Práctica 1: Introducción a R

Estadística

Grado en Inteligencia Artificial (UDC)

Índice

1. Introducción a R

En las prácticas de esta asignatura se empleará `R`, un lenguaje de programación interpretado y un entorno estadístico desarrollado específicamente para el análisis de datos.

Para instalar R se recomienda seguir los pasos descritos en el post *https://rubenfcasal.github.io/post/instalacion-de-r*. Para el desarrollo de código e informes se sugiere emplear *RStudio Desktop*, que se puede instalar y configurar siguiendo las indicaciones proporcionadas en el post *https://rubenfcasal.github.io/post/instalacion-de-rstudio*.

Las prácticas se ha generado empleando RMarkdown (Quarto es la nueva versión), una extensión de Markdown que permite incorporar código y resultados de R. RMarkdown/Quarto es recomendable para difundir análisis realizados con R (y Python) en formato HTML, PDF y DOCX (Word), entre otros. Para una introducción a RMarkdown puede ser de utilidad consultar el Capítulo 3 *Generación de informes* del libro Fernández-Casal (2023): *Notas de Programación en R* (github).

En esta práctica se emplea como referencia el libro:

- Fernández-Casal R., Roca-Pardiñas J., Costa J. y Oviedo-de la Fuente M. (2022). *Introducción al Análisis de Datos con R* (github).

Las siguientes secciones son una selección esquemática de los contenidos más importantes. También puede resultar de utilidad el post *Ayuda y recursos para el aprendizaje de R*.

1.1. Introducción

Ver el Capítulo 1 del libro de referencia.

1.1.1. El lenguaje y entorno estadístico `R`

Ver secciones 1.1 y 1.3 *El entorno de desarrollo RStudio Desktop*.

Por defecto RStudio está organizado en cuatro paneles:

- Editor de código (normalmente un fichero *.R* o *.Rmd*).
- Consola de R (y terminal de comandos del sistema operativo).

- Explorador del entorno e historial.
- Explorador de archivos, visor de gráficos, ayuda y navegador web integrado.

Primeros pasos:

- Presionar *Ctrl-Enter* (*Command-Enter* en OS X) para ejecutar la línea de código actual o el código seleccionado (también se puede emplear el botón *Run* en la barra de herramientas del Editor o el menú *Code*).
- Presionar *Tab* para autocompletado. Tecla *Arriba* para recuperar líneas de instrucciones.
- Pulsar en el nombre del objeto en la pestaña *Environment*, o ejecutar `View(objeto)` en la consola, para visualizar el objeto en una nueva pestaña del editor.

Se recomienda emplear siempre la codificación por defecto UTF-8[1].

Información adicional:

- RStudio cheatsheet
- Using the RStudio IDE

1.1.2. Interfaz de comandos

Ver Sección 1.2.

Normalmente se trabaja en `R` de forma interactiva empleando una **interfaz de comandos** donde se teclean las instrucciones que se pretenden ejecutar. Para ejecutar una línea de instrucciones hay que pulsar *Retorno* (y por defecto se imprime el resultado).

- Lenguaje interpretado orientado a objetos
- Instrucciones (expresiones): **funciones operando sobre objetos**
- Símbolo del sistema: `>` (a la espera de instrucciones; `+` si no se completó algún comando)
- Separador de instrucciones: habitualmente un cambio de línea. También punto y coma `;`
- Los comentarios empiezan por `#`
- Se agrupan instrucciones con `{ }`
- Si una instrucción es solo una expresión, se evalúa y se imprime el resultado.
- Operadores de asignación: `<-`, `=` (no se recomienda)
- Distingue entre mayúsculas y minúsculas

1.1.3. Una primera sesión

Ver Sección 1.5.

```
3 + 5
```

```
## [1] 8
```

```
sqrt(16)  # Raíz cuadrada de 16
```

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

```
a <- 3 + 5
a
```

[1] En versiones antiguas de RStudio puede no ser la opción por defecto y habría que establecerla, empleando el menú *Tools > Global Options > Code > Saving > Default text encoding: UTF-8*, o al emplear los menús *File > Reopen with Encoding…* o *File > Save with Encoding…*.

```
## [1] 8
```

```
A <- "casa"; A
```

```
## [1] "casa"
```

```
a
```

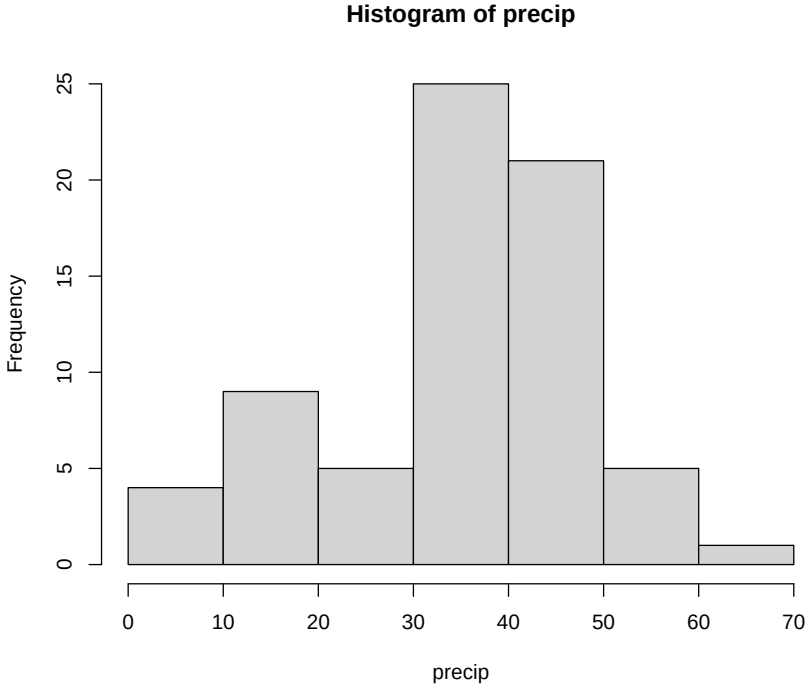
```
## [1] 8
```

Ejemplo de un análisis exploratorio muy básico (de una variable numérica), considerando los datos de lluvia almacenados en `precip` (disponible en el paquete base de `R`):

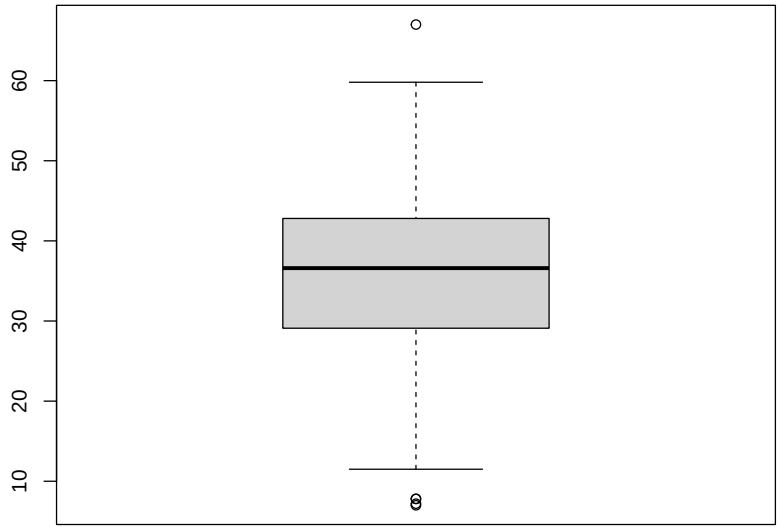
```
data(precip)    # Datos de lluvia
# ?precip       # Mostrar ayuda?
# precip        # Imprimir?
summary(precip) # Resumen estadístico
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    7.00   29.38   36.60   34.89   42.77   67.00
```

```
hist(precip)    # Histograma
```




```
boxplot(precip) # Gráfico de cajas
```



1.1.4. Ayuda

Ver Sección 1.4.

Se puede acceder a la ayuda empleando el entorno de comandos o el menú *Help* de RStudio (también en la pestaña *Help* del panel del visor o pulsando *F1*).

Función	Descripción
`help.start()`	inicia la ayuda HTML
`help(function)` `?function`	ayuda de una función (también de otros objetos o temas)
`help.search()`	realiza una búsqueda en la ayuda

1.1.5. Ejercicio

Calcular la cuota mensual R de devolución de un préstamo de un capital $P = 1500$ a un interés mensual $i = 0{,}05$ en $n = 10$ meses:

$$R = P\frac{i}{1-(1+i)^{-n}}$$

1.1.6. Funciones y librerías (paquetes)

Ver Sección 1.6.

Al iniciar el programa `R` se cargan por defecto una serie de librerías (denominadas paquetes) con las que se pueden realizar una gran cantidad de operaciones empleando las funciones que implementan. Estas librerías conforman el llamado **paquete base**.

Se llama a una función pasando los argumentos (valores de los parámetros) entre paréntesis.

Ejemplos:

```r
x <- sin(pi/2)
# La función `sin()` y el objeto `pi` están en el paquete base
cat("El objeto x contiene:", x, "\n")
```

```
## El objeto x contiene: 1
```

```r
ls()          # Lista objetos en el entorno de trabajo
```

```
## [1] "a"      "A"      "precip" "x"
```

Nota: una función es un tipo de objeto (probar a ejecutar `ls` en lugar de `ls()`).

Los parámetros de la función:

- Pueden tener nombres y valores por defecto
- Se asignan por posición (en el mismo orden en el que están definidos en la función) o por nombre (en cualquier orden y no necesariamente con el nombre completo)
- No se hace ninguna evaluación/chequeo hasta que se usan (*lazy evaluation*).
- `...` aglutina los argumentos no definidos explícitamente (cuando operan sobre múltiples argumentos o para poder añadir parámetros de otra función a la que se llama internamente).

Algunas funciones se comportan de manera diferente dependiendo del tipo de objeto (la clase) de sus argumentos, son lo que se denomina *funciones genéricas*. Entre ellas `summary()`, `print()`, `plot()` (por ejemplo, al ejecutar `methods(plot)` se muestran los métodos asociados a esta función; el método por defecto es `plot.default()`).

Se pueden utilizar muchos otros paquetes además de los "paquetes base", solo hay que cargarlos en memoria (si no es un paquete "recomendado", habrá que instalarlo previamente).

Función	Descripción
`install.packages("paquete")`	instala paquetes (CRAN…)
`library(paquete) require(paquete)`	carga paquetes
`library(help = "paquete")`	ayuda paquete
`demo(paquete)`	demo
`example(funcion)`	ejecuta ejemplos de la ayuda

1.1.7. Objetos básicos

Ver Sección 1.7.

Objetos básicos (atómicos):

- numérico (real, single, **double**, integer, complex)
- lógico (`TRUE`, `FALSE`)
- carácter

Operadores:

- Aritméticos: `+`, `-`, `*`, `/`, `%%` (módulo), `%/%` (división entera), `^` (potencia)
- Lógicos: `!A` (not), `A & B` (and), `A | B` (or), `xor(A,B)`
- Relacionales: `<`, `>`, `==`, `>=`, `<=`, `!=`

Valores especiales:

- `NA` (no disponible), `NaN` (no es un número), `Inf` (infinito)

- `NULL`, `character(0)`, …

Por defecto utiliza doble precisión (64bits), aunque al visualizarlos se redondean los decimales.

```
a <- 49 * (4/49)
a
print(a, digits = 3)
a == 4
# identical(a, 4)
sprintf("%.20f",a)
all.equal(a, 4)
```

1.1.8. Área de trabajo

Ver Sección 1.8.

El conjunto de objetos cargados en memoria se denomina espacio/entorno de trabajo (workspace).

- Al finalizar la sesión R pregunta (por defecto) si se desea guardar una imagen del área de trabajo (en el fichero *.RData*; ver `save.image()`)
- Si al iniciar R hay un fichero "*.RData*" en el directorio de trabajo se recupera la sesión anterior (objetos e historial, y muestra un aviso *[Previously saved workspace restored]*).

```
ls()          # Lista objetos en el entorno de trabajo
rm(x)         # Elimina el objeto x
rm(list=ls()) # Borra todos los objetos en el entorno de trabajo
getwd()       # Directorio de trabajo
dir()         # Lista los ficheros en el directorio de trabajo
# setwd("c:/") # Cambia el directorio de trabajo
```

Es recomendable utilizar un directorio de trabajo diferente para cada tarea (menú *Session > Set Working Directory…* o pestaña *Files > More…* del panel del visor).

Para reutilizar en sesiones próximas los objetos también podemos guardarlos nosotros (en el formato por defecto de datos de R, *file.RData*, o en el que nos interese, desde texto plano, *.csv*, o *.xlsx*, etc.). Ver Sección 1.8.1 *Guardar y cargar objetos.*

1.2. Estructuras de datos

Para más detalles ver el Capítulo 2 del libro de referencia.

Principales tipos de objetos (para datos):

- **vectores**, factores
- matrices, arrays
- listas
- data-frames

Realmente hay muchos tipos de objetos:

- funciones, formulas, expresiones…
- resultados de procedimientos (e.g. modelos ajustados, contrastes de hipótesis…)
- tipos especiales de datos (e.g. ts, "time series")
- `str(objeto)` muestra la e**str**uctura de un objeto

1.2.1. Vectores

Ver Sección 2.1

- Componentes del mismo tipo (conversión automática).
- La función `c()` (concatenar) es la forma más sencilla de crear un vector.
- Se pueden generar secuencias con `from:to`, `seq()` (diferentes alternativas) o `rep()`.

```
a <- c(10, 20)
a <- c(a, 30)
a
```

```
## [1] 10 20 30
```

```
1:21; seq(1, 21) # Instrucciones equivalentes.
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
```

```
seq(1, 21, by = 2)
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19 21
```

```
seq(1, 21, length = 6)
```

```
## [1]  1  5  9 13 17 21
```

```
rep(3, 12)
```

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

```
rep(c(1, 4), 3)
```

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

```
rep(c(1, 4), c(3, 2))
```

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

```
rep(c(1, 4), each = 3)
```

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

- En ocasiones se definen explicitamente `x <- tipo(n)`

```
x <- numeric(10)
```

Operaciones vectorizadas:

- Reciclaje: el vector más corto se reutiliza.

```
a <- 1:10
2 + a
```

```
##  [1]  3  4  5  6  7  8  9 10 11 12
```

```
2*a
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
a > 5
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
a*a; a^2
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

```
b <- c(0, 1)
a*b
```

```
##  [1]  0  2  0  4  0  6  0  8  0 10
```

Indexado de vectores:

- `x[i]` elemento i-ésimo
- `x[vector]` selección según el tipo de `vector` (enteros positivos/índices, enteros negativos, lógico)

```
x <- 0:9
x[1]
```

```
## [1] 0
```

```
x[1:5]
```

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

```
x[-1]
```

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

```
x[x > 0]
```

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

```
x[c(FALSE, TRUE)]
```

```
## [1] 1 3 5 7 9
```

```
x[1:5] <- 0
x
```

```
##  [1] 0 0 0 0 0 5 6 7 8 9
```

- También se pueden asociar nombres/etiquetas a las componentes y emplearlos para seleccionar componentes:

```
names(x) <- letters[1:10]
x
```

```
## a b c d e f g h i j
## 0 0 0 0 0 5 6 7 8 9
```

```
x[c("a", "f")]
```

```
## a f
## 0 5
```

Funciones básicas

- `length(x)` número de componentes de `x`
- `sum(x)`, `prod(x)` suma/producto de las componentes
- `min(x)`, `max(x)` menor/mayor componente de `x`
- `range(x)` = `c(min(x), max(x))`

- `which.min(x)`, `which.max(x)` índice del menor/mayor componente
- `which(logical)` vector de índices correspondientes a `TRUE`
- `rev(x)` invierte las componentes de `x`
- `sort(x)` ordena las componentes de `x` (`decreasing = FALSE`)
- `unique(x)` componentes con valores no repetidos

Al ser R un lenguaje con orientación estadística, no solo tiene predefinidas las funciones estándar, sino también una gran cantidad de funciones estadísticas. Por ejemplo: `mean()`, `sd()`, `var()`, `summary()`.

```
x <- 1:11
n <- length(x)
sum(x)/n
```

```
## [1] 6
```

```
mean(x)
```

```
## [1] 6
```

1.2.2. Ejercicio

Modificar el código del Ejercicio 1.1.5 para calcular las cuotas mensuales de devolución del préstamo correspondientes a la secuencia de intereses $i = 0{,}01k$ con $k = 1, \ldots, 10$.

Nota: más adelante (Sección Programación) veremos que puede ser más cómodo modificar el código para crear una función.

1.2.3. Factores

Ver Sección 2.1.7.

Los emplearemos en estadística descriptiva para variables cualitativas (categóricas).

1.2.4. Matrices y arrays

Ver Sección 2.2.

1.2.5. Data frames

Ver Sección 2.3.

Data frame: es una estructura de datos en forma de tabla en la que cada columna representa una variable. Las columnas pueden ser de tipos diferentes y tienen nombre. Por ejemplo `iris` (un objeto del paquete base de R que contiene datos muy famosos en estadística; ver `?iris`):

```
str(iris)    # estructura
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
```

```
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
# View(iris)
```

La función `names(datos)` lista los nombres de las columnas (variables).

Indexado de data frames (permite las opciones de matrices y de listas):

- `data$nombre`, `data[[i]]`, `data[["nombre"]]` selecciona la correspondiente variable (un vector o un factor).
- `data[index]`, `data[, index]` selecciona un conjunto de variables (un data.frame) de forma análoga al caso de vectores.
- `data[i, j]` observación `i` de la variable `j`.
- `data[index, ]` selecciona un conjunto de observaciones (filas).

Ejemplo:

```
iris[1:2]
iris[c("Sepal.Length", "Sepal.Width")]
iris$Species
iris[ ,"Species"]
iris[iris$Species == "setosa", ]
subset(iris, Species %in% c("setosa", "versicolor"))
```

1.2.6. Listas

Ver Sección 2.4.

`list()`: Las listas son estructuras heterogéneas de datos en la cada uno de sus elementos puede ser de diferentes tipos (vectores, matrices, arrays, data frames, listas).

```
x <- 2:4
y <- seq(2, 6, by = 2)
a <- list(x, y, z = c(2, 4, 5))
a
names(a)
a[[1]]
a$z
```

Un `data.frame` no es más que una lista cuyas componentes son vectores de la misma longitud (realmente las componentes también pueden ser listas y podemos almacenar casi cualquier estructura de datos).

1.3. Gráficos

Para más detalles ver el Capítulo 3 del libro de referencia.

En el paquete base de `R` se dispone de múltiples funciones que permiten la generación de gráficos (los denominados gráficos estándar). Se dividen en dos grandes grupos:

- Gráficos de **alto nivel**: Crean un gráfico nuevo.
  - `plot`, `hist`, `boxplot`, `curve`, ...

- Gráficos de **bajo nivel**: Permiten añadir elementos (líneas, puntos, …) a un gráfico ya existente
  - `points`, `lines`, `legend`, `text`, …

El parámetro `add = TRUE` convierte una función de nivel alto a bajo.

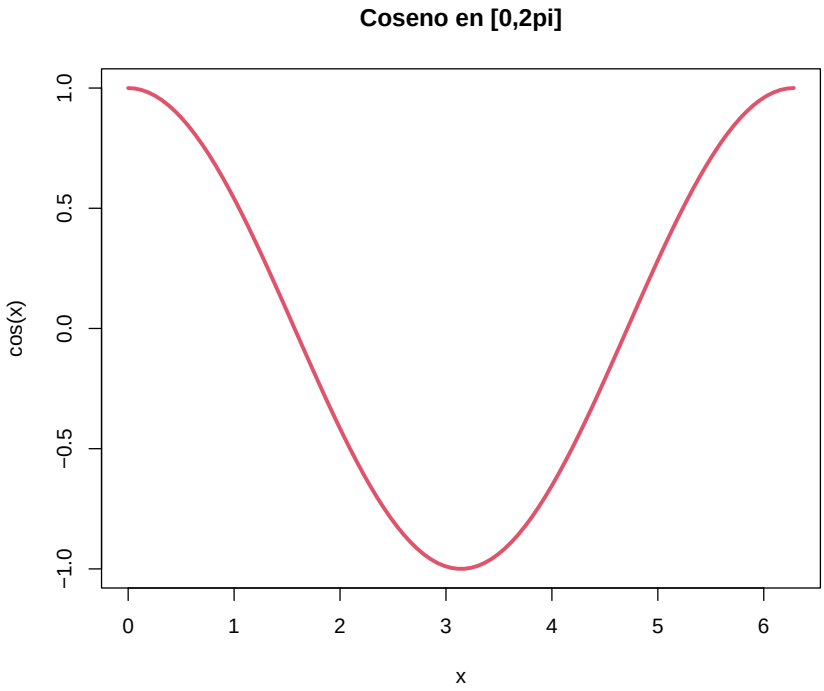
Dentro de las funciones gráficas de alto nivel destaca la función `plot()` (*función genérica*).

1.3.1. La función plot

Ver Sección 3.1.

Si ejecutamos `plot(x, y)` siendo `x` e `y` vectores, entonces `R` generará el denominado **gráfico de dispersión** que representa en un sistema coordenado los pares de valores (x, y).

```
x <- seq(0, 2 * pi, length = 100)
y <- cos(x)
plot(x, y, type = "l", col = 2, lwd = 3, ylab = "cos(x)", main = "Coseno en [0,2pi]")
```



Los principales parámetros son:

Parámetro	Descripción
`type`	tipo de gráfico: `p`: puntos, `l`: líneas, `b`: puntos y líneas, `n`: gráfico en blanco…
`xlim`, `ylim`	límites de los ejes (e.g. `xlim=c(1, 10)` o `xlim=range(x)`)
`xlab`, `ylab`	títulos de los ejes
`main`, `sub`	título principal y subtítulo
`col`	color de los símbolos (véase `colors()`). También `col.axis`, `col.lab`, `col.main`, `col.sub`
`lty`	tipo de línea
`lwd`	anchura de línea

Parámetro	Descripción
`pch`	tipo de símbolo
`cex`	tamaño de los símbolos

Muchas de estas opciones se emplean en otras funciones gráficas. Con la función `par()` se pueden obtener y establecer (de forma permanente) todas las opciones gráficas (ver secciones 3.4 *Parámetros gráficos* y 3.5 *Múltiples gráficos por ventana*). Ejecutar `help(par)` para obtener la lista completa.

1.3.2. Funciones gráficas de bajo nivel

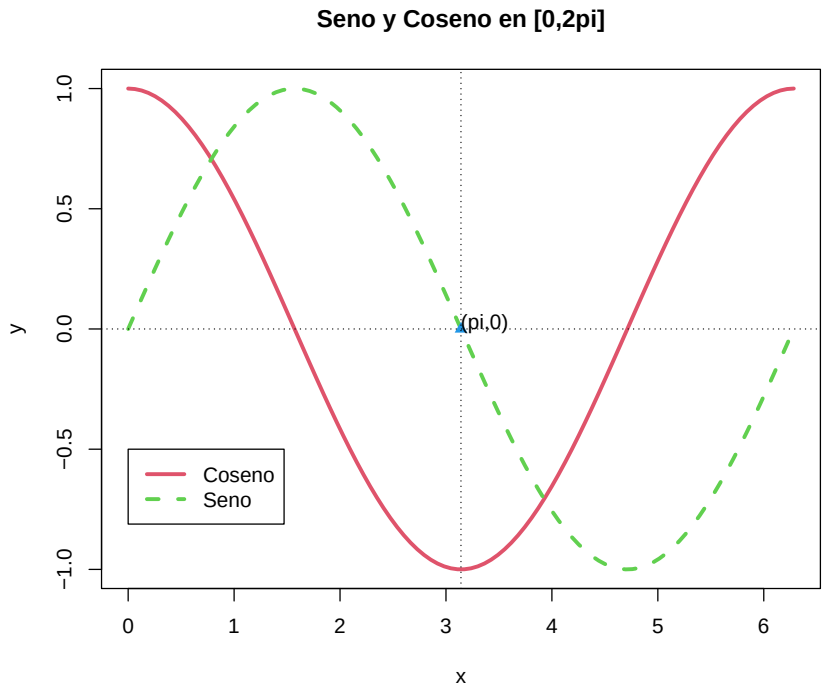
Ver Sección 3.2.

Algunas de las funciones gráficas de bajo nivel más empleadas son:

Función	Descripción
`points`, `lines`	agregan puntos y líneas
`text`	agrega un texto
`abline`	dibuja líneas
`legend`	agrega una leyenda
`axis`	agrega ejes

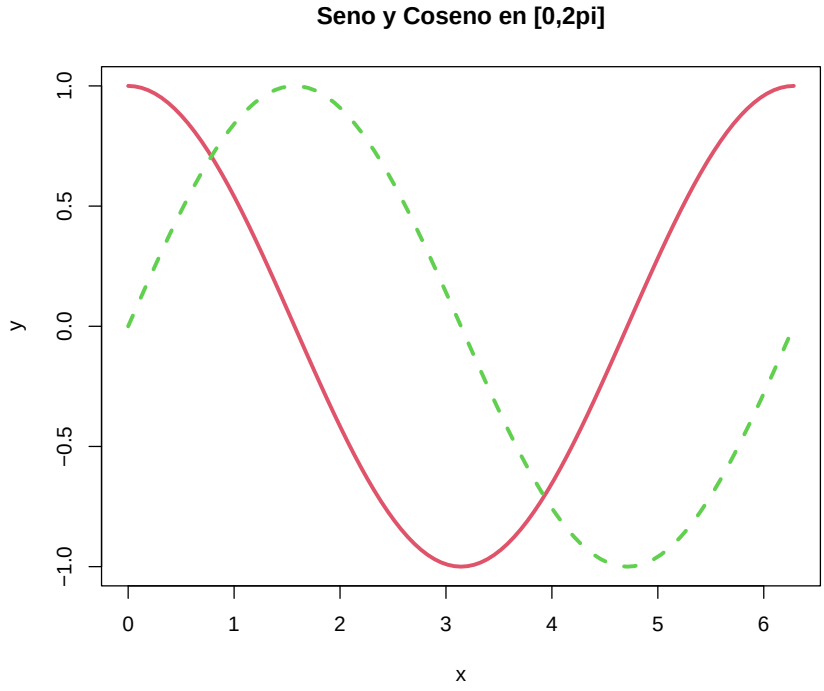
```
y2 <- sin(x)
plot( x, y, type = "l", col = 2, lwd = 3, main = "Seno y Coseno en [0,2pi]")
lines(x, y2, col = 3, lwd = 3, lty = 2)
points(pi, 0, pch = 17, col = 4)
legend(0, -0.5, c("Coseno", "Seno"), col = 2:3, lty = 1:2, lwd = 3)

abline(v = pi, lty = 3)
abline(h = 0, lty = 3)
text(pi, 0, "(pi,0)", adj = c(0, 0))
```



Alternativamente se podría usar `curve()`:

```
curve(cos, 0, 2*pi, col = 2, lwd = 3,
      ylab = "y", main = "Seno y Coseno en [0,2pi]")
curve(sin, col = 3, lwd = 3, lty = 2, add = TRUE)
```



Además de los gráficos estándar, en `R` están disponibles muchas librerías gráficas adicionales, entre ellas destaca ggplot2.

1.4. Programación

Para más detalles ver el Capítulo 11 del libro de referencia.

1.4.1. Funciones

Ver Sección 11.1.

```
nombre <- function(parámetros) {
 comandos
 return(salida)
}
```

Las funciones en R devuelven un único objeto. Si nos interesa retornar varios valores, los agrupamos dentro de un único objeto (típicamente una lista). Mediante `return()` podemos especificar la salida, si no se incluye se devuelve el resultado de la última instrucción evaluada.

En la lista de parámetros podemos poner valores por defecto, de la forma `nombre = valor` (`nombre <- valor` es erróneo, no confundir al llamar a una función).

El *entorno* de una función incluye los objetos definidos en la consola; y las variables definidas en una función serán también reconocidas en cualquier otra nueva función definida dentro de la anterior. Pero no al revés.

Ejemplo:

```
media <- function(x) {
  n <- length(x)
  m <- sum(x)/n
  return(m)
}
# También sería válido:
# media <- function(x) sum(x)/length(x)
# pero no se recomienda
x <- 1:11
media(x)
```

```
## [1] 6
```

```
mean(x)
```

```
## [1] 6
```

1.4.2. Ejercicio

Modificar el código del Ejercicio 1.1.5 para crear la función `cuota` que implemente el cálculo de la cuota mensual R de devolución de un préstamo.

Input: capital P, interés i y meses n.

Output: la cuota mensual de devolución.

1.4.3. Ejecución condicional

Ver Sección 11.2.

```
if (condición) expresion
```

```
if (condición) {
  expresiones
}

if (condición) {
  expresiones (si condición == TRUE)
} else {
  expresiones (si condición == FALSE)
}
```

También hay una versión vectorial: `ifelse(condiciones, si, no)` devuelve componentes de los objetos `si` o `no` dependiendo de si el correspondiente elemento de `condiciones` es `TRUE` o `FALSE`.

1.4.4. Bucles y vectorización

Ver Sección 11.3.

```
for (índice in vector) {
  comandos
}

while (condición) {
  comandos
}

repeat {
  comandos
}
```

Estos bucles se pueden combinar con las órdenes `break` (salir del bucle) y `next` (saltar a la siguiente iteración).

1.4.5. Ejemplo (media truncada)

Consideramos de nuevo los datos de lluvia almacenados en `precip` (disponible en el paquete base de `R`):

```
data(precip)    # Datos de lluvia
summary(precip) # Resumen estadístico
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    7.00   29.38   36.60   34.89   42.77   67.00
```

Como veremos más adelante, la media puede verse afectada por valores anormalmente grandes o pequeños (datos atípicos) y puede interesar emplear medidas robustas (como la mediana, `median()`). Otra alternativa es la media truncada, en la que se elimina un determinado porcentaje de los valores extremos antes de promediarlos. También se puede obtener esta medida empleando el parámetro `trim` de la función `mean()`.

```
mean(precip)
```

```
## [1] 34.88571
```

```
median(precip)
```

```
## [1] 36.6
```

```
mean(precip, trim = 0.2) # Media truncada al 20%
```

```
## [1] 36.29048
```

Como ejemplo final, implementaremos esta medida en una función empleando el siguiente algoritmo:

Input: un vector $x = \{x_1, \ldots, x_n\}$.

Output: la media truncada de x al `trim*100` %.

1. Calcular el número de elementos de x, llamarle n.
2. Calcular el entero más próximo al `trim*100` % de n; llamarle k.
3. Ordenar los elementos de x, seguir llamándole x.
4. Eliminar los k primeros y los k últimos elementos de x, llamarle y al vector formado por los elementos no eliminados.
5. Devolver la media de y.

```
# Media truncada
media_truncada <- function(x, trim = 0.2) {
  n <- length(x)
  k <- round(trim*n)
  if ( k < 0 | k >=  n/2 ) stop("Valor de 'trim' incorrecto")
  x <- sort(x)
  y <- x[ (k+1):(n-k) ]   # ':' tiene prioridad
  return(mean(y))
}
media_truncada(precip)
```

```
## [1] 36.29048
```

1.4.6. Ejercicio (propuesto)

Escribe una función que implemente el cálculo de la media recortada al `trim*100` %:

La media recortada es una variante de la media truncada. Se diferencia de esta en que en lugar de simplemente eliminar los primeros elementos del vector (ordenado), se reemplazan por el menor de los datos restantes; y de forma análoga los últimos elementos del vector se reemplazan por el mayor de los restantes.