Programación Estadística

Jocelyn Simmonds (jsimmond@dcc.uchile.cl)

Departamento de Ciencias de la Computación

Temario: parte practica

Programación en R:

- Clase 1: variables, funciones básicas, manipulación de workspaces
- Clase 2: texto, vectores, factores, arreglos, matrices, listas, data frames
- Clase 3: scripts, estructuras de control (if, while), definición de funciones
- Clase 4: gráficos de datos cualitativos/cuantitativos
- Clase 5: manipulación de strings, expresiones regulares, más graficos

Evaluaciones

- Laboratorios
  - Laboratorio 1: 8 de agosto
  - Laboratorio 2: 9 de agosto
  - Laboratorio 3: 11 de agosto
- Examen:
  - 16 de agosto (3hrs)
  - parte teórica y parte practica

$$\text{Nota final} = 0.4\,\text{promedio}\,(\text{Laboratorios}) + 0.6\,\text{Examen}$$

Introducción

¿Qué es la Programación Estadística?

Apoyar la tarea de análisis estadístico mediante la programación. Para esto, usaremos el ambiente de programación R[1]:

- R permite cargar y manipular datos
- Provee funciones para hacer cálculos en base a objetos (vectores, matrices, etc.)
- Provee funciones para la visualización de datos
- Incluye un lenguaje de programación que incorpora variables, condicionales, bucles, y funciones definidas por el usuario
- Software de libre acceso, disponible para Linux, OS X y Windows

[1] `https://www.r-project.org/`

Componentes de R

- Consola: ejecutar comandos en forma interactiva
- Scripts: programas que pueden ser ejecutados una y otra vez
- Workspace: conjunto de objetos creados por el usuario (a través de la consola, o ejecutando un script)
- Plots: gráficos creados para visualizar los datos

Trabajando con la Consola

```
jsimmond@pita: ~/R-workspace
File  Edit  View  Search  Terminal  Help
jsimmond@pita:~/R-workspace$ ls
total 8.0K
-rw-r--r-- 1 jsimmond users 140 Jul 23 15:37 test.lis
-rw-r--r-- 1 jsimmond users   9 Jul 27 11:26 test.R
jsimmond@pita:~/R-workspace$ R

R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> ls()
[1] "a" "b" "c" "n" "x" "y"
>
```

Trabajando con la Consola

1. Crear directorio donde almacenaran sus datos, scripts y el workspace
2. Inicializar R desde ese directorio

Podemos usar la consola como una calculadora simple:

```
1  > 12 * 23
2  [1] 276
3  > 5 ^ 2
4  [1] 25
```

El "[1]" indica la posición dentro del vector de respuestas.

Pruébenlo

Pueden guardar sus cálculos asignándolos a variables:

```
> a <- 12 * 23
> b <- 5 ^ 2
> a
[1] 276
> b
[1] 25
> a * b -> a
> a
[1] 6900
```

Nombres de variables: `A-Za-z0-9`, además de "." y "_". Inicien R y realicen los siguientes cálculos:

1. 123 por 11, dejando el valor en una variable llamada `a`
2. 4 elevado a 5, dejando el valor en una variable llamada `b`

Vectores

El objeto más comúnmente usado para guardar datos en R es el vector:

```
> x <- c(1, 2, 3.5, 3.6, 5, 29, 1.4)
> x    # todo lo que viene despues del # es un comentario
[1]  1.0  2.0  3.5  3.6  5.0 29.0  1.4
>
> vocales <- c("a", "e", "i", "o", "u")
> vocales
[1] "a" "e" "i" "o" "u"
```

La función `c()` toma un número arbitrario de argumentos, y retorna un vector con estos elementos.

Pruébenlo

Pueden usar el operador : para generar secuencias de números:

```
> x <- 1:15
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
> y <- 30:1
> y
 [1] 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6
[26]  5  4  3  2  1
> z <- 1.2:3.2
> z
[1] 1.2 2.2 3.2
```

Creen los siguientes vectores:

1. -5 -4 -3 -2 -1 0 1 2 3 4 5
2. 1 2 3 3 2 1 1 2 3 (*hint:* `y <- c(1:3, ..)`)

Variables y vectores

Pueden combinar variables y vectores:

```
1  > a <- 4 * 7
2  > b <- 6 * 5
3  > c <- 9 * 2
4  >
5  > d <- c(c, a, b)
6  > d
7  [1] 18 28 30
8  >
```

Posición de elementos

Pueden acceder a los elementos de un vector por posición:

```
1   > e <- c(22, 33, 44, 55)
2   > e
3   [1] 22 33 44 55
4   >
5   > e[1]
6   [1] 22
7   > e[2]
8   [1] 33
9   > e[length(e)]
10  [1] 55
```

La función `length(x)` retorna el largo de un vector:

```
1   > a <- 4 * 7
2   > length(a)
3   [1] 1
```

Aritmética con vectores

Podemos realizar cálculos con vectores:

```
1   > x <- 1:10
2   > x
3    [1]  1  2  3  4  5  6  7  8  9 10
4   > x2 <- c(0, 2)
5   > z <- x + x2    # x2 se "recicla" tantas veces como sea necesario
6   > z              #    z[1] = x[1] + x2[1]
7                    #    z[2] = x[2] + x2[2]
8                    #    z[3] = x[3] + x2[1] ...
9    [1]  1  4  3  6  5  8  7 10  9 12
10  >
11  > y <- 1/x
12  > y
13   [1] 1.0000000 0.5000000 0.3333333 0.2500000 0.2000000 0.1666667 0.1428571
14   [8] 0.1250000 0.1111111 0.1000000
```

Operadores: +, −, ∗, /, ˆ (potencia)

Pruébenlo

Creen los siguientes vectores:

1. los primeros 15 números cuadrados (1, 4, 9, 16, ... 225). No usen `c()`
2. los números pares entre 10 y 40 (incluyendo ambos extremos)

Funciones

R incluye varias funciones que podemos aplicar a variables y vectores:

```
> log(1)
[1] 0
> log(1:10)
 [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
 [8] 2.0794415 2.1972246 2.3025851
> x3 <- c(2, 9, 1, 6, 7, 8, 4, 5, 3, 10)
> x3
 [1]
> max(x3)
[1] 10
> min(x3)
[1] 1
> range(x3)
[1]  1 10
```

Formato de funciones: `salida <- nombre_de_funcion(parametros)`

Otras funciones: `exp`, `sin`, `cos`, `sqrt`, ...

Funciones

```
1   > x4 <- 1:5
2   > sum(x4)
3   [1] 15
4   > prod(x4)
5   [1] 120
6   > mean(x4)     # equivalente a sum(x4)/length(x4)
7   [1] 3
8   > var(x4)      # equivalente a sum((x4 - mean(x4))^2) / (length(x4) - 1)
9   [1] 2.5
10  > median(x4)
11  [1] 3
```

Pueden usar `help(...)` para ver todos los parámetros que recibe una función.

Pruébenlo

Creen el vector de las edades de los participantes del curso, y calculen:

1. la edad promedio de los participantes,
2. las edades mínima y máxima, y
3. la desviación estándar de estos datos

Secuencias de números

Podemos usar la función `seq()` para generar secuencias no-triviales de números. Parámetros de `seq()`:

- desde `x`: `from = x`
- hasta `y`: `to = y`
- indicando la diferencia entre elementos: `by = d` ($-d$ si `from` $>$ `to`)
- indicando el número de elementos: `length = n`

Ejemplo:

```
1  > seq(from = 4, to = 10, by = 0.5)
2   [1]  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0  9.5 10.0
```

Ojo: no pueden usar los parámetros `by` y `length` al mismo tiempo.

Pruébenlo

Creen los siguientes vectores:

1. 20 valores entre π y 2π (π se escribe `pi` en R)
2. 0.500 0.495 0.490 0.485 ... 0.170 0.165 0.160 0.155 (sin usar el parámetro `length`)

Secuencias de números

`seq(vector)` genera un vector con los valores 1, 2, ... `length(vector)`:

```
1  > x <- NULL    # vector no definido
2  > seq(x)
3  integer(0)     # vector de enteros de largo 0 (vacio)
4  > x <- 10:1
5  > x
6   [1] 10  9  8  7  6  5  4  3  2  1
7  > y <- seq(x)
8  > y
9   [1]  1  2  3  4  5  6  7  8  9 10
```

Otras maneras de crear un vector vacio:

```
1  x <- integer(0)
2  x <- numeric(0)
```

Guardando el workspace

Estado del workspace

Pueden revisar el listado del objetos que existen en el workspace usando la función `ls()`:

```
> ls()
 [1] "a"       "b"       "c"       "d"       "e"       "g"       "n"
 [8] "vocales" "x"       "x2"      "x_2"     "x.2"     "x3"      "x4"
[15] "y"       "z"
> rm(c)    # elimina el objeto c
> rm(z)    # elimina el objeto z
> ls()
 [1] "a"       "b"       "d"       "e"       "g"       "n"       "vocales"
 [8] "x"       "x2"      "x_2"     "x.2"     "x3"      "x4"      "y"
```

La función `rm(objeto)` quita el objeto del workspace.

Estado del workspace

Recuperar una sesión de R:

```
> ...     # deben elegir guardar el workspace al salir
> ...     # este queda guardado en el directorio actual
> q()     # salir de R
Save workspace image? [y/n/c]: y
jsimmond@pita:~/R-workspace% ls -la
total 24K
drwxr-xr-x  2 jsimmond users 4.0K Jul 27 16:03 .
drwxr-xr-x 42 jsimmond users 4.0K Jul 26 19:01 ..
-rw-r--r--  1 jsimmond users 2.9K Jul 27 16:03 .RData
-rw-------  1 jsimmond users 2.5K Jul 27 16:03 .Rhistory
-rw-r--r--  1 jsimmond users  140 Jul 23 15:37 test.lis
-rw-r--r--  1 jsimmond users   13 Jul 27 11:42 test.R
jsimmond@pita:~/R-workspace%
jsimmond@pita:~/R-workspace% tail -n 5 .Rhistory
ls()
rm(z)
ls
ls()
q() # salir de R
jsimmond@pita:~/R-workspace%
```

Estado del workspace

Recuperar una sesión de R:

```
1  jsimmond@pita:~/R-workspace% R
2
3  R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
4  Copyright (C) 2013 The R Foundation for Statistical Computing
5  Platform: x86_64-pc-linux-gnu (64-bit)
6
7  ...
8
9  Type 'demo()' for some demos, 'help()' for on-line help, or
10 'help.start()' for an HTML browser interface to help.
11 Type 'q()' to quit R.
12
13 [Previously saved workspace restored]
14
15 > ls()
16  [1] "a"       "b"       "d"       "e"       "g"       "n"       "vocales"
17  [8] "x"       "x2"      "x_2"     "x.2"     "x3"      "x4"      "y"
18 >
```

Pruébenlo

1. Cierren R, guardando el workspace.
2. Vuelvan a iniciar R, cargando su sesión anterior.

Más R

Valores lógicos

R reconoce tres valores lógicos:

- `TRUE`: verdadero
- `FALSE`: falso
- `NA`: dato no disponible

Operadores relacionales: `<`, `<=`, `>`, `>=`, `==`, `!=`

```
1  > 1 > 0
2  [1] TRUE
3  > c(1, 4, 8) > 0
4  [1] TRUE TRUE TRUE
5  > c(1, 4, 8) >= 4
6  [1] FALSE  TRUE  TRUE
```

¿Qué pasa con los NA? ver diapo 33 . . .

Vectores lógicos

```
> 31 %% 7     # resto de la division entre dos enteros
[1] 3
> r <- 31 %% 7
> a <- 31 %/% 7   # retorna la parte entera de la division
> 7 * a + r
[1] 31
>
> x <- seq(1, 10)
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> con <- x %% 2 == 0
> con
 [1] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
> sum(con)
[1] 5
```

TRUE y FALSE toman los valores 1 y 0 en cualquier calculo numérico.
Por ejemplo, TRUE + TRUE = 2

Pruébenlo

Queremos corregir una prueba de alternativas, donde las respuestas y la pauta están guardadas en dos vectores:

```
1  > pauta = c(4, 2, 5, 1, 4, 3, 2, 5, 1, 3, 2)
2  > resp = c(4, 0, 5, 2, 4, 0, 2, 3, 1, 0, 1)   # 0 indica respuesta omitida
```

Obtengan el número de respuestas correctas comparando los dos vectores.

Operadores lógicos

- `c1 & c2`: conjunción
- `c1 | c2`: disyunción
- `!c1`: negación

```
> c1 <- c(TRUE, TRUE, FALSE, FALSE)
> c2 <- c(TRUE, FALSE, TRUE, FALSE)
> c1 & c2
[1]  TRUE FALSE FALSE FALSE
> c1 | c2
[1]  TRUE  TRUE  TRUE FALSE
> !c1
[1] FALSE FALSE  TRUE  TRUE
> !c2
[1] FALSE  TRUE FALSE  TRUE
```

Pruébenlo

Sigamos con el ejemplo de la prueba de alternativas:

```
1  > pauta = c(4, 2, 5, 1, 4, 3, 2, 5, 1, 3, 2)
2  > resp = c(4, 0, 5, 2, 4, 0, 2, 3, 1, 0, 1)   # 0 indica respuesta omitida
```

Calculen el puntaje, ahora descontando $\frac{1}{4}$ de punto por cada respuesta incorrecta (exceptuando las omitidas).

Operadores lógicos

¿Qué pasa con los valores `NA`?

```
> c1 <- c(TRUE, TRUE, FALSE, FALSE)
> c2 <- c(TRUE, FALSE, TRUE, FALSE)
> c3 <- c(NA, NA, NA, NA)
> c1 & c3
[1]    NA    NA FALSE FALSE
> c2 & c3
[1]    NA FALSE    NA FALSE
> c1 | c3
[1] TRUE TRUE   NA   NA
> c2 | c3
[1] TRUE   NA TRUE   NA
> !c3
[1] NA NA NA NA
```

Cuidado: NA no es lo mismo que NaN

NA representa un elemento que falta en los datos de entrada, NaN representa un resultado imposible de calcular:

```
> val <- 0/0
[1] NaN
>
> x <- 0:10
> x/0
 [1] NaN Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf
>
> is.nan(val)
[1] TRUE
```

Datos no disponibles

En la mayoría de los casos, operar sobre un `NA` produce un `NA`:

```
> 5 + NA
[1] NA
>
> a <- 5
> a + NA
[1] NA
>
> x <- c(2, 6, 2, NA, 6, 2, NA, 3, 4, NA, 9)
> x + 4
 [1]  6 10  6 NA 10  6 NA  7  8 NA 13
```

Ignorando datos no disponibles

A veces queremos ignorar los `NA`, para esto usaremos la función `is.na(vector)`:

```
1  > x <- c(2, 6, 2, NA, 6, 2, NA, 3, 4, NA, 9)
2  > is.na(x)
3   [1] FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE
4  >
5  > x[is.na(x)]
6  [1] NA NA NA
7  >
8  > x[!is.na(x)]
9  [1] 2 6 2 6 2 3 4 9
```

OJO: la función `is.na(x)` retorna `TRUE` para elementos `NA` y `NaN`, mientras que la función `is.nan(x)` solo retorna `TRUE` para elementos `NaN`.

Secuencias de índices

Podemos usar [] para seleccionar varios elementos de un vector al mismo tiempo:

```
> x <- 0:10
> x
 [1]  0  1  2  3  4  5  6  7  8  9 10
> x[x > 6]
[1]  7  8  9 10
> x[seq(1, length(x), by = 2)]
[1]  0  2  4  6  8 10
> x
> x[-3]
 [1]  0  1  3  4  5  6  7  8  9 10
> x[-(5:0)]
[1]  5  6  7  8  9 10
```

Secuencias de índices

Podemos usar [] para modificar varios elementos de un vector al mismo tiempo:

```
> x <- 0:10
> x
 [1]  0  1  2  3  4  5  6  7  8  9 10
> x[x %% 2 == 0] <- 0
> x
 [1] 0 1 0 3 0 5 0 7 0 9 0
>
> x <- 0:10
> x[x > 4] <- x[x > 4] + 4
> x
 [1]  0  1  2  3  4  9 10 11 12 13 14
>
```

Pruébenlo

Dado el vector `x <- c(3, 15, 9, 12, -1, 0, -12, 9, 6, 1)`, escriban los comandos necesarios para lograr que:

1. se deje en 0 los elementos negativos de `x`

```
> x
 [1]  3 15  9 12  0  0  0  9  6  1
```

2. se deje en 3 los elementos que son múltiplos de 3

```
> x
 [1]  3  3  3  3 -1  3  3  3  3  1
```

3. se multiplica los elementos pares de `x` por 5

```
> x
 [1]   3  15   9  60  -1   0 -60   9  30   1
```

Pruébenlo

Dado el vector `x <- c(3, 15, 9, 12, -1, 0, -12, 9, 6, 1)`, escriban los comandos necesarios para lograr que:

4. se deje los valores de `x` mayores que 10 en un nuevo vector

```
> x
[1] 15 12
```

5. se deje en 0 los elementos de `x` que son menores que el promedio

```
> x
 [1]  0 15  9 12  0  0  0  9  6  0
```