Inroducción a R

Alberto Torres Barrán

9 de Mayo de 2015

Índice

Índice

Introducción

- R es un lenguaje de programación y un entorno para manipular datos, cálculo y gráficos.
- Ventajas:
  - Proyecto de GNU (*open source*), cualquier puede contribuir al desarrollo.
  - Gran cantidad de paquetes (6551 a 21 de Abril de 2015).
  - La mayoría de nuevas tecnologías y algoritmos relacionados con estadística aparecen primero en R.
  - Documentación abundante en Internet, muchos grupos de usuarios activos.
- Desventajas:
  - Curva de aprendizaje inclinada, como la mayoría de lenguajes de programación.
  - La calidad de algunos paquetes.
  - Menor rendimiento que otros lenguajes de cálculo científico.
  - Gestión de memoria.

Comparación rendimiento

	Fortran	Python	R	Matlab	Java
fib	0.57	95.45	528.85	4258.12	0.96
parse_int	4.67	20.48	54.30	1525.88	5.43
quicksort	1.10	46.70	248.28	55.87	1.65
mandel	0.87	18.83	58.97	60.09	0.68
pi_sum	0.83	21.07	14.45	1.28	1.00
rand_mat_stat	0.99	22.29	16.88	9.82	4.01
rand_mat_mul	4.05	1.08	1.63	1.12	2.35

Tiempos de benchmark relativos a C (más pequeño es mejor, rendimiento de C = 1.0). Fuente: http://julialang.org/

Entorno R

- R es un lenguaje interpretado, por lo que no es necesario compilar el código fuente.
- El intérprete de R está disponible para los principales sistemas operativos (Windows, OSX y Linux): http://cran.r-project.org.
- En este curso vamos a trabajar con el IDE RStudio http://www.rstudio.com.
- RStudio proporciona un entorno similar al de Matlab.
- Documentación y manuales: http://www.r-project.org.

Comandos de R

- Distinguen entre mayúsculas y minúsculas.
- Se clasifican en *asignaciones* (el resultado se guarda) y *expresiones* (el resultado se imprime y se pierde).
- El operador de asignación es `<-` o `->` (no `=`, como en la mayoría de lenguajes).
- Los comandos se pueden escribir de forma interactiva en el intérprete o almacenar en un fichero de texto.
- Un fichero de comandos se carga con la expresión

  ```
  > source("fichero.R")
  ```

- Para obtener ayuda sobre un determinado comando se utiliza la expresión

  ```
  > help("lm")
  ```

  o alternativamente

  ```
  > ?lm
  ```

Índice

Objetos

- Todas las entidades que R crea y manipula se denominan objetos (variables, funciones, etc).
- Los objetos se almacenan en la memoria RAM del ordenador con un nombre específico.
- Listar todos los objetos en memoria

  ```
  > ls()
  ```

- Eliminar objetos `x` y `tmp` de la memoria

  ```
  > rm(x, tmp)
  ```

- Eliminar todos los objetos de la memoria

  ```
  > rm(list=ls())
  ```

- Al cerrar la sesión se pueden almacenar todos los objetos en el fichero `.RData`.
- Al abrir una nueva sesión se cargaran los objetos del fichero `.RData` almacenado en el directorio actual (si existe).

Objetos (cont.)

- Los tipos de datos básicos son `numeric`, `integer`, `complex`, `logical` y `character`.
- El modo de un objeto es el tipo básico de los elementos que contiene, se puede ver con la función `mode()`.
- Para convertir de un modo a otro se utilizan las funciones `as.integer()`, `as.numeric()`, etc.
- Todos los objetos tienen una clase, que se puede ver con la función `class()`.
- Algunas funciones producirán un resultado u otro dependiendo de la clase de sus argumentos.
- Las funciones `is.integer()`, `is.numeric()`, etc. devuelven verdadero o falso dependiendo si el objeto es de esa clase o no.

Índice

Vectores

- Un vector es una secuencia de datos del mismo tipo básico.
- Creación de vectores:
  1. Función `c()`: combina sus argumentos para formar un vector, intenta convertirlos al mismo tipo (si es posible).
  2. Función `vector()`: tiene dos argumentos, el `tipo` y la `longitud`.
  3. Funciones `numeric()`, `integer()`, etc.: igual que `vector()` pero tienen un único argumento, que es la longitud del vector.
- Para ver la longitud de un vector se utiliza `length()`.

Aritmética vectorial

- Los siguientes expresiones se pueden aplicar en vectores:
  - Operadores aritméticos: `+`, `-`, `*`, `/`, `^`, `%%` y `%/%`.
  - Funciones: `log`, `exp`, `sin`, `cos`, `tan` y `sqrt`.
  - Operadores lógicos: `&`, `!` y `|`.
- Realizan las operaciones elemento a elemento.
- Los vectores pueden ser de distintas longitudes: los vectores más cortos reciclan sus elementos hasta tener la longitud del más largo.
- Otras funciones que operan sobre vectores (no elemento a elemento): `sum()`, `prod()`, `max()`, `min()`, `range()`, `mean()`, `var()`, `sd()`, `cumsum()`, `cumprod()`, etc...

Sucesiones

- Operador ":". Genera números enteros en un rango, tiene máxima prioridad.

```
> 1:20
```

- Función `seq()`. Genera secuencias más complejas. Por ejemplo:

```
> seq(-0.5, 0.5, by=.2)
 [1] -0.5 -0.3 -0.1  0.1  0.3  0.5
```

- Función `rep()`. Duplica el objeto. Ejemplos:

```
> rep(1, 10)
 [1] 1 1 1 1 1 1 1 1 1 1
> rep(1:4, 2)
 [1] 1 2 3 4 1 2 3 4
```

Selección de subvectores

- Para seleccionar partes de un vector se utilizan vectores de índices entre corchetes `[` y `]`.
- Hay 4 tipos de vectores de índices:
  1. Vector lógico: se seleccionan los valores que son `TRUE`.
  2. Vector de enteros positivos: se seleccionan los elementos con esos índices.
  3. Vector de enteros negativos: se excluyen los elementos con esos índices.
  4. Vector de caracteres: solo si los elementos del vector tienen nombre, selecciona los elementos con dicho nombre.
- La variable de almacenamiento también puede ser indexada.
- Poner a 0 los elementos de `x` menores que 5:

```
> x[x < 5] <- 0
```

Ejercicio I

Ejecutar el siguiente código en R, que genera dos vectores de enteros aleatorios elegidos entre el 1 y el 1000 de tamaño 250:

```
> n <- 250
> x <- sample(1:1000, n, replace=T)
> y <- sample(1:1000, n, replace=T)
```

A partir de los dos vectores anteriores:

1. Calcular el el máximo y el mínimo de los vectores x e y.
2. Calcular la media de los vectores x e y. Antes de calcularla, ¿que valor esperarías?.
3. Calcula el número de elementos de x divisibles por 2 (el operador módulo es `%%`).

Ejercicio II

4. Ordenar los vectores, primero usando la función `order()` y luego la función `sort()`.
5. Seleccionar los valores de y menores que 600.
6. Crear el vector

$$(x_1 + 2x_2 - x_3,\ x_2 + 2x_3 - x_4, ...,\ x_{n-2} + 2x_{n-1} - x_n)$$

   Pista: tiene tamaño $n - 2$.

7. Crear el vector

$$\left(\frac{\cos(y_1)}{\sin(x_2)},\ \frac{\cos(y_2)}{\sin(x_3)},\ \ldots,\ \frac{\cos(y_{n-1})}{\sin(x_n)}\right)$$

   Pista: tiene tamaño $n - 1$.

Índice

Arrays

- Un `array` es una colección de datos del mismo tipo indexada por varios índices.
- La función `dim()` devuelve la longitud de cada una de las dimensiones del `array`.
- Un vector se puede transformar en un `array` modificando su atributo *dim*:

```
> x <- 1:60
> dim(x) <- c(3,5,4)
```

- Los `arrays` se crean con la función `array()` y sus datos se rellenan con el primer índice variando más rápido:

```
> z <- array(x, dim=c(3,5,4))
```

- La diferencia entre los métodos anteriores es que el primero evita duplicar el objeto.

Selección de subarrays I

- En general, se puede seleccionar una parte del `array` mediante una sucesión de vectores índice.

```
> z[1:2, -(1:4), 2]
[1] 28 29
```

- Si un vector índice está vacio se selecciona todo el rango de valores.

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

Selección de subarrays II

- Los arrays también se pueden indexar con otro array de índices.
- En el siguiente ejemplo, estamos seleccionando los elementos `z[1,1,3]` y `z[2,2,1]`

```
> idx <- array(c(1:2, 1:3), dim=c(2,3))
> idx
     [,1] [,2] [,3]
[1,]    1    1    3
[2,]    2    2    1
> z[idx]
[1] 31  5
```

Aritmética de arrays

- Los arrays y matrices se pueden emplear en operaciones aritméticas (elemento a elemento).
- A diferencia de los vectores, los arrays tienen que tener la misma longitud en todas las dimensiones.
- Las funciones `sum()`, `mean()`, `var()`, etc. se pueden aplicar en arrays, y al igual que en vectores calculan la suma, media, etc. de todos los elementos.
- Las funciones `colSums()`, `rowSums()`, `colMeans()`, `rowMeans()` son especiales para arrays y calculan la suma y media a lo largo de la dimensión especificada.
- La función `aperm()` permuta las dimensiones del array especificado.

Matrices

- Una `matriz` es un `array` de 2 dimensiones.
- Las matrices se crean con la función `matrix()`, y sus datos se rellenan por columnas (igual que en los arrays).
- Las funciones `nrow()`, `ncol()` devuelven el número de filas y columnas de forma respectiva.
- Además de todos los operadores de arrays, existen funciones específicas para matrices:

`t()`	devuelve la transpuesta de una matriz.
`diag()`	devuelve la diagonal de una matriz.
`crossprod()`	realiza el producto cruzado entre matrices (equivalente al operador `%*%`).
`eigen()`	calcula autovalores y autovectores.
`svd()`	realiza la descomposición en valores singulares.

Ejercicio I

1. Crear la matriz 4×5,

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \end{pmatrix}$$

   Pista: ver el parámetro `byrow` de la función `matrix()`.

2. Extraer los elementos $A[4, 3], A[3, 4], A[2, 5]$ utilizando una matriz de índices.
3. Reemplazar dichos elementos con 0.

Ejercicio II

4. Crear la matriz identidad 5×5,

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

   Pista: mirar documentación de la función `diag()`.

5. Convertir la matriz A anterior en una matriz cuadrada B añadiendo al final una fila de unos (función `rbind()`):

$$B = \begin{pmatrix} A \\ \mathbf{1} \end{pmatrix}$$

6. Calcular la inversa de la matriz B con la función `solve()`.

Ejercicio III

7. Multiplicar B por su inversa B^{-1}.
8. Comprobar si el resultado es exactamente la matriz identidad $\mathbf{I}$.
9. En caso contrario, calcular el "error" o "precisión" de la operación, definido como:

$$\text{Error} = \frac{1}{N} \sum_{i,j} |(BB^{-1} - \mathbf{I})_{(i,j)}|$$

donde N es el número de elementos de la matriz B.

Índice

Factores

- Un factor es un vector que contiene una clasificación discreta de sus elementos.
- Se usan para almacenar variables categóricas, por ejemplo `renta` = {alta, baja, media}.
- Se crean con la función factor:

```
> f <- factor(c("hombre", "mujer", "mujer"))
```

- Se pueden ver los niveles (valores distintos) con la función `levels()`:

```
> levels(f)
[1] "hombre" "mujer"
```

- Función `tapply()`: aplica una función a cada uno de los elementos de un vector, divididos en los distintos grupos de un determinado factor.

Listas

- Una lista consiste en una colección ordenada de objetos del mismo o distinto tipo.
- Los componentes de la lista siempre están numerados.
- Son accesibles con el operador doble corchete `[[` y `]]`:

```
> l <- list(nombre="Alberto", numeros=c(3,5,6))
> l[[1]]
[1] "Alberto"
```

- Los componentes también pueden tener nombre. En ese caso también son accesibles de la siguiente forma:

```
> l$nombre
[1] "Alberto"
> l[["numeros"]]
[1] 3 5 6
```

Operaciones sobre listas

- Se crean con la función `list()`.
- Se pueden modificar sus elementos con la sintaxis:

  ```
  > l$nombre <- "Juan"
  ```

- Si el nombre del elemento no existe, se añade a la lista:

  ```
  > l$edad <- 25
  ```

- También se pueden combinar con la función `c()` (similar a como se hacía con vectores).
- Importante destacar la diferencia entre `l[[1]]` y `l[1]`: el primero devuelve el componente mientras que el segundo devuelve una sublista.

Data frames

- Un data frame es una lista donde cada uno de sus elementos se conocen como variables.
- Además, tiene las siguientes restricciones:
  - Los componentes deben de ser vectores, factores, matrices, listas u otros data frames.
  - Las matrices, listas y data frames contribuyen al nuevo data frame con tantas variables como columnas, elementos o variables tengan.
  - Los vectores no numéricos se transforman en factores.
  - Todos los vectores tienen que tener la misma longitud y las matrices el mismo número de filas.
- Resumiendo, un data frame es como una matriz donde sus columnas pueden ser de tipos diferentes.

Operaciones con data frames I

- Por su condición de listas, se puede acceder a cada una de las variables del data frame de las siguientes formas:

```
> mtcars[[1]]
> mtcars[["mpg"]]
> mtcars$mpg
```

- El resultado es un vector del mismo tipo que el elemento del data frame.
- Para obtener un subconjunto de las columnas se pueden indexar numéricamente o por nombre:

```
> mtcars[1]
> mtcars["mpg"]
```

- Al igual que en las listas, el resultado de las operaciones anteriores es otro data frame.

Operaciones con data frames II

- Para obtener un subconjunto de las filas se pueden indexar numéricamente, por el nombre o usando un vector lógico:

```
> mtcars[1,]
> mtcars["Camaro Z28",]
> mtcars[mtcars$mpg > 30,]
```

- Por último, se pueden combinar las dos anteriores para acceder a un subconjunto de las filas y de las columnas:

```
> mtcars[4:8, -c(5:11)]
> mtcars["Camaro Z28", c("mpg", "gear")]
> mtcars[mtcars$mpg > 30, c(1,4)]
```

- El resultado de la operación es otro data frame, menos si es un único elemento ó se seleccionan todas las filas:

```
> mtcars[,1]
```

Funciones attach, detach y search

- La notación para acceder a variables del data frame no es siempre la más apropiada.
- Se puede utilizar la función `attach()` para acceder a variables del data a frame directamente por su nombre.
- Cuando se termina de trabajar con el data frame, hay que desconectar el data frame con `detach()`.
- La función `search()` lista los data frames que están conectados.

```
> attach(mtcars)
> mean(drat)
[1] 3.596563
> detach(mtcars)
```

Otras operaciones con datasets

- Añadir fila: `rbind()`
- Añadir columna (variable): `cbind()`
- Eliminar una variable

```
> mtcars$mpg <- NULL
```

- Renombrar una variable

```
> names(mtcars)[1] <- "miles/galon"
```

- Reordenar el data frame (mayor a menor):

```
> mtcars[order(mtcars$disp, decreasing=TRUE), ]
```

- Detectar *missing values*

```
> mtcars[5, "disp"] <- NA
> mtcars[complete.cases(mtcars), ]
```

Ejercicio

Con el data frame `mtcars` (viene cargado en R).

1. Previsualizar el contenido con la función `head()`.
2. Mirar el número de filas y columnas con `nrow()` y `ncol()`.
3. Crear un nuevo data frame con los modelos de coche que consumen menos de 15 millas/galón.
4. Ordenar el data frame anterior por `disp`.
5. Calcular la media de las marchas (`gear`) de los modelos del data frame anterior.
6. Cambiar los nombres de las variables del data frame a `var1`, `var2`, ..., `var11`.
   Pista: Mirar la documentación de la función `paste` y usarla para generar el vector ("var1", "var2", ..., "varN") donde N es el número de variables del data frame.

Ejercicio I

Con el data frame `iris` (viene cargado en R).

1. ¿Como está estructurado el data frame? (utilizar las funciones `str()` y `dim()`).
2. ¿De qué tipo es cada una de las variables del data frame?.
3. Utilizar la función `summary()` para obtener un resumen de los estadísticos de las variables.
4. Comprobar con las funciones `mean()`, `range()`, que se obtienen los mismos valores.
5. Cambia los valores de las variables `Sepal.Length` `Sepal.Width` de las 5 primeras observaciones por `NA`.
6. ¿Qué pasa si usamos ahora las funciones `mean()`, `range()` con las variables `Sepal.Length` y `Sepal.Width`? ¿Tiene el mismo problema la función `summary()`?

Ejercicio II

7. Ver la documentación de `mean()`, `range()`, etc. ¿Qué parámetro habría que cambiar para arreglar el problema anterior?.
8. Visto lo anterior, ¿por qué es importante codificar los *missing values* como `NA` y no como 0, por ejemplo?
9. Eliminar los valores `NA` usando `na.omit()`.
10. Calcular la media de la variable `Petal.Length` para cada uno de las distintas especies (`Species`). **Pista**: usar la función `tapply()`.

Índice

Scripts de R

- Si se re-utilizan secuencias de comandos con frecuencia, es conveniente guardarlas en un fichero `script.R`.
- El fichero se puede ejecutar posteriormente en R con

  ```
  > source("script.R")
  ```

- El intérprete de R ejecutará cada una de las líneas del fichero, pero sin imprimir el valor de ninguna variable en la consola.
- Para ello, son útiles las funciones `cat()` y `print()`, que imprimen por pantalla el valor de las variables pasadas como argumentos.
- Es importante destacar que el fichero `script.R` tiene que estar en el directorio de trabajo actual, de lo contrario hay que usar la ruta completa.

Cargar datos desde fichero

- Los datos suelen leerse desde archivos de texto externos.
- La función principal es `read.table(`*fichero*`)`, que lee el contenido de *fichero* y devuelve una variable de tipo `data.frame`.
- Existen otras como `read.csv()` o `read.delim()` pero simplemente llaman a `read.table()` con distintos parámetros por defecto.
- También se puede llamar directamente a la función de bajo nivel `scan()`.
- Es preferible editar el fichero de texto con un programa externo para que tenga un formato razonable a escribir código complicado en R para leerlo.
- Igual que en los scripts, el *fichero* tiene que estar en el directorio de trabajo o en su defecto escribir la ruta completa.

Función read.table() I

Tiene los siguientes parámetros opcionales:

header Si TRUE, la primera fila es el nombre de las variables.

sep Carácter que separa las columnas. Por defecto se separan por uno o más espacios en blanco o tabuladores. Para archivos CSV, poner `sep=","` o `sep=";"`.

na.strings Vector con cadenas que se interpretan como *missing values*. Por defecto `"NA"`. Por ejemplo: `na.strings=c("-", "-9999.0")`.

colClasses Vector de clases para las columnas. Por defecto todas las columnas se leen como texto y a continuación se convierten a valores lógicos, enteros, flotantes, números complejos o factores.

Función read.table() II

nrows Número máximo de líneas a leer del fichero.

comment.char Un vector con un único carácter que indica líneas a ignorar en el fichero (comentarios).

skip Número de líneas a ignorar al princpio del fichero.

dec Carácter para separar los decimales, "." por defecto. Útil cuando los decimales se separan con ",".

fill Si TRUE, en el caso de que las líneas tengan longitudes diferentes se añaden variables en blanco. Por defecto, el número de columnas se determina a partir de las 5 primeras líneas, y si no coincide en el resto se produce un error.

Tiempo de ejecución

- La función `system.time()` se puede usar para medir el tiempo de ejecución de una expresión de R.
- Devuelve un vector con tres valores:

user	Tiempo ejecutando código de usuario.
system	Tiempo realizando llamadas al sistema (por ejemplo, operaciones entrada/salida).
elapsed	Tiempo real que tardó la expresión en ejecutarse.

- Generalmente, nos interesa el tercer valor.
- Para medir el tiempo de un segmento de código arbitrario, se puede utilizar `proc.time()`.
- Esta función se usa realizando dos llamadas, una al principio del segmento y otra al final, y después se restan los valores.

Lectura de ficheros: ejemplos y eficiencia

```
> system.time(data1 <- read.table("./data/E2006.train"))
    user   system  elapsed
1944.052   32.996 1990.725

> system.time(data2 <- scan("./data/E2006.train"))
Error in scan(file, what, nmax, sep, dec, quote, skip,
nlines, na.strings,  :
  too many items
Timing stopped at: 387.984 386.219 775.598

> system.time(data3 <- read.table("./data/E2006.train",
  nrows=16087, colClasses=c("numeric"), comment.char=""))
    user   system  elapsed
1110.955   22.027 1135.709
```

```
> system.time(data1 <- read.table("./data/E2006.test"))
   user  system elapsed
365.426   1.714 367.749

> system.time(data2 <- read.table("./data/E2006.test",
  nrows=3308, colClasses=c("numeric"), comment.char=""))
   user  system elapsed
256.677   2.025 259.244

> system.time(data3 <- scan("./data/E2006.test"))
Read 497394188 items
   user  system elapsed
 93.887   3.807  97.881

> system.time(data4 <- as.data.frame(matrix(data3,
  ncol=150361, byrow=TRUE)))
   user  system elapsed
 15.384   0.003  15.404
```

Índice

Ejecución condicional

- La sintaxis de la construcción condicional es:
  `> if (`*cond*`)` *expr_true* `else` *expr_false*
- *cond* es cualquier expresión de R cuyo resultado es un valor lógico.
- A menudo se usan los operadores `&&` (and lógico) y `||` (or lógico) para combinar varias condiciones.
- Si se utilizan con vectores de tamaño mayor que 1, solo miran el primer elemento.
- A diferencia de los anteriores, `&` y `|` operan elemento a elemento de un vector.
- La función `ifelse()` es una versión vectorizada de la construcción condicional.

Bucles

- Hay tres tipos de bucles:

  for Se ejecuta un número fijo de veces:
  `> for (`*var* `in` *seq*`)` *expr*

  while Se ejecuta mientras se cumple una condición:
  `> while(`*cond*`)` *expr*

  repeat Se ejecuta indefinidamente:
  `> repeat` *expr*

- La instrucción `break` termina cualquiera de los tres bucles anteriores.
- La instrucción `next` pasa a la siguiente iteración.
- Una expresión puede contener a su vez un número arbitrario de expresiones, en cuyo caso se agrupan entre llaves {*expr_1* ; ... ; *expr_n*}

Funciones

- R permite crear objetos de tipo `function` que constituyen nuevas funciones del lenguaje.
- Para definir una función hay que hacer una asignación de la forma:

  `> nombre_fun <- function(`*arg1* `,` *arg2* `, ...)` *expr*
- Típicamene, las funciones devuelven un valor con la expresión `return(`*value*`)`.
- Si se llega al final de la función y no hubo ningún `return`, se devuelve el valor de la última expresión.
- Ejemplo:

```
> media <- function(vector) {
    ret <- sum(vector)/length(vector)
    return(ret)
  }

> m <- media(1:10000)
```

Argumentos con nombre. Valores predeterminados

- Se pueden pasar los argumentos a una función dado su nombre, en cuyo caso el orden es irrelevante:

```
> myfun <- function(datos, resp, impr) {}
```

- Las siguientes llamadas a `myfun` son equivalentes:

```
> myfun(X, y, TRUE)
> myfun(datos=X, resp=y, impr=TRUE)
> myfun(impr=TRUE, resp=y, datos=X)
```

- También se pueden definir valores por defecto, en cuyo caso se pueden omitir cuando se llama a la función:

```
> myfun <- function(datos, resp, impr=TRUE) {}
```

- Las siguientes llamadas también son equivalentes:

```
> myfun(X, y)
> myfun(resp=y, datos=X)
```

Ejercicio

- Cargar el fichero `test` en R.
- Escribir una función `myMean` que calcule la media de un vector x que se le pasa como argumento utilizando un bucle `for`. La función tiene que devolver el valor de la media.
- Aplicar la función a los datos del fichero `test` y comprobar que la función es correcta comparando el resultado con la función `mean()`.
- Escribir una función `difTiempos` que tenga como argumento un entero n, genere el vector $1, \ldots, n$ y devuelta la diferencia del tiempo de ejecución entre `myMean` y `mean`. La función también tiene que comprobar que n es positivo, en caso contrario devolverá el valor 0.
- Llamar a `difTiempos` para distintos valores de n (10^5, 10^6, 10^7, ...) y ver como evoluciona la diferencia de tiempos.

Índice

Estadística y muestreos aleatorios

- Las funciones para calcular estadísticos de una muestra son: `mean` (media), `var` (varianza), `sd` (desviación típica), `quantile` (cuantiles), `max` (máximo), `min` (mínimo), `range` (rango), `cov` (covarianza) y `cor` (correlación).
- También existe la función genérica `summary()`, que devuelve un resumen del objeto que se le pasa como primer argumento.
- Para generar muestreos aleatorios y permutaciones se utiliza la función `sample()`. Está función también sirve para generar números enteros aleatorios.
- Otras funciones útiles son `unique()` y `table()`, que calculan el número de elementos únicos y cuantos hay de cada uno de ellos, respectivamente.

Distribuciones de probabilidad

- R contiene un amplio conjunto de tablas estadísticas.
- Para cada una de ellas, está disponible:
  - La función de densidad (pdf), añadiendo `d`.
  - La función de distribución (cdf), añadiendo `p`.
  - La función de distribución inversa, añadiendo `q`.
  - Generación de números pseudo-aleatorios, añadiendo `r`.
- Las más comunes son: `beta`, `binom`, `cauchy`, `gamma`, `lnorm`, `norm`, `pois`, `t`, y `unif`.
- Ejemplo: para la distribución normal tenemos las funciones `dnorm`, `pnorm`, `qnorm` y `rnorm`.
- Otras funciones útiles para estudiar la distribución de unos datos son `density()`, para estimar la función de densidad y `ecdf()`, para calcular la función de distribución empírica.

Índice

Gráficos

- Los comandos gráficos en R se dividen en 3 grupos:
  - Alto nivel: funciones que crean un nuevo gráfico, con ejes, etiquetas, títulos, etc.
  - Bajo nivel: funciones que añaden elementos a un gráfico ya existente, como puntos, líneas, etc.
  - Interactivas: funciones que permiten interactuar con un gráfico.
- Para exportar un gráfico a un fichero, por ejemplo en PDF, hay que inicializar el dispositivo gráfico con

  `> pdf("fig.pdf")`

  a continuación realizar el gráfico y por último cerrarlo con

  `> dev.off()`.
- Alternativamente, desde RStudio se puede exportar con la opción "Export" de la interfaz gráfica.

Función plot()

- Es genérica, es decir, el tipo de gráfico que produce depende de la clase del primer argumento.
- La forma con un argumento, `plot(x)`, produce:
  - Factor: diagrama de barras.
  - Vector numérico: gráfico de sus elementos sobre el índice.
  - Matrix: diagrama de dispersión de la segunda columna sobre la primera.
  - Data frame: diagramas de dispersión de todos los pares de variables.
- La forma con dos argumentos, `plot(x,y)`, produce:
  - Vectores numéricos: gráfico de dispersión de `y` sobre `x`.
  - `x` factor, `y` vector numérico: diagrama de cajas de `y` para cada factor de `x`.
- Para ver los métodos para otros tipos de objetos se usa `methods(plot)`.

Funciones de alto nivel

- Gráficos cuantil-cuantil, que sirven para comparar dos distribuciones:
  - `qqnorm(x)`, compara una muestra con la distribución normal.
  - `qqline(x)`, añade una línea de referencia.
  - `qqplot(x,y)`, compara dos muestras `x` e `y`.
- Histogramas: representa el vector numérico `x` en forma de barras, `hist(x)`.
- Gráficos 3D:
  - `image(x,y,z)`, rejilla de rectángulos con colores que se corresponden a los valores en `z` (*heatmap*).
  - `contour(x,y,z)`, curvas de nivel de `z`.
  - `persp(x,y,z)`, superficie 3D de `z` sobre el plano x-y.
- Diagramas de cajas y "bigotes", `boxplot(x)`.
- Curvas de funciones, `curve(expr)`.

Argumentos de las funciones de alto nivel

`add`	Si TRUE, hace que la función se comporte como una de bajo nivel y se añada el gráfico actual.
`axes`	Si FALSE no se dibujan los ejes. Útil para dibujar tus propios ejes.
`log`	Represente ejes en escala logarítmica, valores posibles: "x", "y" o "xy".
`type`	Tipo del gráfico, valores posibles: "p" (puntos), "l" (líneas), "o" (puntos y líneas), etc.
`xlab`	Etiqueta del eje x.
`ylab`	Etiqueta del eje y.
`xlim`	Límite del eje x.
`ylim`	Límite del eje y.
`main`	Título del gráfico.
`sub`	Subtítulo del gráfico.

Funciones de bajo nivel

`points(x,y)`	Puntos con coordenadas (x,y).
`lines(x,y)`	Linea que pasa por todos los puntos (x,y).
`abline(a,b)`	Recta con pendiente `a` y ordenada `b`.
`abline(h=y)`	Recta horizontal en el punto y.
`abline(v=x)`	Recta vertical en el punto x.
`polygon(x,y)`	Polígono cuyos vértices son los elementos de (x,y).
`text(x,y,lab)`	Texto en las coordenadas (x,y).
`legend(pos,leg,...)`	Leyenda en la posición especificada.
`title(main,sub)`	Título principal y subtítulo.
`axis(lado,...)`	Eje en el lado indicado por argumento, de 1 a 4.

Ejercicio

- Generar 10000 números aleatorios con una distribución normal estándar (media 0 y varianza 1).
- Realizar un histograma de los valores anteriores. ¿Cual es el menor y mayor valor generado?.
- Calcular el valor teórico de la distribución normal en el intervalo anterior, utilizando la función `dnorm()`.
- Al histograma anterior, superponer una curva con la función de densidad teórica calculada. ¿Se aproxima el histograma al valor teórico?
- Ver el parámetro `probability` de la función `hist` y volver a generar el histograma cambiando su valor. ¿Se aproxima ahora a la función de densidad teórica?.
- Superponer al gráfico anterior la función de densidad empírica que aproxima al histograma (función `density`) en color rojo.

Parámetros gráficos

- Muchos aspectos de los gráficos se pueden personalizar a través de opciones.
- Estas opciones se pueden especificar de forma temporal a través de las funciones de alto nivel o hasta que se termine la sesión con la función `par()`.
- Algunas de las opciones más usadas son:

`pch`	Carácter que se utiliza para dibujar un punto.
`lty`	Tipo de línea.
`lwd`	Anchura de la línea, en múltiplos de la anchura base.
`col`	Color de los textos, líneas, texto, imágenes.
`font`	Fuente que se utiliza para el texto.
`cex`	Tamaño del texto y los elementos gráficos.

- Para más información sobre los valores que pueden tomar las opciones anteriores `help(par)` o esta web.

Gráficos múltiples

Hay varias formas de crear gráficos múltiples:

1. Función `par()`, modificando las opciones:
   - `mfrow=c(`n`, `m`)`, crea una matriz de $n \times m$ figuras que se va rellenando por filas.
   - `mfcol=c(`n`, `m`)`, crea una matriz de $n \times m$ figuras que se va rellenando por columnas.
   - `fig=c(`x_1`, `x_2`, `y_1`, `y_2`)`, coloca la figura dentro del rectángulo definido por las coordenadas (x_1, y_1) y (x_2, y_2). Estas coordenadas tienen un valor entre 0 y 1, donde (0, 0) es la esquina superior izquierda. Esta opción crea un nuevo gráfico, por lo que es necesario combinarla con `new=TRUE`.
2. Función `layout(`m`)`, donde m es una matriz que contiene la localización de cada figura. Las filas y columnas de la rejilla de figuras pueden tener distinto tamaño (opciones `widths` y `heights`).

Índice

Modelos lineales

- Dado el siguiente modelo lineal

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon},$$

  donde $\mathbf{X} \in \mathbb{R}^{n \times p}$ es la matriz de datos, $\mathbf{y} \in \mathbb{R}^n$ el vector de la respuesta y $\boldsymbol{\epsilon} \sim N(0, \sigma)$ el ruido.
- Podemos definir en R su fórmula con la sintaxis

$$y \sim x_1 + x_2 + \cdots + x_p$$

- Para ajustar un modelo lineal por mínimos cuadrados usamos la función `lm()`.
- Ejemplo: modelo de regresión de la variable `mpg` sobre `wt`

```
> fit <- lm(mpg ~ wt, data=mtcars)
```

Instalar paquetes de CRAN

- Se pueden instalar nuevos paquetes con el comando
  `> install.packages("`*nombre_paquete*`")`
- Para cargarlo en el entorno de R se usa el comando
  `> library(`*nombre_paquete*`)`
- Alternativamente, en RStudio se puede hacer el procedimiento anterior de forma gráfica.
- Para instalar un paquete, hay ir a la pestaña "Packages" y hacer click en "Install".
- Los paquetes instalados aparecen en la lista inferior, para cargarlos basta con seleccionarlos en el recuadro de la izquierda.

Proyecto I

Con el conjunto de datos `housing`

- Cargar los datos en R.
- Realizar un análisis estadístico de las variables: calcular la media, varianza, rangos, etc. ¿Tienen las distintas variables rangos muy diferentes?.
- Escalar los datos para que tengan media 0 y varianza 1, es decir, restar a cada variable su media y dividir por la desviación típica.
- La variable de respuesta se encuentra en la primera columna, separarla del resto y calcular la correlación de dicha variable con el resto.
- Separar el conjunto de datos en dos, el primero (entrenamiento) conteniendo un 80% de los datos y el segundo (test) un 20%, de forma aleatoria.

Proyecto II

- Realizar un modelo de regresión lineal de la variable de respuesta sobre el resto y ajustarlo por mínimos cuadrados usando **únicamente** los datos del conjunto de entrenamiento.
- Calcular el error cuadrático medio de los datos del conjunto de entrenamiento y de los datos del conjunto de test, definido como

$$\text{ECM} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

donde y es el vector de respuesta de los datos y $\hat{y}$ es el vector que predice el modelo (para los mismos datos).

Más información sobre R

Existen varias comunidades de usuarios de R muy activas, donde se puede escribir en el caso de tener algún problema:

- StackOverflow: las preguntas con el tag R contienen mucha información y problemas resueltos. Además, las nuevas preguntas se responden en cuestión de horas.
- CrossValidated: no es una comunidad específica de R (más bien de estadística), pero hay mucha información acerca de cómo realizar procedimientos concretos de análisis de datos y aprendizaje automático en R.
- High-Performance computing in R Colección de paquetes de R útiles para la computación de alto rendimiento en R.
- @RLangTip Twitter que publica consejos y trucos diarios sobre R.

Gracias!

alberto.torres@uam.es