APUNTES DE R

Instalación de R

- Página principal de R (si se pone en Google "R", es la página asociada con "The R Project for Statistical Computing"):

  www.r-project.org/

- En la columna izquierda seleccionamos

  `Download, Packages`
  CRAN

- A continuación buscamos, dentro de CRAN Mirrors, el pais España

  Spain

http://cran.es.r-project.org/	Spanish National Research Network, Madrid

  y accedemos a dicha página web.

- En la sección

  > `Download and Install R`
  >
  > Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:
  >
  > - Linux
  > - MacOS X
  > - Windows

  seleccionamos el sistema operativo de nuestro ordenador.

- A continuación, seleccionamos el subdirectorio base

base	Binaries for base distribution (managed by Duncan Murdoch)

y descargamos la versión disponible:

Download R 2.14.1 for Windows

Todo lo anterior (si se selecciona el sistema operativo Windows) se puede hacer accediendo directamente a la página

http://cran.es.r-project.org/bin/windows/base

Editor Tinn-R

Se puede descargar por ejemplo de la página http://www.sciviews.org/Tinn-R/

Se pueden escribir instrucciones de R con este editor, guardar el fichero y para ejecutar trozos de ese fichero con R se marcan y se selecciona la opción “Send to R” del menu “R”.

Comenzando con R

Al ejecutar el programa en la interfaz de RGui aparece el símbolo > esperando la entrada de instrucciones.

El menú principal contiene las pestañas típicas de otras aplicaciones: *Archivo, Editar, Visualizar, Ventanas* y *Ayuda*, junto con una específicas de R: *Misc* y *Paquetes*.

Ayuda en R

Para solicitar ayuda sobre un tema general podemos escribir, por ejemplo

```
>help(Uniform)
```

y obtenemos una ventana de ayuda sobre la distribución uniforme. También podemos acceder a la misma información a través del menú *Ayuda/Funciones R(texto).*

Si no se conoce la grafía de una expresión podemos escribir
```
> apropos("vector")
```
y aparecerán las expresiones que contienen el término introducido.

Con la instrucción
```
>help("vector")
```
aparece una ventana de ayuda con información sobre el comando `vector`.

Operaciones aritméticas

Directamente se pueden realizar operaciones aritméticas después del símbolo >:
+(suma), **-** (diferencia), *****(producto), **/**(cociente), **^**(potencia):

```
> 20+34
[1] 54
> 30-5
[1] 25
> 3-6
[1] -3
> 3*4
[1] 12
> 50/5
[1] 10
> 3^2
[1] 9
```

Nota: Se pueden recuperar líneas escritas con anterioridad usando los cursores arriba y abajo

Asignación de valores

R es un lenguaje orientado a objetos. Comenzaremos viendo cómo se realiza una asignación con los símbolos **<** y **-**.

x<- 3

o bien,

3->x

Código R
```
> x<-3
> x
[1] 3
```

También se puede usar el signo = pero no es lo habitual.

Código R
```
> x=4
> x
[1] 4
```

En la salida anterior [1] indica que el número que le sigue es el primer elemento de x (en este caso es evidente puesto que x es un escalar, luego lo veremos con un vector).

Si queremos crear un vector con componentes (3,6,8,9), se hará de la siguiente forma

x<-c(3,6,8,9)

Código R
```
> x<-c(3,6,8,9)
> x
[1] 3 6 8 9
```

Si el vector ocupara más de una línea, cada nueva línea empezaría con [n], indicando n el lugar que ocupa dentro del vector la coordenada que sigue.

Vamos a usar **n:m** para incluir en un vector los valores n, n+1, n+2, …., m, y con ello mostraremos lo anteriormente comentado

```
> x<-1:100
> x
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100
```

Se puede cambiar la amplitud de lo que aparece en pantalla con la siguiente función

```
> options(width=40)
> 1:100
  [1]   1   2   3   4   5   6   7   8
  [9]   9  10  11  12  13  14  15  16
 [17]  17  18  19  20  21  22  23  24
 [25]  25  26  27  28  29  30  31  32
 [33]  33  34  35  36  37  38  39  40
 [41]  41  42  43  44  45  46  47  48
 [49]  49  50  51  52  53  54  55  56
 [57]  57  58  59  60  61  62  63  64
 [65]  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80
 [81]  81  82  83  84  85  86  87  88
 [89]  89  90  91  92  93  94  95  96
 [97]  97  98  99 100
> options(width=60)
> 1:100
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13
 [14]  14  15  16  17  18  19  20  21  22  23  24  25  26
 [27]  27  28  29  30  31  32  33  34  35  36  37  38  39
 [40]  40  41  42  43  44  45  46  47  48  49  50  51  52
 [53]  53  54  55  56  57  58  59  60  61  62  63  64  65
 [66]  66  67  68  69  70  71  72  73  74  75  76  77  78
 [79]  79  80  81  82  83  84  85  86  87  88  89  90  91
 [92]  92  93  94  95  96  97  98  99 100
```

Para generar los valores de un vector también se pueden usar las funciones **seq()** y **rep()**
Con la instrucción **seq(a,b,by=r)** o, simplemente **seq(a,b,r)**, se genera una lista de números que empieza en **a** y termina en **b**, de la forma a, a+r, a+2r, …

```
> seq(1,20)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
> seq(1,20, by=2)
 [1]  1  3  5  7  9 11 13 15 17 19
> seq(20,1)
 [1] 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
> seq(20,1, by=-2)
 [1] 20 18 16 14 12 10  8  6  4  2
```

Con la instrucción **seq(a,b,length=r)** se generan **r** números entre **a** y **b**, igualmente espaciados.

```
> seq(4,10, length=8)
[1]  4.000000  4.857143  5.714286  6.571429  7.428571  8.285714  9.142857
[8] 10.000000
```

Con la instrucción **rep(x,r)** se genera una lista de **r** valores todos iguales a **x**. Se puede encadenar con la instrucción seq(), aplicarla a vectores (en tal caso, si el número de repeticiones de cada elemento del vector es distinto, se debe incluir un vector con los valores de tales repeticiones) y usar la opción **each** .

```
> rep(3,12)
 [1] 3 3 3 3 3 3 3 3 3 3 3 3
> rep(seq(2,20,by=2),2)
 [1]  2  4  6  8 10 12 14 16 18 20  2  4  6  8 10 12 14 16 18 20
```

```
> 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
```

Vectores

Ya se ha visto como asignar valores a un vector

```
> x<-c(3,6,8,9)
```

Par ver el contenido del vector basta escribir su nombre

```
> x
[1] 3 6 8 9
```

También se ha visto como usar el símbolo **:** se puede usar para crear secuencias crecientes (o decrecientes) de valores. Por ejemplo

```
> Numerosde5a20<-5:20
> Numerosde5a20
 [1]  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
> Numerosde20a5<-20:5
> Numerosde20a5
 [1] 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5
```

Se pueden concatenar vectores:

```
> x<-c(1,2)
> y<-c(3,4)
> z<-c(x,y)
> x
[1] 1 2
> y
[1] 3 4
> z
[1] 1 2 3 4
> v<-c(z,5:10)
> v
 [1]  1  2  3  4  5  6  7  8  9 10
```

Podemos cambiar de línea en mitad de una instrucción, se continúa en la línea siguiente con el símbolo + (que indica que la línea anterior está incompleta)

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

Para extraer elementos de un vector se usa **[ ]**

```
> x[3]
[1] 5
```

```
> x[c(1,2)]
[1] 1 3
> x[3:6]
[1] 5 6 5 4
```

Considerar un índice negativo significa ignorar el elemento o elementos correspondientes

```
> x<-11:20
> x
 [1] 11 12 13 14 15 16 17 18 19 20
> x[-3]
[1] 11 12 14 15 16 17 18 19 20
> x[c(-1,-2)]
[1] 13 14 15 16 17 18 19 20
> x[-(3:6)]
[1] 11 12 17 18 19 20
```

<u>Operaciones con vectores</u>

Multiplicación de un vector por un número

```
> x<-c(1,2,3,4,5)
> 2*x
[1]  2  4  6  8 10
```

Suma y resta de un vector y un número

```
> 2+x
[1] 3 4 5 6 7

> 2-x
[1]  1  0 -1 -2 -3
```

Suma de vectores

```
> y<-c(6,7,8,9,10)
> x+y
[1]  7  9 11 13 15
```

Notemos que las operaciones con vectores se hacen elemento a elemento

Potencia de un vector (eleva al cuadrado cada elemento del vector)

```
> x^2
[1]  1  4  9 16 25
```

Raiz cuadrada (hace la raíz cuadrada de cada elemento del vector)

```
> sqrt(x)
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

Producto de dos vectores (elemento a elemento)

¡Cuidado, con el producto!

```
> x<-c(1,2,3,4,5)
> y<-c(6,7,8,9,10)
> x*y
[1]  6 14 24 36 50
```

¡Y la potencia! (Eleva cada elemento del primer vector al correspondiente del segundo)

```
> x^y
[1]       1     128    6561  262144 9765625
```

Cuando los vectores tienen dimensiones diferentes, el de menor dimensión se extiende repitiendo los valores desde el principio, aunque da un mensaje de aviso

```
> x<-c(1,2,3,4,5)
> y<-c(1,2,3,4,5,6)
> x+y
[1]  2  4  6  8 10  7
Mensajes de aviso perdidos
In x + y :
  longitud de objeto mayor no es múltiplo de la longitud de uno menor
> x^y
[1]    1    4   27  256 3125    1
Mensajes de aviso perdidos
In x^y :
  longitud de objeto mayor no es múltiplo de la longitud de uno menor
```

Funciones con vectores

Asignar nombres a los elementos de un vector con la función **names()**

```
> x<-c(3.141592, 2.718281)
> names(x)<-c("pi","e")
> x
      pi        e
3.141592 2.718281
> x["pi"]
      pi
3.141592
```

La función **length()** devuelve la dimensión de un vector

```
> x<-c(1,2,3,4,5)
> length(x)
[1] 5
```

Las funciones **sum()** y **cumsum()** proporcionan la suma y sumas acumuladas de las componentes de un vector

```
> sum(x)
[1] 15
> cumsum(x)
```

```
[1]  1  3  6 10 15
```

Las funciones **max()** y **min()** proporcionan el valor máximo y mínimo de las componentes de un vector

```
> max(x)
[1] 5
> min(x)
[1] 1
```

Las funciones **mean(), median(), var()** y **sd()** proporcionan la media, mediana, cuasivarianza y cuasidesviación típica, respectivamente, de las componentes de un vector

```
> mean(x)
[1] 3
> median(x)
[1] 3
> var(x)
[1] 2.5
> sd(x)
[1] 1.581139
```

Las funciones **prod()** y **cumprod()** proporcionan el producto y productos acumulados de las componentes de un vector

```
> prod(x)
[1] 120
> cumprod(x)
[1]   1   2   6  24 120
```

La función **quantile()** proporciona los cuartiles

```
> quantile(x)
  0%  25%  50%  75% 100% 
   1    2    3    4    5 
```

La función **sort()** ordena en orden creciente de las componentes de un vector

```
> x<-c(2,6,3,7,9,1,4,7)
> sort(x)
[1] 1 2 3 4 6 7 7 9
```

La función **rev()** coloca las componentes de un vector en orden inverso a como han sido introducidas

```
> rev(x)
[1] 7 4 1 9 7 3 6 2
```

¿Cómo se ordenarían en orden decreciente las componentes de un vector?

Las funciones **cov()** y **cor()** proporcionan la covarianza y coeficiente de correlación entre dos vectores

```
> x<-c(1,2,3,4,5)
> y<-c(2,1,5,4,3)
> cov(x,y)
[1] 1.25
> cor(x,y)
[1] 0.5
```

Vectores de carácter

Tanto los escalares como los vectores pueden contener cadenas de caracteres. Todos los elementos de un vector deben ser del mismo tipo

```
> colores<-c("amarillo", "rojo", "verde")
> colores
[1] "amarillo" "rojo"     "verde"
> mas.colores<-c(colores, "azul", "negro")
> mas.colores
[1] "amarillo" "rojo"     "verde"    "azul"     "negro"
```

Si se intenta incluir un número después de caracteres lo interpreta como una cadena de caracteres.

```
> otros.colores<-c("naranja", "rosa", 1)
> otros.colores
[1] "naranja" "rosa"    "1"
```

Crear funciones

La forma de crear una función es con la siguiente instrucción:

Nombre_de_la_función<- function(x, y, …) {expresión de la función}

siendo x, y, … los argumentos de la función. Luego la función se ejecuta como **Nombre_de_la_función(x,y,…)**

Por ejemplo, la siguiente función calcula la media de las componentes de un vector

```
> media<-function(x){sum(x)/length(x)}
> y<-1:100
> media(y)
[1] 50.5
```

como hacía la función **mean()**

```
> mean(y)
[1] 50.5
```

Nota: Es útil ver los objetos que tenemos guardados en cada momento para no usar nombres que contengan valores que queramos conservar. Se hace con la función **objects()**

Gráficos en R

Diagrama de barras

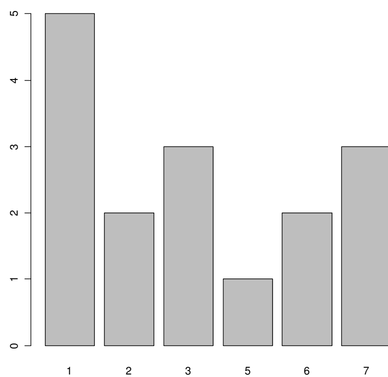
Sea

```
> x<-c(1,1,1,1,1,2,2,3,3,3,5,6,6,7,7,7)
```

la función **table()** tabula los datos en x y da lugar a

```
> table(x)
x
1 2 3 5 6 7
5 2 3 1 2 3
```

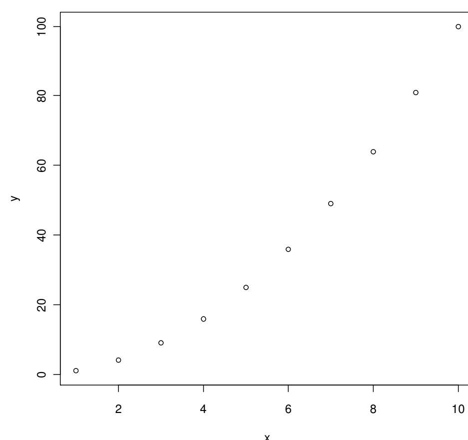
La instrucción **barplot(table(x))** muestra el diagrama de barras asociado



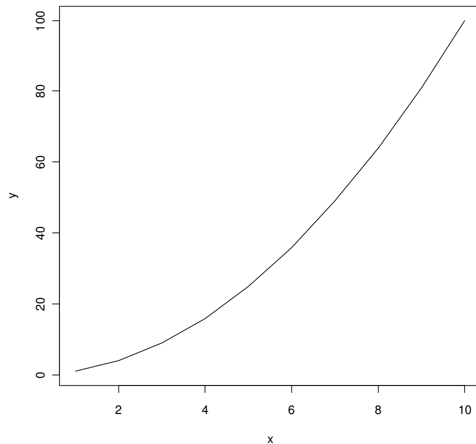
Plots

Se realizan con la instrucción **plot(x,y)**

```
> x<-1:10
> y=x^2
> plot(x,y)
```



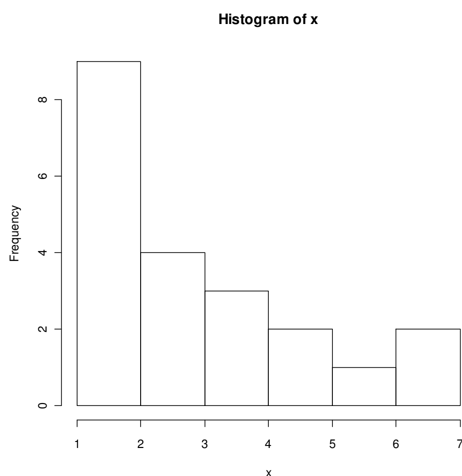
```
> plot(x,y,type=”l”)
```



Histograma (representación gráfica de los valores de una variable en forma de barras, siendo el área de cada barra es proporcional a la frecuencia de los valores representados).

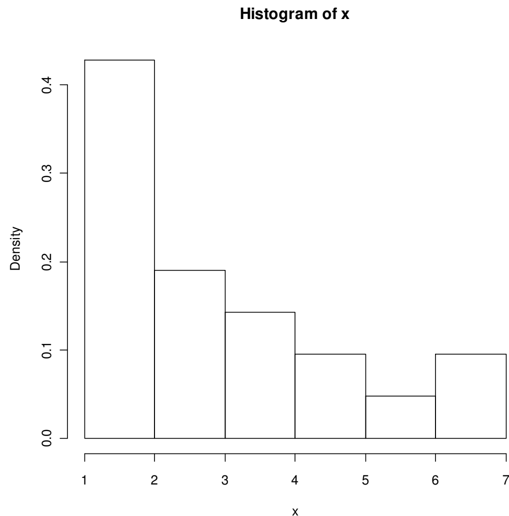
Se realizan con la instrucción **hist(x)**

```
> x<-c(1,2,3,1,2,4,2,6,7,2,4,5,3,7,1,3,5,2,4,3,2)
> hist(x)
```



Si queremos que el área de cada barra sea proporcional a la frecuencia relativa se debe incluir la opción freq=FALSE

```
> hist(x,freq=FALSE)
```

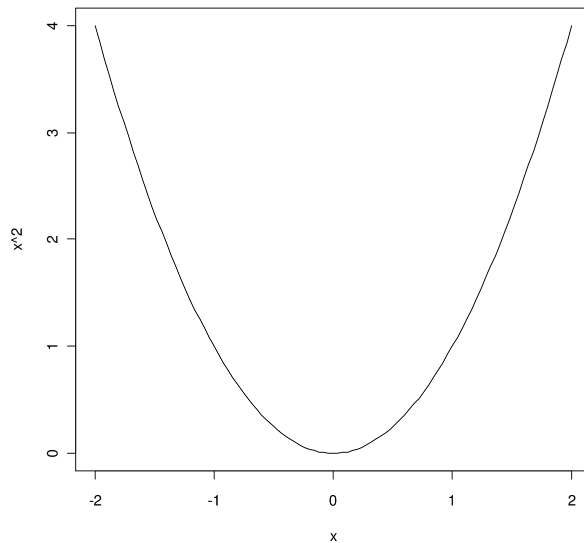


Curvas definidas por una expresión entre dos valores

Se usa el código **curve(expresión de la curva, valor inicial, valor final)**

Por ejemplo, para representar f(x)=x^2 entre -2 y 2 se usa

```
> curve(x^2,-2,2)
```



<u>Otras instrucciones de R</u>

<u>FOR</u>

La sentencia **for()** permite repetir una cierta operación un número de veces.

Su sintaxis es

for (*Nombre_del indice* **in** *vector***){comandos}**

de forma que al ejecutar dicha instrucción crea una variable llamada *Nombre_del_indice* igual a cada uno de los elementos del vector, en secuencia. Para cada valor se ejecuta todos los comandos incluidos entre llaves, considerándolos como un único comando. Si sólo hay que ejecutar un solo comando las llaves no son necesarias, pero es conveniente incluirlas para una mayor claridad.

Cuando se construye un vector a partir de comandos usando for() (o cualquier otra función de programación) es necesario definir previamente dicho vector. La instrucción **Nombre<-numeric(length=r)** o, simplemente, **Nombre<-numeric(r**) crea un objeto de tipo numérico de longitud r.

```
> x<-numeric(10)
> x
 [1] 0 0 0 0 0 0 0 0 0 0
```

Así, si queremos obtener los primeros 12 números de Fibonacci, se procede de la siguiente forma

```
> Fibonacci_12<-numeric(12)
> Fibonacci_12[1]<-Fibonacci_12[2]<-1
> for(i in 3:12){Fibonacci_12[i]<-Fibonacci_12[i-2]+Fibonacci_12[i-1]}
> Fibonacci_12
 [1]   1   1   2   3   5   8  13  21  34  55  89 144
```

SAMPLE

La función **sample()** permite tomar una muestra aleatoria simple a partir de un vector de valores con o sin reemplazamiento. Se usa como

sample(x, size, replace=FALSE, prob=NULL)

donde *x* es un vector de donde se quieren elegir los elementos o un entero positivo n (en este caso se interpreta como el vector generado por 1:n), *size* es un entero positivo que indica el número de elementos que se quieren elegir, *replace=FALSE* indica que el muestreo se hace sin reemplazamiento, mientras que *replace=TRUE* indica con reemplazamiento. Por último en *prob* se puede incluir un vector de probabilidades en el que cada componente será la probabilidad con la que se elegirá la correspondiente componente del vector que va a ser muestreado.

Por ejemplo
sample(c(3,5,7), size=2, replace=FALSE)
conduce a un vector de dos valores tomados (sin reemplazamiento) del conjunto {3,5,7}.

Simulación del lanzamiento de un dado

```
> sample(1:6,1)
[1] 6
> sample(1:6,1)
[1] 3
```

```
> sample(1:6,1)
[1] 4
> sample(1:6,1)
[1] 5
```

Simula del lanzamiento de cuatro dados o de un mismo dado cuatro veces

```
> sample(1:6,4, replace=T)
[1] 4 5 4 2
```

Supongamos una urna con 3 bolas blancas y 7 negras, simular la extracción de una bola (asignar, por ejemplo, el 1 a bola blanca y 0 a negra)

```
> sample(c(1,0), 1, prob=c(0.3,0.7))
[1] 1
> sample(c(1,0), 1, prob=c(0.3,0.7))
[1] 0
> sample(c(1,0), 1, prob=c(0.3,0.7))
[1] 1
> sample(c(1,0), 1, prob=c(0.3,0.7))
[1] 0
> sample(c(1,0), 1, prob=c(0.3,0.7))
[1] 0
> sample(c(1,0), 1, prob=c(0.3,0.7))
[1] 0
```

Si queremos simular 8 extracciones con reemplazamiento

```
> sample(c(1,0), 8, rep=T, prob=c(0.3,0.7))
[1] 1 0 0 0 0 0 1 0
```

Si sólo nos interesara el número de bolas blancas que salen, se puede hacer la suma.