Introducción a R

Aspectos básicos (tipos, I/O, aleatoreidad)

Juan A. Botía

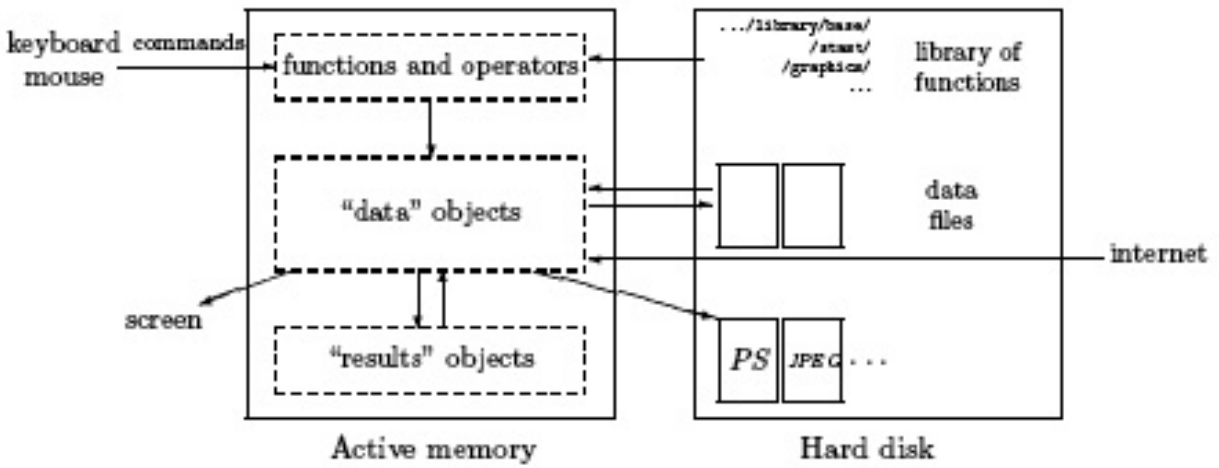
Departamento de Ingeniería de la Información y las Comunicaciones
Universidad de Murcia

Tratamiento Inteligente d la Información y Aplicaciones

¿Qué es R?

- Una herramienta para realizar análisis de datos
  - Matricial y matemático
  - Estadístico
  - Clasificación, clustering, etc...
  - Computación científica
- Es interpretado
- Es orientado a objetos
- Funciona en base a paquetes

Cómo R trabaja



Normalmente se gestionan varios objetos en memoria

- Un objeto se crea mediante un operador de asignación
- Si el objeto ya existía, se sobrescribe
- Es posible listar los objetos en memoria (también usar patrones)
- También es posible mirar con detalle a su contenido
- Podemos crear objetos compuestos a partir de objetos simples

Ejemplos de lo anterior

```
> n <- 15
> n
[1] 15
> 5 -> n
> n
[1] 5
> x <- 1
> X <- 10
> x
[1] 1
> X
[1] 10
```

```
> name <- "Carmen"
> n1 <- 10; n2 <- 100
> m <- 0.5
> ls()
[1] "m" "n1" "n2" "name"
> ls(pat = "m")
[1] "m" "name"
> ls(pat = "^m")
[1] "m"
> ls.str()
m : num 0.5
n1 : num 10
n2 : num 100
name : chr "Carmen"
```

```
> M <- data.frame(n1, n2, m)
> ls.str(pat = "M")
M : 'data.frame':
        1 obs. of 3 variables:
$ n1: num 10
$ n2: num 100
$ m : num 0.5
> ls.str(pat="M", max.level=-1)
M : 'data.frame':
        1 obs. of 3 variables:
```

Atributos de los objetos

Los objetos tienen un nombre y un contenido, pero también tienen

- Un modo: es el tipo del dato básico
  - 4 principales: numérico, caracter, complejo y lógico (`TRUE`, `FALSE`)
  - Utilizamos `mode()`
- Una longitud: la cantidad de elementos básicos
  - Utilizamos `length()`
- Independientemente del modo, si un valor no está, usamos `NA`, si es infinito, `Inf` o `-Inf`, si no es un número `NaN`
- Para representar strings usamos las comillas dobles, para caracteres las simples

Ejemplos de lo anterior

```
> x <- 1
> mode(x)
[1] "numeric"
> length(x)
[1] 1
> A <- "Gomphotherium"
> compar <- TRUE
> z <- 1i
> mode(A)
[1] "character"
> mode(compar)
[1] "logical"
> mode(z)
[1] "complex"
```

```
> x <- 5/0
> x
[1] Inf
> exp(x)
[1] Inf
> exp(-x)
[1] 0
> x - x
[1] NaN
```

```
> x <- "Doub quotes \" delimitate
  R's strings."
> x
[1] "Double quotes \" delimitate
  R's strings."
> cat(x)
Double quotes " delimitate
  R's strings.
> x <- 'Double quotes " delimitate
  R\'s strings.'
> x
[1] "Double quotes \" delimitate
  R's strings."
```

Leyendo datos de un fichero

- R siempre está situada en un directorio de trabajo (`getwd()`), que se puede cambiar (`setwd()`), de ahí lee los ficheros
- Es posible leer de ficheros en ASCII, Excel, SAS, SPSS, SQL-like (estas últimas necesitan paquetes adicionales)
- La función `> mydata <- read.table(''data.dat'')`
  - Creará un objeto `mydata` en el que cada variable se llamará por defecto `V1`, `V2`, ...
  - Se podrán acceder mediante `mydata$Vi` siendo *i* el índice, o mediante `mydata[''Vi'']`, o incluso `mydata[,i]`
  - Hacer `help(read.table)` para ver todas las opciones
- La función `> mydata <- scan("data.dat", what = list("", 0, 0))`
  - Indicamos que la primera es de modo carácter y las otras dos numéricas
  - Hacer `help(scan)` para ver todas las opciones

Escribiendo datos en un fichero

- Podemos hacerlo con `write.table()`
- Podemos guardar cualquier tipo de objeto en un fichero
- La forma más sencilla de utilizarlo es

  `write(x,file=''data.txt'')`

  en donde x es el nombre del objeto
- Podemos hacer `help(write)` para ver todas las opciones

Generando datos artificialmente

- A veces es útil generar secuencias regulares de enteros
- El operador ':' indica una secuencia y tiene prioridad sobre los aritméticos
- Podemos crear secuencias por intervalos y salto mediante `seq()`
- Podemos repetir el mismo número con `rep()`
- Podemos crear una serie de secuencias con `sequence()`
- O series de factores con `gl(k,n)`
- O un frame de datos con combinaciones de factores dados como argumentos con `expand.grid()`

Ejemplos de lo anterior

```
> x <- 1:30
> 1:10-1
[1] 0 1 2 3 4 5 6 7 8 9
> 1:(10-1)
[1] 1 2 3 4 5 6 7 8 9
> seq(1, 5, 0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
> seq(length=9, from=1, to=5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
> c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
rep(1, 30)
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1...
> sequence(4:5)
[1] 1 2 3 4 1 2 3 4 5
> sequence(c(10,5))
[1] 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5
```

Ejemplos de lo anterior (y 2)

```
> gl(3, 5)
[1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels: 1 2 3
> gl(3, 5, length=30)
[1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
Levels: 1 2 3
> gl(2, 6, label=c("Male", "Female"))
[1] Male Male Male Male Male Male
[7] Female Female Female Female Female Female
Levels: Male Female
> gl(2, 10)
[1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
Levels: 1 2
> gl(2, 1, length=20)
[1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
Levels: 1 2
> 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
16
```

Ejemplos de lo anterior (y 3)

```
> expand.grid(h=c(60,80), w=c(100, 300), sex=c("Male", "Female"))
h w sex
1 60 100 Male
2 80 100 Male
3 60 300 Male
4 80 300 Male
5 60 100 Female
6 80 100 Female
7 60 300 Female
8 80 300 Female
```

Generación de números aleatorios

Es muy útil generar muestras artificiales que siguen una determinada distribución de probabilidad, mediante la forma genérica

```
rfunc(n, p1, p2, ...)
```

en donde `rfunc` es la d.p. y los *pi* son parámetros de la misma

law	function
Gaussian (normal)	`rnorm(n, mean=0, sd=1)`
exponential	`rexp(n, rate=1)`
gamma	`rgamma(n, shape, scale=1)`
Poisson	`rpois(n, lambda)`
Weibull	`rweibull(n, shape, scale=1)`
Cauchy	`rcauchy(n, location=0, scale=1)`
beta	`rbeta(n, shape1, shape2)`
'Student' (t)	`rt(n, df)`
Fisher–Snedecor (F)	`rf(n, df1, df2)`
Pearson (χ^2)	`rchisq(n, df)`
binomial	`rbinom(n, size, prob)`
multinomial	`rmultinom(n, size, prob)`
geometric	`rgeom(n, prob)`
hypergeometric	`rhyper(nn, m, n, k)`
logistic	`rlogis(n, location=0, scale=1)`
lognormal	`rlnorm(n, meanlog=0, sdlog=1)`
negative binomial	`rnbinom(n, size, prob)`
uniform	`runif(n, min=0, max=1)`
Wilcoxon's statistics	`rwilcox(nn, m, n)`, `rsignrank(nn, n)`

Vectores

Se crean mediante

```
vector(mode,length)
```

como en

```
>vector(mode="integer",3)
[1] 0 0 0
```

aunque tambien se puede hacer con las funciones, con ejemplos

```
numeric(3) logical(5) character(2)
```

Factores

Un factor es una variable categórica, siendo su llamada por defecto

```
  factor(x, levels = sort(unique(x), na.last = TRUE),
labels = levels, exclude = NA, ordered = is.ordered(x))}
```

siendo los niveles el número de valores distintos y labels las correspondientes etiquetas de dichos valores, p.ej.

```
> factor(1:3)
[1] 1 2 3
Levels: 1 2 3
> factor(1:3, levels=1:5)
[1] 1 2 3
Levels: 1 2 3 4 5
> factor(1:3,
    labels=c("A", "B", "C"))
[1] A B C
Levels: A B C
```

```
> factor(1:5, exclude=4)
[1] 1 2 3 NA 5
Levels: 1 2 3 5
> ff <- factor(c(2, 4), lev
> ff
[1] 2 4
Levels: 2 3 4 5
> levels(ff)
```

Matrices

Como un vector, con un atributo adicional de tipo vector con dos elementos (filas y columnas)

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
  dimnames = NULL)
```

siendo `byrow` la forma de llenar la matriz (por filas o columnas)

```
> matrix(data=5, nr=2, nc=2)
[,1] [,2]
[1,] 5 5
[2,] 5 5
> matrix(1:6, 2, 3)
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6
> matrix(1:6, 2, 3, byrow=TRUE)
[,1] [,2] [,3]
[1,] 1 2 3
[2,] 4 5 6
```

```
> x <- 1:15
> x
[1] 1 2 3 4 5 6 7 8 9
    10 11 12 13 14 15
> dim(x)
NULL
> dim(x) <- c(5, 3)
> x
[,1] [,2] [,3]
[1,] 1 6 11
[2,] 2 7 12
[3,] 3 8 13
[4,] 4 9 14
[5,] 5 10 15
```

Frames de datos

Podemos crearlos con el comando `data.frame(...)` como en

```
> x <- 1:4; n <- 10; M <- c(10, 35); y <- 2:4
> data.frame(x, n)
x n
1 1 10
2 2 10
3 3 10
4 4 10
```

siendo la llamada por defecto

```
data.frame(..., row.names = NULL, check.rows = FALSE,
                check.names = TRUE,
                stringsAsFactors = default.stringsAsFactors())
```

Listas

En las listas no hay restricciones en cuanto a los elementos que pueden aparecer ahí

```
> L1 <- list(x, y);
    L2 <- list(A=x, B=y)
> L1
[[1]]
[1] 1 2 3 4
[[2]]
[1] 2 3 4
```

```
> L2
$A
[1] 1 2 3 4
$B
[1] 2 3 4
> names(L1)
NULL
> names(L2)
[1] "A" "B"
```

Operadores

Más detalles en los manuales

Operators					
Arithmetic		Comparison		Logical	
+	addition	<	lesser than	! x	logical NOT
-	subtraction	>	greater than	x & y	logical AND
*	multiplication	<=	lesser than or equal to	x && y	id.
/	division	>=	greater than or equal to	x | y	logical OR
^	power	==	equal	x || y	id.
%%	modulo	!=	different	xor(x, y)	exclusive OR
%/%	integer division				

Algunos detalles sobre indexación

Podemos acceder muy fácilmente a un elemento en un vector

```
> x <- 1:5
26
> x[3]
[1] 3
> x[3] <- 20
> x
[1] 1 2 20 4 5
```

Si accedemos a una matriz o un frame de datos es igual de simple

```
> x <- matrix(1:6, 2, 3)
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> x[, 3] <- 21:22
> x
     [,1] [,2] [,3]
[1,]    1    3   21
[2,]    2    4   22
> x[, 3]
[1] 21 22
```

Más detalles sobre indexación

Podemos eliminar filas y columnas, utilizando signos negativos

```
> x[, -1]
     [,1] [,2]
[1,]    3   21
[2,]    4   22
> x[, -(1:2)]
[1] 21 22
> x[, -(1:2), drop = FALSE]
     [,1]
[1,]   21
[2,]   22
```

Podemos indicar aquellos datos a los que queremos acceder mediante condiciones

```
> x <- 1:10
> x[x >= 5] <- 20
> x
[1] 1 2 3 4 20 20 20 20 20 20
> x[x == 1] <- 25
> x
[1] 25 2 3 4 20 20 20 20 20 20
```

Cálculos matemáticos sencillos

- Los cálculos pueden hacerse sobre datos simples o sobre vectores
- Los siguientes son ejemplos de exprsiones
  - sum(x), prod(x), max(x), min(x)
  - which.max(x), which.min(x) devuelven los índices de máximo y mínimo respectivamente
  - var(x), cov(x), cor(x), var(x,y), cov(x,y)...

Cálculo matricial

Podemos unir dos matrices, mediante filas y columnas con `rbind()` y `cbind()`, respectivamente

```
> m1 <- matrix(1, nr = 2, nc = 2)
> m2 <- matrix(2, nr = 2, nc = 2)
> rbind(m1, m2)
[,1] [,2]
[1,] 1 1
[2,] 1 1
[3,] 2 2
[4,] 2 2
> cbind(m1, m2)
[,1] [,2] [,3] [,4]
33
[1,] 1 1 2 2
[2,] 1 1 2 2
```

Cálculo matricial (II)

Podemos multiplicar dos matrices, mediante %*%, como en

```
> rbind(m1, m2) %*% cbind(m1, m2)
[,1] [,2] [,3] [,4]
[1,] 2 2 4 4
[2,] 2 2 4 4
[3,] 4 4 8 8
[4,] 4 4 8 8
> cbind(m1, m2) %*% rbind(m1, m2)
[,1] [,2]
[1,] 10 10
[2,] 10 10
```

Cálculo matricial (III)

Podemos extraer o modificar la diagonal de una matriz, o construir una matriz diagonal

```
> diag(m1)
[1] 1 1
> diag(rbind(m1, m2) %*% cbind(m1, m2))
[1] 2 2 8 8
> diag(m1) <- 10
> m1
[,1] [,2]
[1,] 10 1
[2,] 1 10
> diag(3)
[,1] [,2] [,3]
[1,] 1 0 0
[2,] 0 1 0
[3,] 0 0 1
```

```
> v <- c(10, 20, 30)
> diag(v)
[,1] [,2] [,3]
[1,] 10 0 0
[2,] 0 20 0
[3,] 0 0 30
> diag(2.1, nr = 3, nc = 5)
[,1] [,2] [,3] [,4] [,5]
[1,] 2.1 0.0 0.0 0 0
[2,] 0.0 2.1 0.0 0 0
[3,] 0.0 0.0 2.1 0 0
```

Podemos utilizar `t()` para la traspuesta, `solve()` para la inversa,...

Conclusiones

En esta primera parte hemos visto

- Lecturas y escrituras desde y en ficheros
- Vectores, factores, frames de datos
- Operadores matemáticos y acceso a objetos
- Simple cálculo matricial

Conclusiones: R es un lenguaje interpretado en que los tipos se manejan de forma sencilla y flexible