Initiation to R software Session II

Pierre Michel

Master AMSE 1st year, 2019

Matrices

Matrices: `matrix()`

Matrices are vectors that take one more argument: the **dimenson**: `dim()`.

The dimension is a vector of length 2 defining the number of **rows** and **columns** of the matrix.

Values in a matrix have the **same mode**.

```
M = matrix(1:10, 2, 5)
M
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

Matrices: `dim()`

Really ? Matrices are vectors ?

```
M = 1:10
dim(M) = c(2,5)
M
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

```
dim(M)
```

```
## [1] 2 5
```

Creating matrices: `matrix()`

```
matrix(data, nrow, ncol, byrow, dimnames)
```

- `data`: a vector,
- `nrow`: number of rows,
- `ncol`: number of columns,
- `byrow`: filling by row ?
- `dimnames`: list of two vectors, with names of rows and columns respectively.

Creating matrices: `matrix()`

```
M=matrix(1:10, 2, 5)
N=matrix(1:10, 2, 5, byrow = T)
R=matrix(1:10, 2, 5, dimnames = list(1:2, letters[1:5]))
print(M); print(N); print(R)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
```

```
##   a b c d  e
## 1 1 3 5 7  9
## 2 2 4 6 8 10
```

Creating matrices: `matrix()`

What if the length of the vector and the dimension of the matrix do not match ?

```
P=matrix(1:5, 2, 5); P;
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    2    4
## [2,]    2    4    1    3    5
```

```
Q=matrix(1:3, 2, 5);
```

```
## Warning in matrix(1:3, 2, 5): la longueur des données [3] n'est pas
## diviseur ni un multiple du nombre de lignes [2]
```

Selecting matrix elements

(De)select **one row/column** using **position indices** (i, j).

```
M = matrix(1:15, 3, 5)
M; M[1,]; M[3,]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    2    5    8   11   14
## [3,]    3    6    9   12   15

## [1]  1  4  7 10 13

## [1]  3  6  9 12 15
```

Selecting matrix elements

(De)select **several rows/columns** using **position indices** (i, j).

```
M; M[c(1,3),]; M[-c(2,4),]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    2    5    8   11   14
## [3,]    3    6    9   12   15

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    3    6    9   12   15

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    3    6    9   12   15
```

Selecting matrix elements

(De)select **sub-matrices** using **position indices** (i, j).

```
M; M[1,2]; M[c(1,2),-3]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    2    5    8   11   14
## [3,]    3    6    9   12   15
```

```
## [1] 4
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4   10   13
## [2,]    2    5   11   14
```

Selecting matrix elements

(De)select **sub-matrices** using **logical matrices**.

```
M; M[M[,1]>1,]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    2    5    8   11   14
## [3,]    3    6    9   12   15
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2    5    8   11   14
## [2,]    3    6    9   12   15
```

Selecting matrix elements

(De)select **vectors** using **linear indices** (return a vector).

```
M[3]
```

```
## [1] 3
```

```
v = 1:3; M[v]
```

```
## [1] 1 2 3
```

```
M[-v]; M[M>7]
```

```
##  [1]  4  5  6  7  8  9 10 11 12 13 14 15
```

```
## [1]  8  9 10 11 12 13 14 15
```

Matrix operations

- Usual operators: `+,-,*,/,=,>,<, ...`
- Algebraic matrix product: `%*%`
- Matrix transposition: `t()`
- Diagonalization: `eigen()`
- Determinant: `det()`
- Determinant: `det()`
- Inverse matrix: `solve()`

```
# Try this example
M=matrix(c(1,2,1,0),2,2)
v=c(1,0,2,1)
M
v
M+v
```

Matrix operations: + and *

```
M = matrix(c(1,2,1,0),2,2); N = matrix(1:4,2,2)
M + 2
```

```
##      [,1] [,2]
## [1,]    3    3
## [2,]    4    2
```

```
M * N
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    4    0
```

Matrix operations: %*% and t()

```
M %*% N
```

```
##      [,1] [,2]
## [1,]    3    7
## [2,]    2    6
```

```
t(M)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    1    0
```

Matrix operations: useful functions

- `diag()` can take both vectors and matrices as input. If the input is a **vector**, it returns a **diagonal matrix** with the vector in diagonal. If the input is a **matrix**, it returns a vector which is the matrix diagonal.
- `sum()` computes the sum of all elements in a matrix, use `sum(, na.rm = T)` to avoid missing values.
- `rbind()` and `cbind()` concatenate vertically (resp. horizontally) two matrices.

```
# Try this example
v=1:2; diag(v)
M=matrix(1:2,2,2); diag(M)
sum(M)
cbind(M,c(1,1)); rbind(M,c(1,1))
```

Matrix operations: `apply()`

This function applies a function (or an operator) to the rows/columns of a matrix.

`apply(M, margin, fun, ...)`

- `M` is a matrix
- `margin` specifies rows (`margin = 1`) or columns (`margin = 2`)
- `fun` is the function to apply

```
# Try this example
M=matrix(c(1,2,1,0),2,2)
apply(M, 1, sum) # sum of lines
apply(M, 2, sum) # sum of columns
```

Lists

Lists

A list is an ordered collection of **objects** (vectors, matrices, factors, dataframes, lists). These objects do not need to have the same type, mode and length.

```
list(x = 1:5, y = letters[1:5], a = c(T, F))
```

```
## $x
## [1] 1 2 3 4 5
##
## $y
## [1] "a" "b" "c" "d" "e"
##
## $a
## [1]  TRUE FALSE
```

Creating lists

Lists can be **named**, or **unnamed**.

```
L = list(x = 1:5, y = letters[1:8]); L # named list
```

```
## $x
## [1] 1 2 3 4 5
##
## $y
## [1] "a" "b" "c" "d" "e" "f" "g" "h"
```

```
L = list(1:5, letters[1:8]); L # unnamed list
```

```
## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## [1] "a" "b" "c" "d" "e" "f" "g" "h"
```

Selecting list elements

An object in a list can be accessed by **name** (using $) or by **index** (using `[[]]`), selection of **sublists** is also possible.

```
# try this example
L$x # select by name
is.vector(L$x) # is vector ?
L[[1]] # select by index
L[1] # a sublist
L[c(1,2)] # another sublist
```

Dataframes

Dataframes (`data.frame`)

- Inside R, dataframes are **lists**, for which fields represent **columns** of same length.
- Visually, dataframes are **matrix**, in which columns can have different modes (character, numeric, logical).

```
L = LETTERS[1:3]
D = data.frame(x=rep(1,5), y=1:5, z=sample(L, 5, repl=T))
D
```

```
##   x y z
## 1 1 1 C
## 2 1 2 B
## 3 1 3 A
## 4 1 4 A
## 5 1 5 A
```

Selecting `data.frame` elements

- One can select the rows/columns and elements using their **position indices** or **names**.
- An element from a `data.frame` is a **factor**

```
# Try this example
L = LETTERS[1:3]
D = data.frame(x=rep(1,5), y=1:5, z=sample(L, 5, repl=T))
D[,3]
D["fac"]
D$fac
D[1,3]
D[1,"z"]
is.factor(D[,3])
```

Useful functions for `data.frame` (and `matrix`)

- `names()`, `colnames()`: return the names of columns or create them.
- `rownames()`: returns the names of rows or create them.
- `dimnames()`: returns a 2-field list containing the names of rows/columns
- `dim()`: returns the dimension of the `data.frame` (or `matrix`).
- `rbind()`, `rbind()`: concatenate by rows/columns.
- `edit()`, `fix()`: opens an editor to set the values of the `data.frame`.

Useful functions for data.frame (and matrix)

```r
# Try this example
L = LETTERS[1:3]
D = data.frame(x=rep(1,5), y=1:5, z=sample(L, 5, repl=T))
colnames(D)
colnames(D)=c(LETTERS[1:2], "z")
D
rownames(D)=letters[1:5]
D
dimnames(D)
dim(D)
cbind(D, room = rep(c(1,2), length = 5))
```

Reading `data.frame`

You can list objects of type `data.frame`:

- `data()`: list all objects of type `data.frame` available by default in the package `datasets`.
- `try(data(package = "cluster"))`: list all objects of type `data.frame` available in the package `cluster`.

You can also load them:

- `data(USArrests)`: load the `data.frame` *USArrests* (in package `datasets`).
- `help(USArrests)`: give information about `data.frame` *USArrests*

```
# Try this example
library(cluster)
data(agriculture)
# or data(agriculture,package="cluster")
```

Complements

Modes

type	mode	multimode
vector	numeric, character, logical	no
factor	numeric, character	no
array	numeric, character, logical	no
matrix	numeric, character, logical	no
data.frame	numeric, character, logical	yes
ts	numeric, character, logical	yes
list	numeric,character, logical	yes

Conversion / Identification

`as` for conversion, `is` for identification

- `as.vector`
- `is.vector()`
- `as.factor()`
- `is.factor()`
- `as.list()`
- `is.list()`
- `as.matrix()`
- `is.matrix()`
- `as.data.frame()`
- `is.data.frame()`

Very useful functions...

function_name	description
sum(x)	Sum of the elements of x
prod(x)	Product of the elements of x
max(x), min(x)	Maximum, minimum of the elements of
which.max(x)	Returns the index of the maximum of the elements of x
which.min(x)	Returns the index of the minimum of the elements of x
range(x)	Similar to c(min(x), max(x))
mean(x)	Mean of the elements of x
median(x)	Median of the elements of x

Other very useful functions...

function_name	description
var(x)	Variance of the elements of x
cov(x)	Covariance matrix if x is a matrix
cor(x)	Correlation matrix of x if x is a matrix or a data.frame
sd(x)	Standard deviation of the elements of x
round(x,n)	Rounds the elements of x to n decimals
rev(x)	Reverse the order of the elements of x
sort(x)	Sort the elements of x in ascending order
rank(x)	Rank the elements of x
scale(x)	Scale (center and reduce) x

. . . and other very useful functions. . .

function_name	description
pmin(x,y,. . .)	A vector whose element i is the minimum of x[i] and y[i]
cumsum(x)	A vector whose element i is the sum of x[i] and y[i]
cumprod()	Similar as cumsum() with product
cummin()	Similar as cumsum() with minimum
cummax()	Similar as cumsum() with maximum
match(x,y)	Vector of same length as x with elements of x that are in y
choose(x,k)	Binomial coefficient
na.omit()	Remove observations with missing values
na.fail()	Returns an error message if x contains at least one NA

. . . and others

function_name	description
unique()	Returns a similar object without duplicates
table()	Returns table of counts of the values of x
subset()	Returns a selection of x based on criteria
sample(x,n)	Random sampling of x of size n, without replacement

Do not hesitate to try all of these powerful functions !