	`> x <- c(1,2,3,4,5)`
	`> var(x)`
	`[1] 2.5`
	Độ lệch chuẩn: $\sqrt{s^2}$:
	`> sd(x)`
	`[1] 1.581139`

5.2 Số liệu về ngày tháng

Trong phân tích thống kê, các số liệu ngày tháng có khi là một vấn đề nan giải, vì có rất nhiều cách để mô tả các dữ liệu này. Chẳng hạn như 01/02/2003, có khi người ta viết 1/2/2003, 01/02/03, 01FEB2003, 2003-02-01, v.v… Thật ra, có một qui luật chuẩn để viết số liệu ngày tháng là tiêu chuẩn ISO 8601 (nhưng rất ít ai tuân theo!) Theo qui luật này, chúng ta viết:

2003-02-01

Lí do đằng sau cách viết này là chúng ta viết số với đơn vị lớn nhất trước, rồi dần dần đến đơn vị nhỏ nhất. Chẳng hạn như với số "123" thì chúng ta biết ngay rằng "một trăm hai mươi ba": bắt đầu là hàng trăm, rồi đến hàng chục, v.v… Và đó cũng là cách viết ngày tháng chuẩn của R.

```
> date1 <- as.Date("01/02/06", format="%d/%m/%y")
> date2 <- as.Date("06/03/01", format="%y/%m/%d")
```

Chú ý chúng ta nhập hai số liệu khác nhau về thứ tự ngày tháng năm, nhưng chúng ta cũng cho biết cụ thể cách đọc bằng `%d` (ngày), `%m` (tháng), và `%y` (năm). Chúng ta có thể tính số ngày giữa hai thời điểm:

```
> days <- date2-date1
> days
Time difference of 28 days
```

Chúng ta cũng có thể tạo một dãy số liệu ngày tháng như sau:

```
> seq(as.Date("2005-01-01"), as.Date("2005-12-31"), by="month")

[1]  "2005-01-01" "2005-02-01" "2005-03-01" "2005-04-01" "2005-05-01"
[6]  "2005-06-01" "2005-07-01" "2005-08-01" "2005-09-01" "2005-10-01"
[11] "2005-11-01" "2005-12-01"

> seq(as.Date("2005-01-01"), as.Date("2005-12-31"), by="2 weeks")

[1]  "2005-01-01" "2005-01-15" "2005-01-29" "2005-02-12" "2005-02-26"
[6]  "2005-03-12" "2005-03-26" "2005-04-09" "2005-04-23" "2005-05-07"
[11] "2005-05-21" "2005-06-04" "2005-06-18" "2005-07-02" "2005-07-16"
[16] "2005-07-30" "2005-08-13" "2005-08-27" "2005-09-10" "2005-09-24"
[21] "2005-10-08" "2005-10-22" "2005-11-05" "2005-11-19" "2005-12-03"
[26] "2005-12-17" "2005-12-31"
```

5.3 Tạo dãy số bằng hàm `seq`, `rep` và `gl`

R còn có công dụng tạo ra những dãy số rất tiện cho việc mô phỏng và thiết kế thí nghiệm. Những hàm thông thường cho dãy số là `seq` (sequence), `rep` (repetition) và `gl` (generating levels):

Áp dụng `seq`

- Tạo ra một vector số từ 1 đến 12:

```
> x <- (1:12)
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12

> seq(12)
[1]  1  2  3  4  5  6  7  8  9 10 11 12
```

- Tạo ra một vector số từ 12 đến 5:

```
> x <- (12:5)
> x
[1] 12 11 10  9  8  7  6  5

> seq(12,7)
[1] 12 11 10  9  8  7
```

Công thức chung của hàm `seq` là `seq(from, to, by= )` hay `seq(from, to, length.out= ).` Cách sử dụng sẽ được minh hoạ bằng vài ví dụ sau đây:

- Tạo ra một vector số từ 4 đến 6 với khoảng cách bằng 0.25:

```
> seq(4, 6, 0.25)
[1] 4.00 4.25 4.50 4.75 5.00 5.25 5.50 5.75 6.00
```

- Tạo ra một vector 10 số, với số nhỏ nhất là 2 và số lớn nhất là 15

```
> seq(length=10, from=2, to=15)
 [1]  2.000000  3.444444  4.888889  6.333333  7.777778  9.222222
10.666667 12.111111 13.555556 15.000000
```

Áp dụng `rep`

Công thức của hàm `rep` là `rep(x, times, ...)`, trong đó, *x* là một biến số và `times` là số lần lặp lại. Ví dụ:

- Tạo ra số 10, 3 lần:

```
> rep(10, 3)
[1] 10 10 10
```

- Tạo ra số 1 đến 4, 3 lần:

```
> rep(c(1:4), 3)
 [1] 1 2 3 4 1 2 3 4 1 2 3 4
```

- Tạo ra số 1.2, 2.7, 4.8, 5 lần:

```
> rep(c(1.2, 2.7, 4.8), 5)
 [1] 1.2 2.7 4.8 1.2 2.7 4.8 1.2 2.7 4.8 1.2 2.7 4.8 1.2 2.7 4.8
```

- Tạo ra số 1.2, 2.7, 4.8, 5 lần:

```
> rep(c(1.2, 2.7, 4.8), 5)
 [1] 1.2 2.7 4.8 1.2 2.7 4.8 1.2 2.7 4.8 1.2 2.7 4.8 1.2 2.7 4.8
```

Áp dụng `gl`

`gl` được áp dụng để tạo ra một biến thứ bậc (categorical variable), tức biến không để tính toán, mà là đếm. Công thức chung của hàm `gl` là `gl(n, k, length = n*k, labels = 1:n, ordered = FALSE)` và cách sử dụng sẽ được minh hoạ bằng vài ví dụ sau đây:

- Tạo ra biến gồm bậc 1 và 2; mỗi bậc được lặp lại 8 lần:

```
> gl(2, 8)
 [1] 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
Levels: 1 2
```

Hay một biến gồm bậc 1, 2 và 3; mỗi bậc được lặp lại 5 lần:

```
> gl(3, 5)
[1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels: 1 2 3
```

- Tạo ra biến gồm bậc 1 và 2; mỗi bậc được lặp lại 10 lần (do đó length=20):

```
> gl(2, 10, length=20)
```

```
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
Levels: 1 2
```

Hay:

```
> gl(2, 2, length=20)
[1] 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2
Levels: 1 2
```

- Cho thêm kí hiệu:

```
> gl(2, 5, label=c("C", "T"))
 [1] C C C C C T T T T T
Levels: C T
```

- Tạo một biến gồm 4 bậc 1, 2, 3, 4. Mỗi bậc lặp lại 2 lần.

```
> rep(1:4, c(2,2,2,2))
[1] 1 1 2 2 3 3 4 4
```

Cũng tương đương với:

```
> rep(1:4, each = 2)
[1] 1 1 2 2 3 3 4 4
```

- Với ngày giờ tháng:

```
> x <- .leap.seconds[1:3]
> rep(x, 2)
[1] "1972-06-30 17:00:00 Pacific Standard Time" "1972-12-31 16:00:00
Pacific Standard Time"
[3] "1973-12-31 16:00:00 Pacific Standard Time" "1972-06-30 17:00:00
Pacific Standard Time"
[5] "1972-12-31 16:00:00 Pacific Standard Time" "1973-12-31 16:00:00
Pacific Standard Time"

> rep(as.POSIXlt(x), rep(2, 3))
[1] "1972-06-30 17:00:00 Pacific Standard Time" "1972-06-30 17:00:00
Pacific Standard Time"
[3] "1972-12-31 16:00:00 Pacific Standard Time" "1972-12-31 16:00:00
Pacific Standard Time"
[5] "1973-12-31 16:00:00 Pacific Standard Time" "1973-12-31 16:00:00
Pacific Standard Time"
```

5.4 Sử dụng R cho các phép tính ma trận

Như chúng ta biết ma trận (matrix), nói đơn giản, gồm có dòng (row) và cột (column). Khi viết **A**[*m*, *n*], chúng ta hiểu rằng ma trận **A** có *m* dòng và *n* cột. Trong R, chúng ta cũng có thể thể hiện như thế. Ví dụ: chúng ta muốn tạo một ma trận vuông A gồm 3 dòng và 3 cột, với các phần tử (element) 1, 2, 3, 4, 5, 6, 7, 8, 9, chúng ta viết:

$$A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

Và với R:

```
> y <- c(1,2,3,4,5,6,7,8,9)
> A <- matrix(y, nrow=3)
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Nhưng nếu chúng ta lệnh:

```
> A <- matrix(y, nrow=3, byrow=TRUE)
> A
```

thì kết quả sẽ là:

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Tức là một **ma trận chuyển vị (transposed matrix).** Một cách khác để tạo một ma trận hoán vị là dùng `t()`. Ví dụ:

```
> y <- c(1,2,3,4,5,6,7,8,9)
> A <- matrix(y, nrow=3)
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

và B = A' có thể diễn tả bằng R như sau:

```
> B <- t(A)
> B
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Ma trận vô hướng (scalar matrix) là một ma trận vuông (tức số dòng bằng số cột), và tất cả các phần tử ngoài đường chéo (off-diagonal elements) là 0, và phần tử đường chéo là 1. Chúng ta có thể tạo một ma trận như thế bằng R như sau:

```
> # tạo ra mộ ma trận 3 x 3 với tất cả phần tử là 0.
> A <- matrix(0, 3, 3)

> # cho các phần tử đường chéo bằng 1
```

```
> diag(A) <- 1
> diag(A)
[1] 1 1 1

> # bây giờ ma trận A sẽ là:
> A
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

5.4.1 Chiết phần tử từ ma trận

```
> y <- c(1,2,3,4,5,6,7,8,9)
> A <- matrix(y, nrow=3)
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> # cột 1 của ma trận A
> A[,1]
[1] 1 4 7

> # cột 3 của ma trận A
> A[3,]
[1] 7 8 9

> # dòng 1 của ma trận A
> A[1,]
[1] 1 2 3

> # dòng 2, cột 3 của ma trận A
> A[2,3]
[1] 6

> # tất cả các dòng của ma trận A, ngoại trừ dòng 2
> A[-2,]
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    3    6    9

> # tất cả các cột của ma trận A, ngoại trừ cột 1
> A[,-1]
     [,1] [,2]
[1,]    4    7
[2,]    5    8
[3,]    6    9
```

```
> # xem phần tử nào cao hơn 3.
> A>3
      [,1] [,2] [,3]
[1,] FALSE TRUE TRUE
[2,] FALSE TRUE TRUE
[3,] FALSE TRUE TRUE
```

5.4.2 Tính toán với ma trận

Cộng và trừ hai ma trận. Cho hai ma trận A và B như sau:

```
> A <- matrix(1:12, 3, 4)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

> B <- matrix(-1:-12, 3, 4)
> B
     [,1] [,2] [,3] [,4]
[1,]   -1   -4   -7  -10
[2,]   -2   -5   -8  -11
[3,]   -3   -6   -9  -12
```

Chúng ta có thể cộng A+B:

```
> C <- A+B
> C
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
```

Hay A-B:

```
> D <- A-B
> D
     [,1] [,2] [,3] [,4]
[1,]    2    8   14   20
[2,]    4   10   16   22
[3,]    6   12   18   24
```

Nhân hai ma trận. Cho hai ma trận:

$$A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix} \qquad \text{và} \qquad B = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Chúng ta muốn tính *AB*, và có thể triển khai bằng R bằng cách sử dụng %*% như sau:

```
> y <- c(1,2,3,4,5,6,7,8,9)
> A <- matrix(y, nrow=3)
> B <- t(A)
> AB <- A%*%B
> AB
     [,1] [,2] [,3]
[1,]   66   78   90
[2,]   78   93  108
[3,]   90  108  126
```

Hay tính *BA*, và có thể triển khai bằng R bằng cách sử dụng %*% như sau:

```
> BA <- B%*%A
> BA
     [,1] [,2] [,3]
[1,]   14   32   50
[2,]   32   77  122
[3,]   50  122  194
```

Nghịch đảo ma trận và giải hệ phương trình. Ví dụ chúng ta có hệ phương trình sau đây:

$$3x_1 + 4x_2 = 4$$
$$x_1 + 6x_2 = 2$$

Hệ phương trình này có thể viết bằng kí hiệu ma trận: $AX = Y$, trong đó:

$$A = \begin{pmatrix} 3 & 4 \\ 1 & 6 \end{pmatrix}, \quad X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \qquad \text{và} \qquad Y = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$$

Nghiệm của hệ phương trình này là: $X = A^{-1}Y$, hay trong R:

```
> A <- matrix(c(3,1,4,6), nrow=2)
> Y <- matrix(c(4,2), nrow=2)
> X <- solve(A)%*%Y
> X
          [,1]
[1,] 1.1428571
[2,] 0.1428571
```

Chúng ta có thể kiểm tra:

```
> 3*X[1,1]+4*X[2,1]
[1] 4
```

Trị số eigen cũng có thể tính toán bằng function `eigen` như sau:

```
> eigen(A)
$values
[1] 7 2

$vectors
           [,1]       [,2]
[1,] -0.7071068 -0.9701425
[2,] -0.7071068  0.2425356
```

Định thức (determinant). Làm sao chúng ta xác định một ma trận có thể đảo nghịch hay không? Ma trận mà định thức bằng 0 là **ma trận suy biến** (**singular matrix**) và không thể đảo nghịch. Để kiểm tra định thức, R dùng lệnh `det()`:

```
> E <- matrix((1:9), 3, 3)
> E
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> det(E)
[1] 0
```

Nhưng ma trận F sau đây thì có thể đảo nghịch:

```
> F <- matrix((1:9)^2, 3, 3)
> F
     [,1] [,2] [,3]
[1,]    1   16   49
[2,]    4   25   64
[3,]    9   36   81
> det(F)
[1] -216
```

Và nghịch đảo của ma trận F (F^{-1}) có thể tính bằng function `solve()` như sau:

```
> solve(F)
          [,1]      [,2]       [,3]
[1,]  1.291667 -2.166667  0.9305556
[2,] -1.166667  1.666667 -0.6111111
[3,]  0.375000 -0.500000  0.1805556
```

Ngoài những phép tính đơn giản này, R còn có thể sử dụng cho các phép tính phức tạp khác. Một lợi thế đáng kể của R là phần mềm cung cấp cho người sử dụng tự do tạo ra những phép tính phù hợp cho từng vấn đề cụ thể. Trong vài chương sau, tôi sẽ quay lại vấn đề này chi tiết hơn.

R có một package `Matrix` chuyên thiết kế cho tính toán ma trận. Bạn đọc có thể tải package xuống, cài vào máy, và sử dụng, nếu cần. Địa chỉ để tải là:

http://cran.au.r-project.org/bin/windows/contrib/r-release/Matrix_0.995-8.zip

cùng với tài liệu chỉ dẫn cách sử dụng (dài khoảng 80 trang):

http://cran.au.r-project.org/doc/packages/Matrix.pdf