

# R básico

10-2022

- 1 Factores
- 2 Lists
- 3 Matrices
- 4 Gráficos R base
- 5 Hojas de datos: data frames

## Sección 1

### Factores

---

# Factor

**Factor**: es como un vector, pero con una estructura interna más rica que permite usarlo para clasificar observaciones

- `levels`: atributo del factor. Cada elemento del factor es igual a un nivel. Los niveles clasifican las entradas del factor. Se ordenan por orden alfabético
- Para definir un factor, primero hemos de definir un vector y trasformarlo por medio de una de las funciones `factor()` o `as.factor()`.

## La función `factor()`

- `factor(vector, levels=...)`: define un factor a partir del vector y dispone de algunos parámetros que permiten modificar el factor que se crea:
  - `levels`: permite especificar los niveles e incluso añadir niveles que no aparecen en el vector
  - `labels`: permite cambiar los nombres de los niveles
- `levels(factor)`: para obtener los niveles del factor

# Factor ordenado

**Factor ordenado.** Es un factor donde los niveles siguen un orden

- `ordered(vector, levels=...)`: función que define un factor ordenado y tiene los mismos parámetros que `factor`

# Factores y factores ordenados

```
fac = factor(c(1,1,1,2,2,3,2,4,1,3,3,4,2,3,4,4),
             levels = c(1,2,3,4),
             labels = c("Sus", "Apr", "Not", "Exc"))
fac
```

```
[1] Sus Sus Sus Apr Apr Not Apr Exc Sus Not Not Exc Apr Not Exc Exc
Levels: Sus Apr Not Exc
```

```
facOrd = ordered(c(1,1,1,2,2,3,2,4,1,3,3,4,2,3,4,4),
                 levels = c(1,2,3,4),
                 labels = c("Sus", "Apr", "Not", "Exc"))
facOrd
```

```
[1] Sus Sus Sus Apr Apr Not Apr Exc Sus Not Not Exc Apr Not Exc Exc
Levels: Sus < Apr < Not < Exc
```

## Sección 2

### Lists

---



# List

**List.** Lista formada por diferentes objetos, no necesariamente del mismo tipo, cada cual con un nombre interno

- `list(...)`: función que crea una list
  - Para obtener una componente concreta usamos la instrucción `list$componente`
  - También podemos indicar el objeto por su posición usando dobles corchetes: `list[[i]]`. Lo que obtendremos es una list formada por esa única componente, no el objeto que forma la componente

## Obtener información de una list

- `str(list)`: para conocer la estructura interna de una list
- `names(list)`: para saber los nombres de la list

## Obtener información de una list

```
x = c(1,-2,3,4,-5,6,7,-8,-9,0)
miLista = list(nombre = "X", vector = x, media = mean(x),
               sumas = cumsum(x))

miLista
```

```
$nombre
[1] "X"
```

```
$vector
[1] 1 -2 3 4 -5 6 7 -8 -9 0
```

```
$media
[1] -0.3
```

```
$sumas
[1] 1 -1 2 6 1 7 14 6 -3 -3
```

## Obtener información de una list

```
str(miLista)
```

```
List of 4
 $ nombre: chr "X"
 $ vector: num [1:10] 1 -2 3 4 -5 6 7 -8 -9 0
 $ media : num -0.3
 $ sumas : num [1:10] 1 -1 2 6 1 7 14 6 -3 -3
```

```
names(miLista)
```

```
[1] "nombre" "vector" "media"  "sumas"
```

## Sección 3

### Matrices

---

## Cómo definir las

- `matrix(vector, nrow=n, byrow=valor_lógico)`: para definir una matriz de  $n$  filas formada por las entradas del vector
  - `nrow`: número de filas
  - `byrow`: si se iguala a `TRUE`, la matriz se construye por filas; si se iguala a `FALSE` (valor por defecto), se construye por columnas. `-ncol`: número de columnas (puede usarse en lugar de `nrow`)
  - R muestra las matrices indicando como  $[i,]$  la fila  $i$ -ésima y  $[,j]$  la columna  $j$ -ésima
  - Todas las entradas de una matriz han de ser del mismo tipo de datos

# Cómo definir las

## Ejercicio

- ¿Cómo definirías una matriz constante? Es decir, ¿cómo definirías una matriz  $A$  tal que  $\forall i = 1, \dots, n; j = 1, \dots, m, a_{i,j} = k$  siendo  $k \in \mathbb{R}$ ? Como R no admite incógnitas, prueba para el caso específico  $n = 3, m = 5, k = 0$

```
matrix(0, nrow = 3, ncol = 5)
```

- Con el vector  $\text{vec} = (1,2,3,4,5,6,7,8,9,10,11,12)$  crea la matriz

$$\begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix}$$

```
matrix(vec, ncol = 4)
```

## Cómo construirlas

- `rbind(vector1, vector2, ...)`: construye la matriz de filas `vector1`, `vector2`,...
- `cbind(vector1, vector2, ...)`: construye la matriz de columnas `vector1`, `vector2`,...
  - Los vectores han de tener la misma longitud
  - También sirve para añadir columnas (filas) a una matriz o concatenar por columnas (filas) matrices con el mismo número de filas (columnas)
- `diag(vector)`: para construir una matriz diagonal con un vector dado
  - Si aplicamos `diag` a un número  $n$ , produce una matriz identidad de orden  $n$



# Submatrices

- `matriz[i,j]`: indica la entrada  $(i,j)$  de la matriz, siendo  $i,j \in \mathbb{N}$ . Si  $i$  y  $j$  son vectores de índices, estaremos definiendo la submatriz con las filas pertenecientes al vector  $i$  y columnas pertenecientes al vector  $j$
- `matriz[i,]`: indica la fila  $i$ -ésima de la matriz, siendo  $i \in \mathbb{N}$
- `matriz[,j]`: indica la columna  $j$ -ésima de la matriz, siendo  $j \in \mathbb{N}$ 
  - Si  $i(j)$  es un vector de índices, estaremos definiendo la submatriz con las filas (columnas) pertenecientes al vector  $i(j)$

# Funciones

- `diag(matriz)`: para obtener la diagonal de la matriz
- `nrow(matriz)`: nos devuelve el número de filas de la matriz
- `ncol(matriz)`: nos devuelve el número de columnas de la matriz
- `dim(matriz)`: nos devuelve las dimensiones de la matriz
- `sum(matriz)`: obtenemos la suma de todas las entradas de la matriz
- `prod(matriz)`: obtenemos el producto de todas las entradas de la matriz
- `mean(matriz)`: obtenemos la media aritmética de todas las entradas de la matriz

# Funciones

- `colSums(matriz)`: obtenemos las sumas por columnas de la matriz
- `rowSums(matriz)`: obtenemos las sumas por filas de la matriz
- `colMeans(matriz)`: obtenemos las medias aritméticas por columnas de la matriz
- `rowMeans(matriz)`: obtenemos las medias aritméticas por filas de la matriz

# Funciones

## Ejemplo

Dada la matriz

$$A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

```
A = matrix(c(1,2,3,4,5,6,7,8,9), ncol = 3)  
dim(A)
```

```
[1] 3 3
```

```
diag(A)
```

```
[1] 1 5 9
```

## Función `apply()`

- `apply(matriz, MARGIN=..., FUN=función)`: para aplicar otras funciones a las filas o las columnas de una matriz
  - `MARGIN`: ha de ser 1 si queremos aplicar la función por filas; 2 si queremos aplicarla por columnas; o `c(1,2)` si la queremos aplicar a cada entrada

# Función `apply()`

```
apply(A, MARGIN = c(1,2), FUN = cuadrado)
```

	[,1]	[,2]	[,3]
[1,]	1	16	49
[2,]	4	25	64
[3,]	9	36	81

```
apply(A, MARGIN = 1, FUN = sum)
```

```
[1] 12 15 18
```

```
apply(A, MARGIN = 2, FUN = sum)
```

```
[1] 6 15 24
```

# Operaciones

- `t(matriz)`: para obtener la transpuesta de la matriz
- `+`: para sumar matrices
- `*`: para el producto de un escalar por una matriz
- `%*%`: para multiplicar matrices
- `mtx.exp(matriz,n)`: para elevar la matriz a  $n$ 
  - Del paquete Biodem
    - No calcula las potencias exactas, las aproxima
- `%^%`: para elevar matrices
  - Del paquete expm
    - No calcula las potencias exactas, las aproxima

# Operaciones

## Ejercicio

Observad qué ocurre si, siendo  $A = \begin{pmatrix} 2 & 0 & 2 \\ 1 & 2 & 3 \\ 0 & 1 & 3 \end{pmatrix}$  y  $B = \begin{pmatrix} 3 & 2 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ , realizamos las operaciones

$A * B$ ,  $A^2$  y  $B^3$



# Operaciones

- `det(matriz)`: para calcular el determinante de la matriz
- `qr(matriz)$rank`: para calcular el rango de la matriz
- `solve(matriz)`: para calcular la inversa de una matriz invertible
  - También sirve para resolver sistemas de ecuaciones lineales. Para ello introducimos `solve(matriz,b)`, donde  $b$  es el vector de términos independientes

# Valores y vectores propios

## Vector propio y valor propio

- `eigen(matriz)`: para calcular los valores (vaps) y vectores propios (veps)
  - `eigen(matriz)$values`: nos da el vector con los vaps de la matriz en orden decreciente de su valor absoluto y repetidos tantas veces como su multiplicidad algebraica.
  - `eigen(matriz)$vectors`: nos da una matriz cuyas columnas son los veps de la matriz.

# Valores y vectores propios

```
M = rbind(c(2,6,-8), c(0,6,-3), c(0,2,1))  
eigen(M)
```

```
eigen() decomposition  
$values  
[1] 4 3 2
```

```
$vectors  
      [,1]      [,2] [,3]  
[1,] 0.2672612 -0.8164966  1  
[2,] 0.8017837  0.4082483  0  
[3,] 0.5345225  0.4082483  0
```

# Valores y vectores propios

## Ejercicio

Comprobad, con los datos del ejemplo anterior, que si  $P$  es la matriz de vectores propios de  $M$  en columna y  $D$  la matriz diagonal cuyas entradas son los valores propios de  $M$ , entonces se cumple la siguiente igualdad llamada **descomposición canónica**:

$$M = P \cdot D \cdot P^{-1}$$

## Valores y vectores propios

Si hay algún vap con multiplicidad algebraica mayor que 1 (es decir, que aparece más de una vez), la función `eigen()` da tantos valores de este vap como su multiplicidad algebraica indica. Además, en este caso, R intenta que los veps asociados a cada uno de estos vaps sean **linealmente independientes**. Por tanto, cuando como resultado obtenemos veps repetidos asociados a un vap de multiplicidad algebraica mayor que 1, es porque para este vap no existen tantos veps linealmente independientes como su multiplicidad algebraica y, por consiguiente, la matriz no es **diagonalizable**.

# Valores y vectores propios

```
M = matrix(c(0,1,0,-7,3,-1,16,-3,4), nrow=3, byrow=TRUE)
eigen(M)
```

```
eigen() decomposition
$values
[1] 3 2 2
```

```
$vectors
      [,1]      [,2]      [,3]
[1,] -0.1301889 -0.1825742 -0.1825742
[2,] -0.3905667 -0.3651484 -0.3651484
[3,]  0.9113224  0.9128709  0.9128709
```

## Sección 4

### Gráficos R base

---

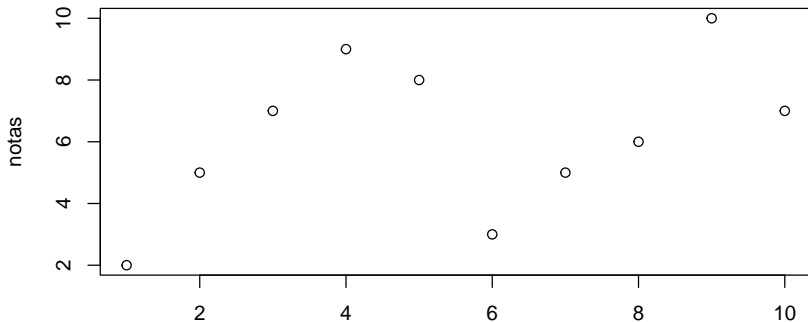
## Gráfico básico de puntos

- `plot(x,y)`: para dibujar un gráfico básico de puntos siendo `x,y` vectores numéricos
  - `plot(x) = plot(1:length(x),x)`
- `plot(x,función)`: para dibujar el gráfico de una función



## Gráfico básico de puntos





















```
alumnos = c(1:10)  
notas = c(2,5,7,9,8,3,5,6,10,7)  
plot(alumnos,notas)
```



## Parámetros de la función `plot()`

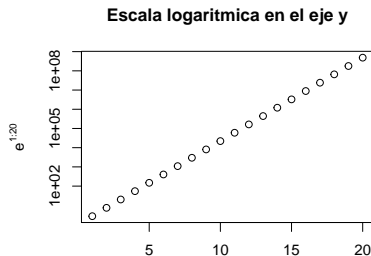
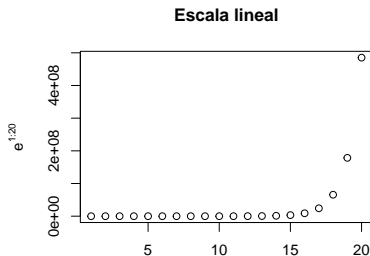
- `log`: para indicar que queremos el gráfico en escala logarítmica
- `main("título")`: para poner título al gráfico. Si en vez de un texto queráis poner una expresión matemática, tenéis que utilizar la función `expression()`
- `xlab("etiqueta")`: para poner etiqueta al eje  $X$
- `ylab("etiqueta")`: para poner etiqueta al eje  $Y$
- `pch=n`: para elegir el símbolo de los puntos.  $n = 0, 1, \dots, 25$ . El valor por defecto es `pch = 1`
- `cex`: para elegir el tamaño de los símbolos
- `col="color en inglés"`: para elegir el color de los símbolos. [Gama de colores](#).

## Parámetro pch - Tipos de símbolos

<b>0</b> 	<b>1</b> 	<b>2</b> 	<b>3</b> 	<b>4</b> 
<b>5</b> 	<b>6</b> 	<b>7</b> 	<b>8</b> 	<b>9</b> 
<b>10</b> 	<b>11</b> 	<b>12</b> 	<b>13</b> 	<b>14</b> 
<b>15</b> 	<b>16</b> 	<b>17</b> 	<b>18</b> 	<b>19</b> 

## Escala logarítmica

```
par(mfrow = c(1,2))  
plot = plot(exp(1:20), xlab = "Indice",  
            ylab = expression(e^{1:20}),  
            main = "Escala lineal")  
plotLog = plot(exp(1:20), log = "y", xlab = "Indice",  
               ylab = expression(e^{1:20}),  
               main = "Escala logaritmica en el eje y")
```



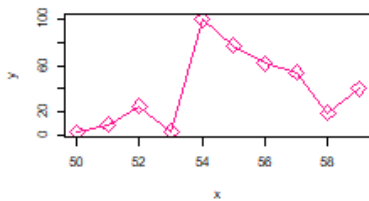
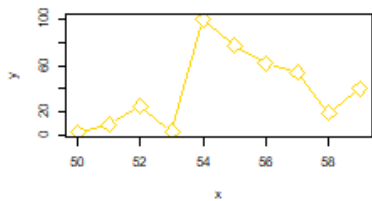
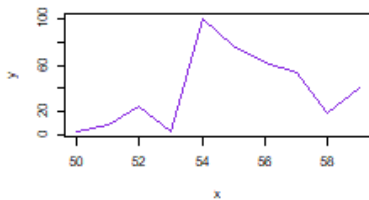
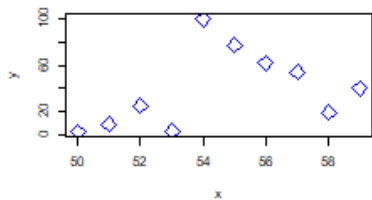
# Parámetros de la función `plot()`

- `type`: para elegir el tipo de gráfico que queremos:
  - `p`: puntos (valor por defecto)
  - `l`: líneas rectas que unen los puntos (dichos puntos no tienen símbolo)
  - `b`: líneas rectas que unen los puntos (dichos puntos tienen símbolo). Las líneas no traspasan los puntos
  - `o`: como el anterior pero en este caso las líneas sí que traspasan los puntos
  - `h`: histograma de líneas
  - `s`: histograma de escalones
  - `n`: para no dibujar los puntos

## Tipos de gráfico

```
par(mfrow = c(3,2))
x = c(50:59)
y = c(2,9,25,3,100,77,62,54,19,40)
plot(x,y, pch = 23, cex = 2, col = "blue", type = "p")
plot(x,y, pch = 23, cex = 2, col = "blueviolet", type = "l")
plot(x,y, pch = 23, cex = 2, col = "gold", type = "b")
plot(x,y, pch = 23, cex = 2, col = "deeppink", type = "o")
plot(x,y, pch = 23, cex = 2, col = "springgreen",
      type = "h")
plot(x,y, pch = 23, cex = 2, col = "firebrick1",
      type = "s")
par(mfrow = c(1,1))
```

# Tipos de gráfico



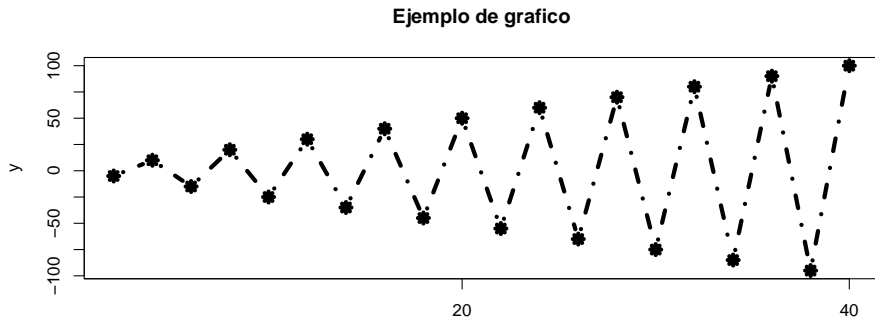
## Parámetros de la función `plot()`

- `lty`: para especificar el tipo de línea
  - “solid” : 1: línea continua (valor por defecto)
  - “dashed” : 2: línea discontinua
  - “dotted” : 3: línea de puntos
  - “dotdashed” : 4: línea que alterna puntos y rayas
- `lwd`: para especificar el grosor de las líneas
- `xlim`: para modificar el rango del eje  $X$
- `ylim`: para modificar el rango del eje  $Y$
- `xaxp`: para modificar posiciones de las marcas en el eje  $X$
- `yaxp`: para modificar posiciones de las marcas en el eje  $Y$



## Parámetros de la función plot()

```
x = (2*(1:20))  
y = (-1)^(1:20)*5*(1:20)  
plot(x,y, main = "Ejemplo de grafico", pch = 8, cex = 1,  
      type = "b", lty = 4, lwd = 4,  
      xaxp = c(0,40,2), yaxp = c(-100,100,8))
```



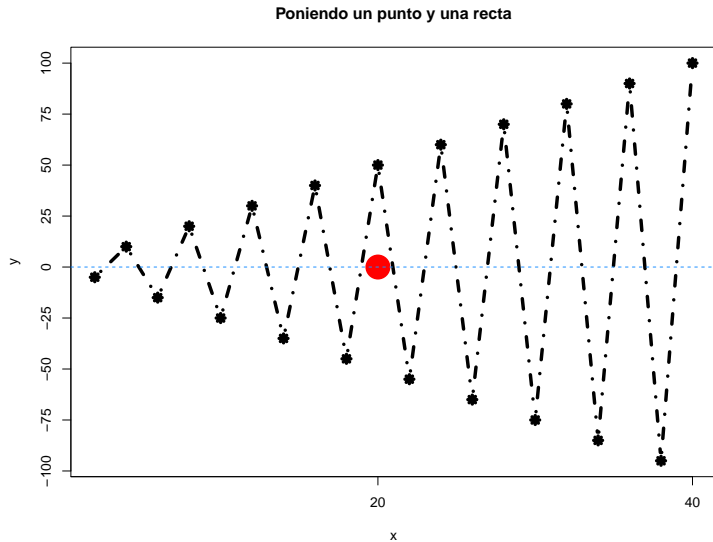
## Añadir elementos al gráfico

- `points(x,y)`: añade un punto de coordenadas  $(x,y)$  a un gráfico ya existente
- `abline`: para añadir una recta a un gráfico ya existente
  - `abline(a,b)`: añade la recta  $y = ax + b$
  - `abline(v = x0)`: añade la recta vertical  $x = x_0$ .  $v$  puede estar asignado a un vector
  - `abline(h = y0)`: añade la recta horizontal  $y = y_0$ .  $h$  puede estar asignado a un vector

## Añadiendo punto y recta

```
x = (2*(1:20))
y = (-1)^(1:20)*5*(1:20)
plot(x,y, main = "Poniendo un punto y una recta", pch = 8,
      cex = 1, type = "b", lty = 4,
      lwd = 4, xaxp = c(0,40,2), yaxp = c(-100,100,8))
points(20,0, col = "red", cex = 4, pch = 16)
abline (h = 0, lty = 2, col = "dodgerblue")
```

# Añadiendo punto y recta



## Añadir elementos al gráfico

- `text(x,y,labels = "....")`: añade en el punto de coordenadas  $(x,y)$  el texto especificado como argumento de `labels`
  - `pos`: permite indicar la posición del texto alrededor de las coordenadas  $(x,y)$ . Admite los siguientes valores:
    - 1: abajo
    - 2: izquierda
    - 3: arriba
    - 4: derecha
    - 5: sin especificar: el texto se sitúa centrado en el punto  $(x,y)$

## Añadiendo etiquetas

```
alumnos = c(1:10)
notas = c(2,5,7,9,8,3,5,6,10,7)
plot(alumnos,notas, main = "Grafico con texto")
text(alumnos,notas,
     labels = c("S","A","N","E","N","S","A","A","E","N"),
     pos = c(rep(3,times = 8),1,3))
```

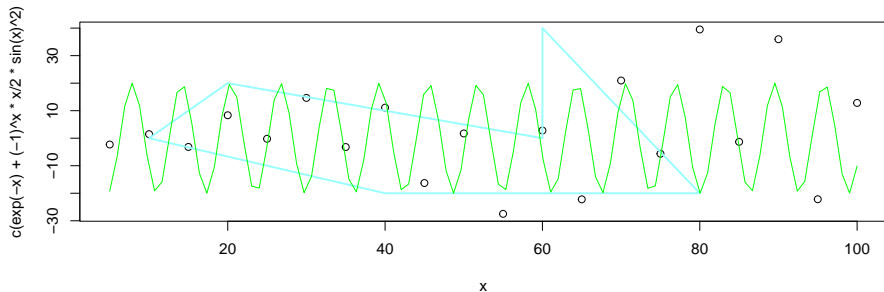


## Añadir elementos al gráfico

- `lines(x, y)`: añade a un gráfico existente una línea poligonal que une los puntos  $(x_i, y_i)$  sucesivos.  $x, y$  son vectores numéricos
- `curve(curva)`: permite añadir la gráfica de una curva a un gráfico existente
  - `add=TRUE`: si no, la curva no se añade
  - La curva se puede especificar mediante una expresión algebraica con variable  $x$ , o mediante su nombre si la hemos definido antes

## Añadiendo líneas y curvas

```
x = c(5*(1:20))  
plot(x, c(exp(-x) + (-1)^x * x/2 * sin(x)^2))  
lines(c(20, 10, 40, 80, 60, 60, 20), c(20, 0, -20, -20, 40, 0, 20),  
      lwd = 2, col = "darkslategray1")  
curve(20 * sin(x), add = TRUE, col = "green")
```





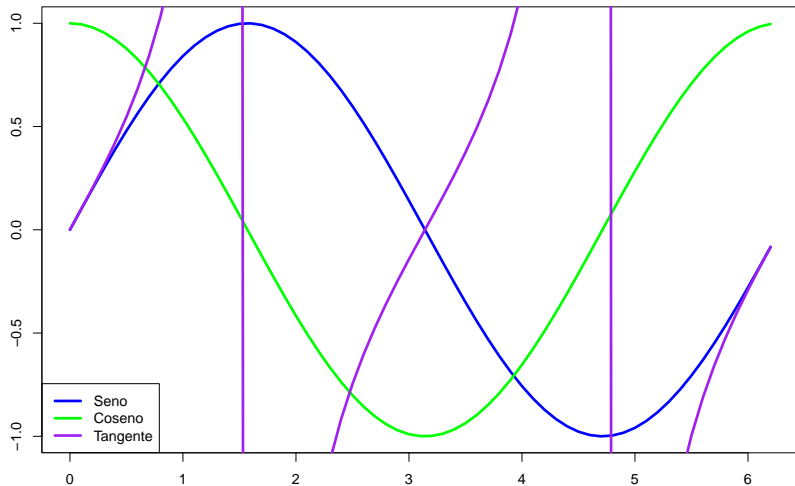
## Añadir elementos al gráfico

- `legend(posición, legend = ...)`: para añadir una leyenda
  - La posición indica donde queremos situar la leyenda. Puede ser o bien las coordenadas de la esquina superior izquierda de nuestra leyenda, o bien una de las palabras siguientes:
    - “bottom” / “bottomright” / “bottomleft”
    - “top” / “topright” / “topleft”
    - “center” / “right” / “left”
  - `legend`: contiene el vector de nombres entre comillas con los que queremos identificar a las curvas en la leyenda

## Añadiendo leyenda

```
x = seq(0,2*pi,0.1)
plot(x,sin(x),type="l",col="blue",lwd=3, xlab="", ylab="")
lines(x,cos(x),col="green",lwd=3)
lines(x, tan(x), col="purple",lwd=3)
legend("bottomleft",col=c("blue","green","purple"),
      legend=c("Seno","Coseno", "Tangente"),
      lwd=3, bty="l")
```

## Añadiendo leyenda



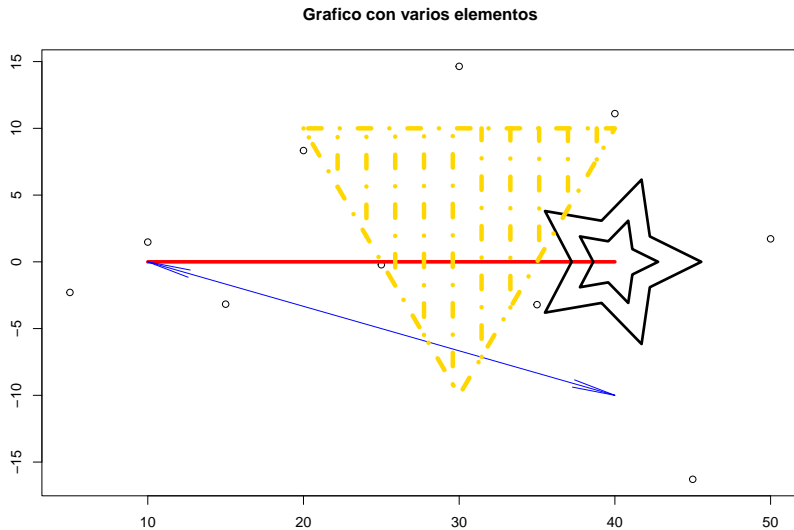
## Añadir elementos al gráfico

- `segments`: para añadir segmentos a un gráfico existente
- `arrows`: para añadir flechas a un gráfico existente
- `symbols`: para añadir símbolos a un gráfico existente
- `polygon`: para añadir polígonos cerrados especificando sus vértices a un gráfico existente

## Añadiendo elementos

```
x = c(5*(1:10))
plot(x,c(exp(-x)+(-1)^x*x/2*sin(x)^2), xlab = "", ylab = "",
      main = "Grafico con varios elementos")
segments(10,0,40,0, col = "red", lwd = 4)
arrows(10,0,40,-10, col = "blue", length = 0.5,
       angle = 5, code = 3)
symbols(40,0,stars = cbind(1,.5,1,.5,1,.5,1,.5,1,.5),
        add = TRUE, lwd = 3, inches = 0.5)
symbols(40,0,stars = cbind(1,.5,1,.5,1,.5,1,.5,1,.5),
        add = TRUE, lwd = 3)
polygon(c(20,30,40),c(10,-10,10), col = "gold",
        density = 3, angle = 90, lty = 4,
        lwd = 5)
```

# Añadiendo elementos



## Sección 5

### Hojas de datos: data frames

---

# Data frames

**Data frame.** Un data frame es una tabla de doble entrada, formada por variables en las columnas y observaciones de estas variables en las filas, de manera que cada fila contiene los valores de las variables para un mismo caso o un mismo individuo.

- `data()`: para abrir una ventana con la lista de los objetos de datos a los que tenemos acceso en la sesión actual de R (los que lleva la instalación básica de R y los que aportan los paquetes que tengamos cargados).
- Si entramos `data(package=.packages(all.available = TRUE))` obtendremos la lista de todos los objetos de datos a los que tenemos acceso, incluyendo los de los paquetes que tengamos instalados, pero que no estén cargados en la sesión actual.



## Acceder a la información, estructura y atributos de un data frame

- `head(d.f,n)`: para mostrar las  $n$  primeras filas del data frame. Por defecto se muestran las 6 primeras filas
- `tail(d.f,n)`: para mostrar las  $n$  últimas filas del data frame. Por defecto se muestran las 6 últimas
- `str(d.f)`: para conocer la estructura global de un data frame
- `names(d.f)`: para producir un vector con los nombres de las columnas

# Acceder a la información, estructura y atributos de un data frame

```
str(Orange)
```

```
Classes 'nfnGroupedData', 'nfGroupedData', 'groupedData' and 'data.frame': 35 obs. of 3 variables:
 $ Tree          : Ord.factor w/ 5 levels "3"<"1"<"5"<"2"<...: 2 2 2 2 2 2 2 2 4 4 4 ...
 $ age           : num 118 484 664 1004 1231 ...
 $ circumference: num 30 58 87 115 120 142 145 33 69 111 ...
 - attr(*, "formula")=Class 'formula' language circumference ~ age | Tree
 .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
 - attr(*, "labels")=List of 2
 ..$ x: chr "Time since December 31, 1968"
 ..$ y: chr "Trunk circumference"
 - attr(*, "units")=List of 2
 ..$ x: chr "(days)"
 ..$ y: chr "(mm)"
```

# Acceder a la información, estructura y atributos de un data frame

```
head(Orange,4)
```

	Tree	age	circumference
1	1	118	30
2	1	484	58
3	1	664	87
4	1	1004	115

```
tail(Orange,4)
```

	Tree	age	circumference
32	5	1004	125
33	5	1231	142
34	5	1372	174
35	5	1582	177

## Acceder a la información, estructura y atributos de un data frame

- `rownames(d.f)`: para producir un vector con los identificadores de las filas
  - R entiende siempre que estos identificadores son palabras, aunque sean números, de ahí que los imprima entre comillas
- `colnames(d.f)`: para producir un vector con los identificadores de las columnas
- `dimnames(d.f)`: para producir una list formada por dos vectores (el de los identificadores de las filas y el de los nombres de las columnas)
- `nrow(d.f)`: para consultar el número de filas de un data frame
- `ncol(d.f)`: para consultar el número de columnas de un data frame
- `dim(d.f)`: para producir un vector con el número de filas y el de columnas

## Acceder a la información, estructura y atributos de un data frame

- `d.f$nombre_variable`: para obtener una columna concreta de un dataframe
  - El resultado será un vector o un factor, según cómo esté definida la columna dentro del data frame
  - Las variables de un data frame son internas, no están definidas en el entorno global de trabajo de R

## Sub-data frames

- `d.f[n,m]`: para extraer “trozos” del data frame por filas y columnas (funciona exactamente igual que en matrices) donde  $n$  y  $m$  pueden definirse como:
  - intervalos
  - condiciones
  - números naturales
  - no poner nada
  - Si sólo queremos definir la subtabla quedándonos con algunas variables, basta aplicar el nombre del data frame al vector de variables
  - Estas construcciones se pueden usar también para reordenar las filas o columnas

## Sub-data frames

```
dataOrange = Orange  
dataOrange[c(10:12),]
```

	Tree	age	circumference
10	2	664	111
11	2	1004	156
12	2	1231	172

```
dataOrange[c(2,17),c(1,3)]
```

	Tree	circumference
2	1	58
17	3	75

# Sub-data frames

```
dataOrange[2,3]
```

```
[1] 58
```

```
dataOrange[dataOrange$circumference<=50,]
```

	Tree	age	circumference
1	1	118	30
8	2	118	33
15	3	118	30
22	4	118	32
29	5	118	30
30	5	484	49



## Leyendo tablas de datos

- `read.table()`: para definir un data frame a partir de una tabla de datos contenida en un fichero
  - Este fichero puede estar guardado en nuestro ordenador o bien podemos conocer su url. Sea cual sea el caso, se aplica la función al nombre del fichero o a la dirección entre comillas

Aquí tenéis una [lista de data frames](#) para practicar

## Parámetros de la función `read.table()`

- `header = TRUE`: para indicar si la tabla que importamos tiene una primera fila con los nombres de las columnas. El valor por defecto es `FALSE`
- `col.names = c(...)`: para especificar el nombre de las columnas. No olvidéis que cada nombre debe ir entre comillas
- `sep`: para especificar las separaciones entre columnas en el fichero (si no es un espacio en blanco). Si es así, hay que introducir el parámetro pertinente entre comillas
- `dec`: para especificar el signo que separa la parte entera de la decimal (si no es un punto. Si es así, hay que introducir el parámetro pertinente entre comillas

## Parámetros de read.table()

```
notas = read.table(  
  "http://aprender.uib.es/Rdir/Controls11-12.txt",  
  col.names = c("Nota_Parcial", "Nota_Final", "Grup"),  
  sep="," , header=TRUE)  
head(notas, 8)
```

	Nota_Parcial	Nota_Final	Grup
1	35	34	1
2	45	30	0
3	64	19	1
4	67	30	0
5	82	31	0
6	50	34	1
7	68	30	0
8	46	23	2

## Más parámetros de `read.table()`

- `stringsAsFactors`: para prohibir la transformación de las columnas de palabras en factores debemos usar `stringsAsFactors=FALSE` (ya que por defecto, R realiza dicha transformación)
- Para importar un fichero de una página web segura (cuyo url empiece con `https`), no podemos entrar directamente la dirección en `read.table()`; una solución es instalar y cargar el paquete `Rcurl` y entonces usar la instrucción `read.table(textConnection(getURL("url ")), ...)`.

## Otros formatos de fichero de datos

- `read.csv()`: para importar ficheros en formato CSV
- `read.xls()` o `read.xlsx()`: para importar hojas de cálculo tipo Excel u OpenOffice en formato XLS o XLSX, respectivamente. Se necesita el paquete `xlsx`
- `read.mtb()`: para importar tablas de datos Minitab. Se necesita el paquete `foreign`
- `read.spss()`: para importar tablas de datos SPSS. Se necesita el paquete `foreign`

## Exportación de datos a ficheros

- `write.table(df, file = "")`: para exportar un data frame a un fichero
  - `file = ""`: es donde indicaremos el nombre que queremos darle al fichero
  - Podemos usar el parámetro `sep` para indicar el símbolo de separación de columnas. Siempre entre comillas
  - También podemos utilizar el parámetro `dec` para indicar la separación entre la parte entera y decimal de los datos

## Exportando datos a ficheros

```
write.table(notas, file = "../data/NotasData.csv",  
            dec = ".")  
notas2 = read.table("../data/NotasData.csv", header = TRUE)  
str(notas2)
```

```
'data.frame':  156 obs. of  3 variables:  
 $ Nota_Parcial: int  35 45 64 67 82 50 68 46 43 77 ...  
 $ Nota_Final  : int  34 30 19 30 31 34 30 23 51 53 ...  
 $ Grup        : int  1 0 1 0 0 1 0 2 2 1 ...
```

## Crear data frames

- `data.frame(vector_1, ..., vector_n)`: para construir un data frame a partir de vectores introducidos en el orden en el que queremos disponer las columnas de la tabla
  - R considera del mismo tipo de datos todas las entradas de una columna de un data frame
  - Las variables tomarán los nombres de los vectores. Estos nombres se pueden especificar en el argumento de `data.frame` entrando una construcción de la forma `nombre_variable = vector`
  - `rownames`: para especificar los identificadores de las filas
  - También en esta función podemos hacer uso del parámetro `stringsAsFactors` para evitar la transformación de las columnas de tipo palabra en factores



## Crear data frames

```
Programacion = c(1,2,0,5,4,6,7,5,5,8)
Calculo = c(3,3,2,7,9,5,6,8,5,6)
Empresa = c(4,5,4,8,8,9,6,7,9,10)
grados = data.frame(Pr = Programacion,
                     Ca = Calculo, Em = Empresa)

str(grados)
```

```
'data.frame':  10 obs. of  3 variables:
 $ Pr: num  1 2 0 5 4 6 7 5 5 8
 $ Ca: num  3 3 2 7 9 5 6 8 5 6
 $ Em: num  4 5 4 8 8 9 6 7 9 10
```

## Crear data frames

- `fix(d.f)`: para crear / editar un data frame con el editor de datos
- `names(d.f)`: para cambiar los nombres de las variables
- `rownames(d.f)`: para modificar los identificadores de las filas. Han de ser todos diferentes
- `dimnames(d.f)=list(vec_nom_fil, vec_nom_col)`: para modificar el nombre de las filas y de las columnas simultáneamente

## Crear data frames

- `d.f[núm_fila,] = c(...)`: para añadir una fila a un data frame
  - Las filas que añadimos de esta manera son vectores, y por tanto sus entradas han de ser todas del mismo tipo
  - Si no añadimos las filas inmediatamente siguientes a la última fila del data frame, los valores entre su última fila y las que añadimos quedarán no definidos y aparecerán como NA
  - Para evitar el problema anterior, vale más usar la función `rbind()` para concatenar el data frame con la nueva fila

## Crear data frames

```
Ingles = c(5,4,6,2,1,0,7,8,9,6)
grados2 = cbind(grados, Ingles)
head(grados2)
```

	Pr	Ca	Em	Ingles
1	1	3	4	5
2	2	3	5	4
3	0	2	4	6
4	5	7	8	2
5	4	9	8	1
6	6	5	9	0

## Crear data frames

- `d.f$new_var`: para añadir una nueva variable al data frame
  - Podemos concatenar columnas con un data frame existente mediante la función `cbind()`. De este modo se puede añadir la columna directamente sin necesidad de convertirla antes a data frame
  - Esta nueva variable ha de tener la misma longitud que el resto de columnas del data frame original. Si no, se añadirán valores NA a las variables del data frame original o a la nueva variable hasta completar la misma longitud

## Cambiando los tipos de datos

- `as.character`: para transformar todos los datos de un objeto en palabras
- `as.integer`: para transformar todos los datos de un objeto a números enteros
- `as.numeric`: para transformar todos los datos de un objeto a números reales

## Más sobre sub-data frames

- `droplevels(d.f)`: para borrar los niveles sobrantes de todos los factores, ya que las columnas que son factores heredan en los sub-data frames todos los niveles del factor original, aunque no aparezcan en el trozo que hemos extraído
- `select(d.f, parámetros)`: para especificar que queremos extraer de un data frame
  - `starts_with("x")`: extrae del data frame las variables cuyo nombre empieza con la palabra "x"
  - `ends_with("x")`: extrae del data frame las variables cuyo nombre termina con la palabra "x"
  - `contains("x")`: extrae del data frame las variables cuyo nombre contiene la palabra "x"
  - Se necesita el paquete `dplyr` o mejor aún `tidyverse`

## Más sobre sub-data frames

- `subset(d.f,condición,select = columnas)`: para extraer del data frame las filas que cumplen la condición y las columnas especificadas
  - Si queremos todas las filas, no hay que especificar ninguna condición
  - Si queremos todas las columnas, no hace especificar el parámetro `select`
  - Las variables en la condición se especifican con su nombre, sin añadir antes el nombre del data frame



## Aplicando funciones a data frames

- `sapply(d.f, función)`: para aplicar una función a todas las columnas de un data frame en un solo paso
  - `na.rm=TRUE`: para evitar que el valor que devuelva la función para las columnas que contengan algún NA sea NA
- `aggregate(variables~factors,data=d.f,FUN=función)`: para aplicar una función a variables de un data frame clasificadas por los niveles de un, o más de un, factor
  - Si queremos aplicar la función a más de una variable, tenemos que agruparlas con un `cbind`
  - Si queremos separar las variables mediante más de un factor, tenemos que agruparlos con signos `+`

## Variables globales

No son funciones de **R** etiqueta

- `attach(d.f)`: para hacer que R entienda sus variables como globales y que las podamos usar por su nombre, sin necesidad de añadir delante el nombre del data frame y el símbolo \$
  - Si ya hubiera existido una variable definida con el mismo nombre que una variable del data frame al que aplicamos `attach`, hubiéramos obtenido un mensaje de error al ejecutar esta función y no se hubiera reescrito la variable global original
- `detach(d.f)`: para devolver la situación original, eliminando del entorno global las variables del data frame