# Prácticas sobre técnicas estadísticas aplicadas a la Climatología.

***Introducción y uso del paquete estadístico R dirigido hacia técnicas estadísticas aplicadas a la Climatología.***



## 1. Introducción al paquete estadístico R

*R* es un conjunto integrado de programas para manipulación de datos, cálculo y gráficos.

*R* nació principalmente con un *enfoque estadístico*, de hecho tanto en el paquete base, el núcleo de R, como en las bibliotecas externas tiene implementadas multitud de técnicas estadísticas tanto clásicas como modernas.

Sin embargo R no es solo un paquete estadístico, sino que tiene capacidades para manejo de datos, calculo sobre variables y arrays, en particular matrices, posibilidades gráficas bastante potentes para la representación de datos, así como un lenguaje de programación propio con posibilidad de crear funciones a medida. Este lenguaje propio bien desarrollado, simple y efectivo le da una gran potencia y flexibilidad.

Todo esto hace que el paquete R se pueda utilizar no solo como para fines estadísticos, para el que esta espléndidamente dotado, sino también para otros tipos de utilidades que podamos imaginar de una manera mucho más cómoda que otros programas.

Si a esto añadimos que R es gratis y de libre distribución y que los usuarios o desarrolladores pueden crear paquetes o herramientas nuevas que se implementan fácilmente en R hacen de el una opción muy buena para utilizarlo como entorno de programación, para cálculos y representaciones gráficas.

Otras propiedades muy interesantes del R es que esta disponible para *distintas plataformas*, existe en varias versiones *Windows* o *Linux*, y esta muy preparado para utilizar integrado en scripts Unix o Linux, permitiendo procesos de automatización y de planificación de tareas.

### 1.1 Instalación: paquete básico y librerías adicionales

Depende del sistema operativo pero todas las versiones se puede encontrar en: **https://www.r-project.org/** en *download R*, eligiendo posteriormente alguno de los *CRAN mirrors*, de los cuales hay varios en España, o en **https://cran.r-project.org/**.

R consta de un sistema o *paquete base* y de *paquetes adicionales* creados por desarrolladores o usuarios que extienden su funcionalidad.

Una vez descargado depende del sistema operativo:

- *Windows*: Desde el ejecutable que hayamos descargado se instala directamente
- *Linux*: Depende de la distribución. En las versiones modernas de estos sistemas operativos como *Ubuntu*, *OpenSuse*, etc. se puede descargar e instalar automáticamente desde los repositorios. También, desde la pantalla de comandos o consola, por ejemplo en *Ubuntu* (versiones modernas):

  ***sudo apt-get update***
  ***sudo apt-get install r-base***

## Inicio y primer contacto

Una vez instalado el *R*, se abre, desde Windows, haciendo un doble clic en el icono del Escritorio. Bajo Linux se abre primero una terminal de comandos o shell y se teclea ***R***. Para salir de la aplicación ***q()*** en ambos sistemas o cerrar desde la ***gui*** (*graphical user interface*). La aplicación preguntara si se quiere guardar el área de trabajo, por ahora decir que no.

## Entornos de trabajo para R

Además de la ***gui*** que sale por defecto al instalar el R, existen entornos que facilitan el manejo de scripts y la utilización de R en general.

Otras veces un editor de texto adecuado para programación como el ***NOTEPAD ++*** (Windows o Linux) o la mismo ***GUI*** nos pueden servir para tareas sencillas.

Un GUI distinto al que trae por defecto es el ***R commander*** (*Rcmdr*) que se instala y carga como una librería cualquiera de R. Es un sistema de ventanas donde, mediante menús, se pueden ejecutar las funciones más usadas de R.

Los más sofisticados, ***IDEs*** (*Integrated Development Environment*) facilitan todas las herramientas para ejecutar programas, editar el código y depurarlo, mostrar resultados, para hacer más cómoda la labor del desarrollador, aunque esto supone un esfuerzo inicial, a veces grande, para entender el IDE y sacar el máximo provecho. Un IDE muy usado y recomendado para R, y que sirve exactamente igual para Windows y Linux es el ***RStudio*** (https://www.rstudio.com/) que tiene una versión gratuita.

Nosotros en el curso trabajaremos con la ***gui*** que trae el R por defecto

## Paquetes o librerías

Con la instalación del *paquete básico* de R, tenemos muchas funcionalidades, no obstante existen multitud de módulos adicionales creados por desarrolladores que facilitan una gran cantidad de tareas en distintos temas. Los paquetes (*packages*) son colecciones de funciones y datos.

Los módulos se instalan en la carpeta ***library*** por defecto, pero que podemos cambiar si queremos. La carpeta donde están instaladas las librerías se muestra ejecutando desde la consola de R: ***.libPaths()***

Para ver los paquetes instalados: ***library()*** El hecho de que un paquete este instalado no significa que lo podamos usar, antes hay que cargarlo con ***library("nombre_paquete")***. Para ver los paquetes cargados en memoria: ***search()***

Los paquetes se pueden descargar o ver en: **https://cran.r-project.org/** en *Packages*. También se pueden descargar e instalar desde la misma aplicación R que es lo más práctico.

Para *instalar un paquete* se puede hacer desde la ***gui*** o desde la línea de comandos con:

***install.packages("nombre_paquete")***

Tiene las opciones de poner ***dependencies=TRUE*** para instalar también las dependencias, poner el repositorio con ***repos*** como por ejemplo ***repos= 'http://cran.rstudio.com'*** y otras opciones.

Si solo se escribe ***install.packages( )***, sin repositorio, nos saldrá una ventana con la lista de *mirrors* donde descargarlo y después otra con la lista de paquetes para elegir.

Ej.: ***install.packages('ColorPalette')***. Si estamos dentro de la intranet de *Aemet*, nos pedirá el proxy y la password.

(En el caso de que no pregunte por el proxy:
En el icono de acceso directo de Windows: destino:
***"C:\Program Files\R\R-3.1.1\bin\x64\Rgui.exe" http_proxy=http://proxy.aemet.es:3128 http_proxy_user=ask***)

Para *cargarlo en memoria* y que podamos ejecutarlo:

***library("nombre_paquete")***

Para obtener información o ayuda sobre un paquete ya instalado empleamos la función ***help ( package = "nombre_del_paquete" )***.

También se pueden instalar los paquetes descargando directamente el binario (***zip*** para Windows o ***gz*** para linux desde https://cran.r-project.org/web/packages/ al archivo de trabajo y luego desde la ***gui*** en paquetes e instalar paquetes a partir de archivos ***zip*** locales, se le da la localización donde lo hemos descargado. A veces es lo mejor para encontrar paquetes actualizados.

También se puede descargar en el directorio de trabajo, o darle la ruta completa de fichero, y después

***install.packages("jpeg_0.1-8.zip", repos=NULL, type='source')***

Para saber la versión de R instalada y la versión de los paquetes cargados:

***sessionInfo()***

## 1.2 Breve descripción de las características principales de R, tipos de datos y objetos

### Espacio y directorio de Trabajo

- El ***espacio de trabajo*** o *"**workspace**"* es un espacio virtual donde se almacenan los objetos definidos por el usuario (ya veremos qué son estos objetos, que incluyen variables, vectores, *dataframes*…), se almacenan en una memoria intermedia mientras trabajas con R. Cuando termina una sesión de R la propia aplicación te pregunta si quieres guardar el *"workspace"* para usos futuros; si se decide guardarlo se crea un fichero con todos los objetos actuales, en caso contrario se borran de la memoria.

- El ***Directorio de trabajo*** o *"**working directory**"* (***wd***) es el directorio o carpeta actual donde trabajamos con R. Aquí se guardaran los archivos de texto o scripts que hagamos. También es donde guardará el *workspace* al finalizar la sesión, si se decide guardarlo, y donde buscará un workspace guardado al inicio. Si se quiere que R lea un fichero que no esté en *"working directory"* hay que especificar la ruta completa.

*Nota: para que el R se abra siempre en el mismo directorio de trabajo en Windows se puede añadir en el icono del escritorio de R donde pone iniciar en, por ejemplo:* ***C:\Curso_Estadistica***

## Funciones de manejo del espacio y directorio de trabajo:

***El signo # es para comentarios en R, (ignora todo lo que viene después)***

```
> getwd()                  # muestra el emplazamiento del directorio de trabajo
> setwd( 'C:/Curso_Estadistica')     # cambia el directorio de trabajo a otra posición
```
***Nota:*** *dentro de R hay que poner la notación de Linux para los directorios:* ***/*** *en vez de* ******

```
> dir()                    # lista los ficheros dentro del directorio de trabajo actual
> history()                # muestra los comandos ejecutados, también se ven con las flechas de dirección.
> savehistory()            # Guarda el historial de comandos, por defecto en .RHistory
> loadhistory()            # Carga el historial de comandos
> save.image("Espacio.R")  # Guarda los objetos del workspace con un nombre que le pongamos (en este caso "Espacio.R" aunque podemos tener varios). Si al salir del GUI le decimos que queremos guardar la imagen del área de trabajo los guarda en .RData y los cargará automáticamente la próxima vez.
> load ("Espacio.R")  # Carga el workspace "Espacio.R". No borra los objetos que tengamos actualmente, sino que solo incorpora los que tengamos salvados en el workspace.
```

Para saber cuántos objetos tenemos en memoria:
```
> objects()
> ls()
```

Y se pueden borrar con ***rm(x)*** donde ***x*** es el nombre del objeto
***rm(list=ls())*** eliminaría todos los objetos de la memoria.

## Un primer vistazo de R:

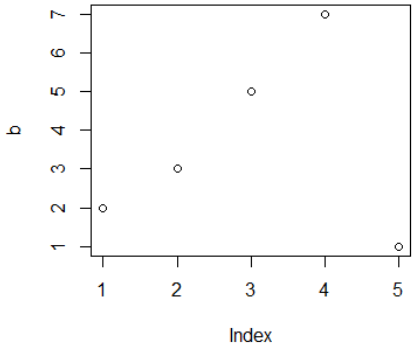
```
# En la línea de comandos de la gui ejecutamos:
> 2+2
> sqrt(10)     # raíz cuadrada de 10
> 2*3*4*5
> #   R conoce pi:
> pi
> a <- pi      # rellenamos una variable a con el numero pi
> a
> # Convertimos ángulos en grados a radianes y luego calculamos el seno.
Ejemplo 60 grados a radianes
> sin(60*pi/180)

> b <- c(2,3,5,7,1)    # asignamos unos números a un objeto b (un vector). (c combina elementos a un vector)
```

***> mean(b)  #media aritmetica de los numeros que contiene el vector b***

***> rnorm(5) # 5 numeros aleatorios de una distribución normal (mean= 0, sd = 1)***

***> x <- rnorm(5) # asignamos 5 números aleatorios de una distribución normal a un objeto (un vector) x***

***# Los números se calculan y se muestran (print) o tecleando el nombre del objeto***
***> x***
***> print(x)***
***# escribir x y print(x) produce el mismo resultado***
***> plot(b) # pinta en un gráfico sencillo el vector b de arriba.***




# Ayuda de R

Los comandos de *R* tienen una ayuda que se puede ver de varias formas:
***> ?rnorm*** o ***help(rnorm) # abre el navegador con ayuda sobre el comando rnorm***
***> help.start() # abre el navegador con la pantalla ayuda***
***> ?help.search # informa sobre el modo de buscar en R***
***> help.search("norm") #busca todos los comandos que contengan "norm"***
***> apropos("norm") # da la lista rápida de comandos que contengan "norm" en el nombre***

## *Ejercicios 1.1*

- Encontrar la ayuda para la función ***sort***
- Obtener la carpeta de trabajo o directorio actual (***wd***)
- Listar los ficheros del directorio o carpeta actual
- Crear un par de objetos (por ejemplo, ***a <- 2 ; b <- pi***) y luego listar todos los objetos del espacio de trabajo (***workspace***)
- Salvar el espacio de trabajo con los objetos que contiene, con el nombre ***'ultimo_wd'***
- Salir de R y decir que ***no*** cuando nos pregunte si queremos guardar la imagen del área de trabajo (workspace).
- Volver a entrar en R y listar los objetos. Cargar después el espacio de trabajo que hemos salvado antes y listar los objetos de nuevo. Si le decimos salvar el espacio de trabajo antes de salir de R, al volver a entrar nos cargara el workspace (lo guarda por defecto como ***.RData*** (no se ve al hacer ***dir()***)
- Crear desde fuera de R una carpeta de trabajo ***C:\Curso_Estadistica*** y descomprimir en ella el fichero ***Material.zip***
- Cambiar el directorio de trabajo a la nueva carpeta de trabajo ***C:\Curso_Estadistica***

# Operadores y funciones

Los *operadores aritméticos* elementales son los habituales:
**+**, **-**, *****, **/** para suma, resta, multiplicación y división y **^** o ****** para elevar a una potencia.

Hay muchas *funciones* disponibles aparte como ***exp()***, ***sin()***, ***cos()***, ***tan()***, ***sqrt()***, ***abs(), log10(), log(),*** etc. bien conocidas. *Las funciones en R siempre llevan un paréntesis*.

*Nota: las funciones trigonométricas son para ángulos en radianes.*

Para ver la documentación sobre operadores matemáticos: ***help('Arithmetic'), help('Math'), help('Trig'), help('Log')***.

Existen otras muchas que vienen documentadas en el manual como ***max()*** y ***min()*** para buscar extremos entre los elementos de un vector o varios, ***length(x)*** para la longitud de un vector x, ***sum(x)*** o ***prod(x),*** para la suma o producto de los elementos de un vector.

La función ***mean(x)*** calcula la media de un vector, mientras que ***var(x)*** calcula la cuasivarianza esto es:

***sum((x-mean(x))^2)/(length(x)-1)***

Para ordenar un vector se puede usar ***sort(x)*** que devuelve un vector del mismo tamaño de x pero ordenado en orden creciente; la función ***sort*** admite la opción: ***decreasing = TRUE*** (ordena de menor a mayor).
El comando ***rev(x)*** daría el vector invertido con el primer elemento colocado el último, el segundo lo coloca en antepenúltimo y así sucesivamente (no los ordena por valor numérico sino solo por orden de posición, mientras que ***sort*** si los ordena por valor numérico).

Los operadores logicos son:
**<**, **>**, **<=**, **>=**, **==**, ***!=*** (distinto)

***!*** (NO o contrario)
***&*** (Y)
**|** (O)

# Tipos de datos

***El tipo de los datos*** (***modos)*** en R que contienen *los objetos R* pueden ser de varias clases. Los más sencillos son ***numeric*** (numérico), ***character*** (carácter) y ***logical*** (***T*** o ***F***, true o false). Para ver el tipo de datos de un objeto como un vector *A*, etc se utiliza ***mode(A)***. Cuando el dato no esta disponible se representa con ***NA*** (not available). (Ver casos particulares de datos en ***Notas.Sueltas.sobre.R***)

# Objetos en R

R utiliza varios tipos de objetos como variables, vectores, matrices, arrays, listas, hojas de datos, funciones y algunos más.

## Vectores

Los vectores son conjuntos de elementos del mismo tipo o modo. Es el tipo de objeto más utilizado en R, especialmente como argumento de funciones. Los vectores no tienen que ser numéricos necesariamente, pero ciertas funciones como ***sqrt()***, etc., sobre un vector no funcionaran entonces al no ser numéricos.

La manera más sencilla de formar un vector es:

```
> x <- c(1, 2, 3, 4, 5)  # definimos el vector x con numeros
> x <- c('rojo','azul','verde')  # definimos el vector x con cadenas de texto
> x <- 1:10   # secuencia desde el 1 al 10
> x <- seq(1,10)   # secuencia desde el 1 al 10
> x <- seq(10,1,-1)    # secuencia desde el 10 al 1 decreciente
> x <- seq(1,10,0.5)   # secuencia desde el 1 al 10, en incrementos de 0.5
> x <- seq(1,10,len=100)  # secuencia de 1 a 10, dividida en 100 elementos
> x <- rep(1,10)  # vector con 10 elementos iguales a 1.
> x<- c(seq(1,20),rep(1,12))  # secuencia del 1 al 20 y repetición de 1 12 veces.
```

Selección de elementos de un vector:

```
> x <- c(1, 2, 3, 4, 5)  # definimos el vector x
> x[5]   #  muestra el elemento 5
> x[3:5]    # muestra los elementos del 3 al 5
> x[c(3,5)]   # muestra los elementos de la posición 3 y 5
> x[x>3]  # mostrará solo los elementos mayores que 3
> x[x>2 | x <4]  #  muestra solo los elementos mayores que 2 o menores que 4
> x[x>2 &  x <4]  #  muestra solo los elementos mayores que 2 y menores que 4
> x[x!=3]  # mostrará los elementos distintos que 3
> sqrt(x)   # nos devuelve el vector de las raíces cuadradas de cada elemento
            # se aplica la función a cada uno de los elementos
```

Los vectores y otros objetos de R tiene un argumento ***length*** que da el numero de elementos del vector.

```
> length(x)
```

### *Ejercicios 1.2*

- Hallar el seno, logaritmo decimal y neperiano, raíz cuadrada y el valor absoluto de un numero cualquiera.
- Ver el numero ***e*** por medio de ***exp()***
- Formar un vector ***x*** con elementos que vayan de 2 al 20, del 100 al 140, de 2 en 2 y del 45 al 35 en orden descendente.
- Mostrar en pantalla el elemento 5 y el 10 del vector ***x***.
- Ordenar este vector en orden decreciente y obtener su longitud.
- Obtener otro vector ***y*** con los elementos de ***x*** comprendidos entre 20 y 120, ambos incluidos.
- Calcular la suma y el producto de todos los elementos del vector ***x***
- Calcular el máximo, mínimo, media y cuasivarianza del vector ***x***.

## Listas y Matrices (Ver anexo)

## Dataframes

Una ***hoja*** o ***estructura de datos*** o ***marco de datos*** es una lista de un tipo particular llamado **data.frame**, con las siguientes particularidades:

- Los componentes tienes que ser vectores, aunque estos pueden ser numéricos, de caracteres, etc. y todos tienen que tener la misma longitud. También pueden estar compuestos por matrices, listas u otras hojas de datos.

Veremos la forma de construcción de marcos de datos. Si por ejemplo tenemos tres vectores:

```
> facturas<-c(3,4,5,7); recibos<-c(23,45,50,60) ; nombres <- c('Pepe', 'Juan', 'Maria', 'Rosa')   # (tienen que tener el mismo número de elementos)
> marco <- data.frame(fac=facturas, rec=recibos, nom=nombres) # creamos el marco de datos
> marco  # para visualizarlo
```

Se pueden listar como las listas, haciendo referencia siempre a las columnas:

```
> marco[3]   # columna 3 (nom)
> marco$nom  # Otra forma, por la cabecera de la columna

> marco$nom[2]   # Segundo elemento de la columna nom
> marco[[3]][2]   # Otra forma, segundo elemento de la columna 3 (nom)
```

Veremos una aplicación muy importante de los marcos de datos en la lectura y escritura de ficheros.

## Leyendo datos desde archivos

R toma los datos, los ficheros y los scripts del directorio de trabajo. Cuando leamos o escribamos un fichero a no ser que le pongamos la ruta completa lo buscara en el directorio de trabajo.

Como ya sabemos el directorio de trabajo (***wd***) actual lo obtenemos con: ***getwd()***. El directorio de trabajo se puede cambiar desde la mismo GUI o de la forma

```
> setwd('C:/Curso_Estadistica')
```

Si tenemos puesto el directorio que vamos a usar como *wd* en el icono de *R* en Windows rellenando: (*iniciar en:* ***C:/Curso_Estadistica***) cuando empecemos ***R*** ya nos colocará directamente este directorio de trabajo. En Linux es un poco más complicado.

Para este curso vamos a necesitar unos ficheros. Antes de empezar es mejor descargarse la carpeta ***material.zip*** y copiarla y descomprimirla en el directorio de trabajo ***C:/Curso_Estadistica.***

*R* puede leer datos de archivos de texto (ASCII). También puede leer archivos en otros formatos como *Excel*, *csv* y otros muchos más, acceder a base de datos *SQL*, aunque a veces es necesario instalarse paquetes que no están incluidos en el base. Nos centraremos en este apartado en leer y escribir ficheros de texto.

R puede leer datos de archivos de texto (ASCII) con las funciones **read.table, scan** o **read.fwf**.

Cuando trabajemos con ficheros de texto que vengan estructurados en columnas de datos nos vendrá muy bien la función ***read.table***. Esta función lee los datos y los convierte a un ***data.frame***.

Al leer un fichero por ejemplo: ***datos.txt*** de la forma:

| indicativo | lat | lon | temperatura | presion | humedad |
|---|---|---|---|---|---|
| 08221 | 40.46 | -3.56 | 18 | 1015.1 | 60 |
| 08481 | 36.73 | -4.48 | 23 | 1016.2 | 72 |
| 08001 | 43.36 | -8.41 | 15 | 1012.1 | 88 |

……

***datos < -read.table('datos.txt' , header = TRUE)***

si tecleamos ***datos*** R escribirá el fichero completo, pero además los datos de cada columna los asigna a un vector. Estos vectores pueden tener ya un nombre que toma del mismo fichero como indicativo, lat, etc., si hemos indicado: ***header=TRUE*** o sino por defecto toman los nombres *V1*, *V2*, etc. Así si escribimos ***datos$indicativo*** R listaria:

08221 08481 ….

Si no hubiésemos escrito la primera línea con los nombres (***header=FALSE***) habría que nombrarlos como ***datos$V1.*** Un elemento dentro de este vector se llamaría: ***datos$V[3]*** por ejemplo para el elemento 3.

Siempre nos podremos referir a los elementos como ***datos[1]*** al vector o columna 1, y como ***datos[[2]][1]*** al primer elemento de la variable o columna 2.

La función ***read.table*** tiene muchas opciones. La forma general con las opciones más importantes es:

***read.table(file= 'nombre_fichero', header = FALSE, sep ="" , dec = ".",skip = 0, fill = FALSE, comment.char = "#")***

> ***file*** es el nombre del archivo: entre comillas o una variable que contiene el nombre.
> ***Header=FALSE*** Si es TRUE toma la primera línea como nombres de variables.
> ***sep=*** es el separador entre datos, normalmente es "" para espacios en blanco o "\t" tabuladores.
> ***dec="."*** Indica a *R* el formato de los decimales en el fichero punto (defecto), pero se puede poner coma si por ejemplo vienen de una hoja excel.
> ***skip = 0*** Se salta las primeras *n* líneas del fichero, las ignora.
> ***fill*** si es TRUE rellena los espacios en blanco que encuentre.
> ***comment.char = "#"*** ignora las líneas que comiencen con #.

Hay unas variantes de ***read.table*** pensadas para ficheros ***csv*** que son prácticamente iguales pero con unas opciones definidas por defecto:
***read.csv(file, header = TRUE, sep = ",", quote="\, dec=".",fill = TRUE, ...)***
***read.csv2(file, header = TRUE, sep = ";", quote="\, dec=",",fill = TRUE, ...)***

La function ***read.csv*** considera por defecto el separador entre datos como '**,**' mientras que la ***read.csv2*** toma como separador el '**;**'. De cualquier forma todo esto se puede configurar en la function ***read.table***.

La función ***scan*** es más flexible que ***read.table***. A diferencia de esta última es posible especificar el modo de las variables:
***> datos <- scan("data.dat", what = list("", 0, 0))***

En este ejemplo ***scan*** lee tres variables del archivo ***data.dat***; el primero es un carácter (***""***) y los siguientes dos son numéricos (***0***). Por defecto, es decir si se omite el argumento ***what***, ***scan()*** crea un vector numérico. Si ***scan*** no lee el tipo de datos esperados da un mensaje de error. Scan no tiene la opción head por lo que si tiene cabecera habría que saltarla al leer con ***skip=1***. Nos referiremos a los elementos como ***datos[[2]][5]*** para la segunda variable o columna y su elemento numero 5.

La forma con las opciones más importantes es:

***scan(file= 'nombre_fichero', what = double(0), sep ="" , dec = ".",skip = 0, na.strings = "NA", fill = FALSE, blank, comment.char = "#")***

La función ***read.fwf*** puede usarse para leer datos en archivos en formato fijo ancho.

## Guardando datos a archivos

Se suelen usar *archivos de texto*, que son manejables desde cualquier editor.

Existen varias funciones para grabar los datos u objetos del workspace a ficheros.

Una manera sencilla de escribir los contenidos de un objeto en un archivo es utilizando el comando: ***write(x, file="data.txt")*** donde ***x*** es el nombre del objeto (que puede ser un vector o una matriz).

Esta función tiene dos opciones: ***nc*** (o ***ncol***) que define el número de columnas en el archivo (por defecto ***nc=1*** si ***x*** es de tipo carácter, ***nc=5*** para otros tipos), y ***append*** que agrega los datos al archivo sin borrar datos ya existentes (***TRUE***) o borra cualquier dato que existe en el archivo (***FALSE***, por defecto).

Otra función, muy útil, para volcar datos a ficheros es ***write.table***. La función ***write.table*** guarda el contenido de un objeto en un archivo, pero lo normal es que guardamos objetos de tipo ***data.frame***.

Las opciones más importantes son:
***write.table(x, file = 'nombre_fichero', append = FALSE, sep = " ", na = "NA", dec = ".", quote = TRUE, row.names = TRUE,col.names = TRUE)***

> ***x*** es el nombre del objeto a exportar (*dataframe*)
> ***file*** es el fichero donde vamos a escribir
> ***append = FALSE*** significa que borramos el fichero si existiese previamente antes de escribir.
> ***sep = " "*** el separador entre los datos. Normalmente será blanco, coma, punto y coma o tabulador "\t"
> ***row.names/col.names*** pone el nombre o numero de las columnas
> ***quote=TRUE*** escribe los cadenas de caracteres con comillas.

### *Ejercicios 1.3*

- Leer el fichero de ejemplo: '***ESTACIONES_radiacion.dat'*** con ***read.table*** asignándosela a una nueva variable ***datos***, sin la opción ***header=FALSE*** (por defecto)
- Listar el *dataframe* ***datos*** entero, despues la *segunda columna*, y luego el *tercer elemento* de esa columna.
- Haced lo mismo, leyendo primero el fichero con ***read.table***, pero añadiendo la opción ***header=TRUE***).
- Listar el *dataframe* ***datos***, después la *segunda columna*, y luego el *tercer elemento* de esa columna y observar las diferencias con el punto anterior.
- Escribir el *dataframe* ***datos*** leída anteriormente con ***write.table*** en un fichero llamado ***salida.txt***, desactivando las comillas en los caracteres y omitiendo

nombres de columnas y filas ***(row.names = FALSE, col.names=FALSE, quote=FALSE).*** Ver el fichero creado con un editor de texto.

- Hacer lo mismo pero escribiendo solo la columna ***NOMBRE*** en el fichero.

# R desde scripts

Para ciertas tareas sencillas con R basta con introducir órdenes desde la pantalla de comandos, pero cuando empezamos a hacer labores más complicadas, repetitivas o que necesitan introducir muchos comandos de R se hace imprescindible el trabajar a través de *programas* o *scripts*. Estos son simplemente ficheros de texto donde se escriben todas las órdenes que pensamos ejecutar, datos y objetos. Luego al final se ejecutan todas las órdenes del script secuencialmente.

Además de permitir trabajos repetitivos o extensos los scripts permiten trabajar con el *lenguaje de programación de R*, donde se disponen de *if* condicionales, bucles, definición de funciones y otras funcionalidades de cualquier lenguaje informático. Solo vamos a ver algunas cosas, pero como R está muy bien documentado se dispone de abundante información.

Los scripts tienen la extensión **.R** para que se interpreten correctamente tanto por R como por otros programas externos. Si necesitamos cargar algún librería se hace al principio con ***library('nombre_libreria')***.

Lógicamente los scripts tiene que estar en el directorio de trabajo o sino poner la ruta completa en el ***source().***

Para escribir un script nuevo desde el *GUI* en el menú: *Archivo/nuevo script* y luego *salvar como* donde se le pone el nombre que queramos. Una vez que esté abierto en el menú *Ventanas: dividir horizontalmente*. De forma que tendremos el script arriba y la ventana de comandos abajo.

Se puede *ejecutar* el script entero desde el menú *Editar/Ejecutar todo* o en la ventana de comandos ***source('nombre_script.R')***. Hay que salvarlo previamente con ***CTRL+S*** o *Archivo/guardar*)

Si solo se quiere ejecutar una línea del script donde tengamos el cursor ***CTRL+R*** o ***F5***.

Otra forma de trabajar es crear un fichero en la carpeta de trabajo con un editor de texto que reconozca el lenguaje de programación R como el ***notepad++,*** salvarlo y ejecutarlo desde líneas de comandos con ***source()***

## *Ejemplos de scripts en R*

Abrir en el *GUI* un script nuevo y copiar primero el script del ejemplo 1 que viene a continuación y ejecutarlo con:

```
> source('prueba1.R')
```

o desde el *GUI*. Después hacer lo mismo con el script del Ejemplo 2.

***Ejemplo1:***

```
# Script de prueba: prueba1.R (Sep-2018)
```

```
# va a mostrar el directorio actual
a <- getwd()
print('El directorio actual es:')
print(a)

# va a mostrar los ficheros dentro del directorio
print(paste('Los directorios dentro de ',a,' son:'))
print(dir())

# paste() sirve para unir tanto cadenas de texto (character) y variables de cualquier tipo.
# van separadas por comas. Deja un espacio en blanco entre cadenas a no ser que se ponga al
final  ,sep=' '
```

***Ejemplo 2:***

```
# Script de prueba: prueba2.R (Sep-2018)

# Define unas variables
n_inicial = 9
n_final = 8

if ( n_final < n_inicial){
  cat (' No se puede hacer el bucle, el num. final es menor que el inicial \n')
  stop(' El programa acabo mal')
}

# hace un bucle desde n_inicial a n_final
for (numero in n_inicial:n_final){
  # escribe el cuadrado del numero
  print(paste('El cuadrado de ',numero,' es ',numero*numero, sep=''))
}
```

***Ejercicio 1.4***

- Vamos a crear un script de prueba, ***prueba.R*** que cree un vector ***a*** de 7 números (reales o enteros) y primero que lo visualice en pantalla, después que calcule la media de los elementos del vector y los muestre en pantalla (con la explicación de lo que está mostrando). Por último que calcule el valor mínimo del vector y lo visualice.

*Opcional:* Mediante un bucle ***for*** calcular y mostrar la suma de los cuadrados de todos los elementos del vector.

## 1.3 Evaluación y visualización de datos: gráficos

# GRAFICOS EN R

*R* ofrece una enorme cantidad de gráficos. Además posee una gran flexibilidad para generar gráficos a medida con un gran número de opciones para cada una de sus funciones de dibujo. El sistema permite desde gráficos muy simples a figuras de gran calidad para incluir en artículos o libros.

## Introducción a los gráficos

Los resultados de una función gráfica son enviados a un dispositivo de representación que puede ser una ventana o un fichero de imagen en distintos formatos.

Existen *dos tipos de funciones gráficas*: ***de alto nivel*** que crean una nueva gráfica y las funciones ***de bajo nivel*** que agregan elementos a una grafica ya existente. Las gráficas se producen con respecto a parámetros gráficos que están definidos por defecto y pueden ser modificados con la función ***par***.

Primero veremos como manejar gráficos y dispositivos gráficos; después veremos en detalle algunas funciones graficas y sus parámetros.

## Manejo de gráficos

Cuando se ejecuta por primera vez una función grafica *R* abre una ventana para mostrar el grafico. El dispositivo grafico por defecto son las ventanas bajo *Windows* o *X11* en sistemas operativos *Linux*.

El último dispositivo abierto se convierte en el dispositivo activo sobre el que irán todas nuestras acciones.

La función ***dev.list()*** muestra los dispositivos abiertos. Para saber el dispositivo activo: ***dev.cur().*** Si se quiere cerrar un dispositivo: ***dev.off (num)*** donde ***num*** es el número de dispositivo. Si no se pone nada se cierra el activo: ***dev.off().*** Para abrir un dispositivo nuevo: ***dev.new()***

Además de visualizar los gráficos en el ordenador también se pueden convertir, si se desea, en archivos gráficos en multitud de formatos como ***pdf***, ***png***, ***jpeg***, etc.

## Funciones gráficas de alto nivel

Vamos a describir unas cuantas funciones graficas de alto nivel aunque existen muchas más.

- ***plot(x)*** representa los valores de vector *x* en el eje y ordenados en el eje x.
- ***plot(x,y)*** grafico divariado de *x* (en el eje x) frente a *y* (en el eje y). Tienen que tener la misma longitud *x* e *y*.
- ***pie(x)*** grafico de tarta o de pastel.
- ***boxplot(x)*** grafico de cajas
- ***hist(x)*** histograma de frecuencias de x
- ***barplot(x)*** histograma de los valores de x

Las opciones y argumentos para cada una de estas opciones se pueden encontrar en la ayuda incorporada en R. Por ejemplo:

```
> ?plot o help(plot)
```

Para dibujar una función cualquiera primero se crea un vector con los puntos del eje de abscisas (x) y luego se representa la función. Por ejemplo para pintar el ***cos(x),*** creamos un vector entre ***–pi*** y ***pi*** con muchos puntos (50 o más)

```
> x <- seq(-pi,pi,len=200)
```

Y después ploteamos la función ***cos(x)*** frente a ***x***. El type ***'l'*** es para que una los puntos con una línea continua:

```
> plot(x, cos(x), type='l')
```

Existe otra función en R, ***curve()***, que nos pinta directamente cualquier función en un intervalo, sin necesidad de crear primero un vector para dibujar el eje x, simplemente dando la función y el intervalo del eje x. La función ***curve*** admite los mismos parámetros que ***plot***, como títulos, colores, etc. El caso anterior lo plotearíamos así:

***> curve (cos, -pi, pi)***

Algunas de estas opciones son idénticas para varias funciones graficas; estas son las principales:

***type="p"*** especifica el tipo de línea al pintar el grafico***;***

- ***"p":*** puntos (por defecto)
- ***"l":*** líneas
- ***"b":*** puntos conectados por líneas
- ***"o":*** igual al anterior, pero las líneas están sobre los puntos
- ***"h":*** líneas verticales
- ***"s":*** escaleras, los datos se representan como la parte superior de las líneas verticales
- ***"S":*** igual al anterior pero los datos se representan como la parte inferior de las líneas verticales

***xlim=, ylim=*** especifica los lımites inferiores y superiores de los ejes; por ejemplo con: ***xlim=c(1, 10)*** o ***xlim=range(x)***

***xlab=, ylab=*** títulos (etiquetas, labels) en los ejes; deben ser variables de tipo carácter

***main=*** título principal; debe ser de tipo carácter

***sub=*** subtitulo (escrito en una letra más pequeña y en la parte inferior)

*Nota:* A todas estas opciones se le pueden añadir las de los *parámetros gráficos* que se verán más adelante, como ***col='red'*** (color de los símbolos), ***cex=0.9*** (tamaño símbolos), ***lwd=1.1*** (anchura de las líneas), etc.

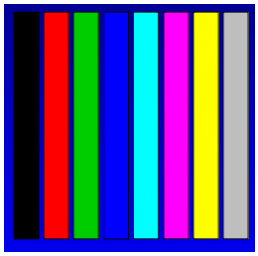
***axes=TRUE*** si es FALSE no dibuja los ejes ni la caja del grafico

***ann=TRUE*** si es FALSE no dibuja anotaciones en los ejes

## Colores en R

Los colores que se utilizan en los gráficos pueden designarse en R de diversas maneras. La forma más sencilla es la ***paleta básica***.

***> barplot (rep(1,8), yaxt="n", col=1:8) # pintamos un diagrama de barras con los numeros del 1 al 8 y con los colores de la paleta básica del 1 al 8. (yaxt='n' es para que no pinte el eje y)***



Para ver los colores asignados a la ***paleta básica***:

***> palette()***

También se pueden definir los colores por su nombre:

***colores <- c('blue','red','yellow')***

***> barplot (rep(1,3), yaxt="n", col=colores)***

Para ver la lista de colores con sus nombres:
***> colors()***

(Para saber más sobre colores ver: ***Notas.Sueltas.sobre.R***)

### *Ejercicios 1.5*

- Representar la función ***sin(x)*** entre ***-π*** y ***π***
- Crear un vector ***y*** de 10 elementos aleatorios entre 1 y 50 con la función ***runif (n, a1, a2)*** que genera *n* números aleatorios *reales*, entre *a1* y *a2,* reales (***?runif***) o con la función ***sample (a1:a2, n)*** que es igual que la anterior pero con números aleatorios *enteros*.
- Plotear el vector ***y*** con ***plot(y)***
- Plotear ***plot(y)*** variando el tipo de gráficos: puntos, líneas, etc., con la opción ***type=***
- Igual pero que solo represente los valores entre x=2 y x=8
- Mostrar en el ploteo unos títulos para cada eje, un tipo principal y un subtítulo.
- Poner las líneas o puntos en colores, y variar el tamaño de los puntos o símbolos y la anchura de las líneas.
- Dibujar un gráfico de tarta, un gráfico de cajas (*boxplot*), histograma de frecuencias y de valores con el vector ***y.***

## Salvar un gráfico a fichero

***R***, además de visualizarlos en pantalla, permite salvar los gráficos en ficheros de diversos formatos: ***png***, ***jpg***, ***pdf***, ***bmp***, ***tiff***, etc.

El método consiste en escribir, antes del ploteo, la orden con el tipo de formato de salida que queremos para el grafico, por ejemplo para obtener un fichero en formato ***png***:

***png ('nombre_fichero.png')***

después se pinta el grafico:
***plot()***

y terminar cerrando el dispositivo gráfico:
***dev.off()***

En el nombre del fichero se puede incluir la ruta completa si lo queremos dejar en otro lugar distinto del directorio de trabajo.

Si queremos un fichero ***jpg*** usaríamos ***jpeg ('nombre_fichero.jpg')***

Las órdenes admiten diversas opciones como la resolución, la anchura o la altura de la imagen. Estas opciones dependen del formato. Hay que mirar en la ayuda de cada una.
***help(png)***

Para el formato ***png*** se pueden usar:
***png('nombre_fichero.png',res=130, width = 910, height = 690,units="px")*** donde se obtendrá una imagen de *910x690* pixeles con una resolución de *130*.

Para ficheros ***jpeg*** admite también *width* y *height* pero en vez de resolución se usa *quality* para elegir la compresión del ***jpg*** entre 0 y 100.

***Ejemplo:***

```
x <- sample(1:10)  # creamos un vector x con números aleatorios de 1 al 10
png('prueba.png')  # ponemos el formato de salida a png
plot(x)   # hacemos un gráfico sencillo con el vector x (no se abre la pantalla ni se crea el
            grafico todavía.
dev.off() # se crea en el directorio de trabajo un fichero prueba.png con el ploteo
```

## Funciones gráficas de bajo nivel (ver anexo)

# ANEXO

## Listas

Es una colección ordenada de objetos que no tienen porque ser del mismo tipo, así una Lista puede estar compuesta de un vector numérico, un valor lógico, una matriz y una función. Se definen con la función ***list***

***Milista<-list(jefe='Juan',***
***encargado='Pedro',num.trabajadores=4,edad.trabajadores=c(21,34,48,50))***

En este caso **Milista** tiene 4 componentes, referidos como **Milista[[1]]** etc. Como ademas el elemento 4 es un vector nos referiremos a cada componente como **Milista[[1]][n]** donde n es el numero de componente.

La función **length()** aplicada a una lista devuelve el número de componentes (del primer nivel) de la lista.

Los componentes de la lista pueden tener nombre y en ese caso nos podemos también referir a ellos por su nombre. En el ejemplo anterior la componente **Milista$jefe** es igual que **Milista[[1]]** y vale "Juan". Si no se incluye el nombre, los componentes estarán automáticamente numerados. No hace falta referirnos con el nombre completo, basta con solo decir las primeras letras de forma que no haya confusión con los otros componentes. Por ejemplo podemos decir **Milista$enc** para **Milista$encargado**

También es posible utilizar los nombres de los componentes entre dobles corchetes, por ejemplo, **Lst[["jefe"]]** coincide con **Lst$jefe**. Esta opción es muy útil en el caso en que el nombre de los componentes se almacena en otra variable, como en
***x <- "jefe"; Lst[[x]]***

## Matrices

Una matriz (*matrix*) es una colección de datos del mismo tipo de dos dimensiones. (Los arrays son matrices, pero con más de dos dimensiones).

La matriz se crea con la función **matrix**:

***matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE)***

Si no se ponen datos a rellenar la completa con ***NA*** (valores not avalaibles). La opción **byrow =FALSE** (por defecto) indica que los valores van llenando la matriz por columnas.

```
> matrix(data=6, nrow=3, ncol=2)
     [,1] [,2]
[1,]  6    6
[2,]  6    6
[3,]  6    6

> c <-  matrix(1:6, 2, 3)
     [,1] [,2] [,3]
[1,]  1   3    5
[2,]  2   4    6
```

La dimensión de la matriz se obtiene con ***dim(c)***

Si tenemos dos o más vectores del mismo tipo, numéricos o de caracteres, y tienen la misma longitud se pueden combinar para formar una matriz con ***cbind***

```
> x <- c(1,2,3,4); y <- c(5,6,7,8)
> c <- cbind(x,y)
```

***cbind*** une los vectores por columnas. Existe la función ***rbind*** que los une por filas.

A los elementos de una matriz se accede de igual manera que los vectores pero con dos indices, el primero hace referencia a la fila y el segundo a la columna.

```
> c[1,2]
[1] 3
```

Si queremos acceder a una fila completa seria***: c[1,]*** y la columna ***c[,2]***

## Operaciones con matrices:

Si creamos dos matrices, por ejemplo:

```
A <- matrix(data=c(2,0,1,3,0,0,5,1,1),nrow=3,ncol=3,byrow=T)
B <- matrix(data=c(1,0,1,1,2,1,1,1,0),nrow=3,ncol=3,byrow=T)
> A %*% B   # producto de matrices (dos matrices se dicen multiplicables si el número de columnas de A coincide con el de filas de B, ver Nota abajo)
> A + B   # suma de matrices (cuando tienen las mismas dimensiones)
> t(A)   #  traspuesta de la matriz A
> det(A)   # determinante de la Matriz (solo matrices cuadradas)
> diag(A)  #  devuelve un vector con los elementos de la diagonal de la matriz A
```

### *Ejercicios Anexo.1*

- Formar una matriz 5 por 6 con valores del 1 al 30 pero rellenándola por filas.
- Crear 2 matrices A y B y hallar su producto C. Mostrar después la dimensión, la transpuesta, su determinante y la matriz diagonal de C. Mostrar también el elemento de la fila 1,columna 2 de C y toda la fila 1 de C.

*Nota: se recuerda que dos matrices son multiplicables si el número de columnas de A coincide con el número de filas de B. (Anxm * Bpxn = Cpxm siendo n el numero de filas en A y m el num. de columnas en A y el elemento Cjxi se obtiene multiplicando la fila j de A por la columna i de B )*

## Funciones gráficas de bajo nivel

Son funciones para modificar los gráficos hechos con alguna función de alto nivel, para mejorar el gráfico con otros elementos más específicos u otros detalles mas finos, e incluso hacer un grafico desde cero (habría que empezar por abrir una ventana nueva de gráfico con ***plot.new()***). Si no hay ningún grafico abierto, antes hay que hacer un ploteo. Si se quiere un ploteo vacío, poner ***type="n"***, por ejemplo: ***plot(1:10,1:10,type="n").***

Las funciones de bajo nivel más usadas son las que se muestran a continuación. En algunos el par (*x,y*) puede ser un solo punto o a veces vectores de la misma longitud.

| | |
|---|---|
| ***points(x,y)*** | Agrega puntos, se puede usar la opción (***type=)*** |
| ***lines(x,y)*** | Igual que la anterior pero con líneas. |
| ***text(x, y, 'texto')*** | Agrega texto en la posición *x,y* |

| | |
|---|---|
| ***mtext(text, side=3, line=n, ...)*** | Agrega texto dado por *text* en la posición *side* y en la línea que se especifique. |
| ***segments(x0,y0,x1,y1)*** | Agrega una línea desde el punto (*x0,y0*) hasta el (*x1,y1*) |
| ***arrows(x0, y0,x1, y1, angle=30)*** | Igual que el anterior pero con flechas y un ángulo. |
| ***abline(h=y)*** | Dibuja una línea horizontal en la posición del eje y = *y* |
| ***abline(v=x)*** | Dibuja una línea vertical en la posición del eje x = *x* |
| ***rect(x1, y1, x2, y2)*** | Rectángulo en la posición indicada |
| ***polygon(x, y)*** | Polígono uniendo los puntos dados en *x,y* |
| ***legend(x, y, legend)*** | Dibuja una leyenda en (*x,y*) con los símbolos dados en legend ('*','.','P', etc) |
| ***title()*** | Agrega un titulo y si se quiere un subtitulo |
| ***axis(side,vect)*** | Agrega un eje en la parte inferior (*side=1*), izquierda (2), superior (3), o derecha (4); vect (opcional) da la abscisa (u ordenada) donde se deben dibujar los marcadores ('*tick marks*') del eje |

La función ***text*** sirve también para incluir como un dibujo en el grafico fórmulas matemáticas si se escribe la función expresión. Por ejemplo:

***text(x, y, expression(p == over(1, 1+eˆ-(beta*x+alpha))))***

se vería en la grafica como la siguiente ecuación en el punto de coordenadas (x,y):

$$p = \frac{1}{1 + e^{-(\beta x + \alpha)}}$$

***Ejercicios Anexo.2:***

- Crear un ploteo vacío con 10 puntos en el eje x y 10 en el eje y
- Pintar puntos, líneas, textos, segmentos, flechas, líneas horizontales y verticales, rectángulos, etc.
- Opcional: escribir una fórmula de ejemplo en un punto del gráfico.

## Parámetros gráficos

Además de la utilización de comandos de bajo nivel, la presentación de gráficos se puede mejorar con parámetros gráficos adicionales.

Estos se pueden utilizar como opciones de funciones graficas (pero no funciona para todas), o usando la función ***par*** para cambiar de manera permanente opciones generales en los gráficos. La lista completa de parámetros gráficos se puede ver con ***?par***.

Cualquier cambio que hagamos en un parámetro grafico general, *se aplicara a todos los gráficos siguientes* por lo es conveniente antes de cambiar algún parámetro grafico hacer una copia de los valores actuales y cuando terminemos restaurar la copia inicial. Esto se hace con:
***old.par <- par(no.readonly = TRUE)*** o simplemente ***old.par <- par()*** *para salvarlos*.

Cuando queramos *recuperar* otra vez los que teníamos al principio: ***par(old.par)***

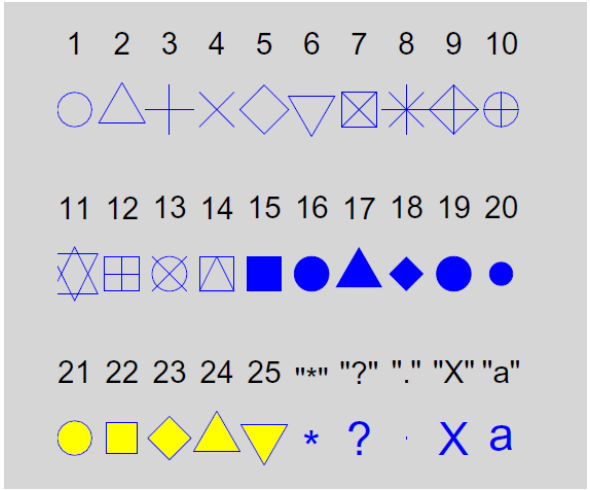
De cualquier forma si salimos y volvemos a entrar de R se pondrán de nuevo los gráficos por defecto.

Por ejemplo, el siguiente comando: ***par(bg="yellow")*** dará como resultado que todos los gráficos siguientes tengan el fondo de color amarillo.

Mostraremos algunos de ellos

| | |
|---|---|
| ***adj*** | *controla la justificación del texto (0 justificado a la izquierda, 0.5 centrado, 1 justificado a la derecha)* |
| ***bg*** | *especifica el color del fondo (ej. **: bg='red'**) La lista de colores disponibles se puede ver con **colors()*** |
| ***bty*** | *controla el tipo de caja que se dibuja alrededor del gráfico: ej: ¨o", "l" (la caja se parece a su respectivo carácter); si **bty="n"** no se dibuja la caja* |
| ***cex*** | *un valor que controla el tamaño del texto y símbolos con respecto al valor por defecto; los siguientes parámetros tienen el mismo control para números en los ejes: **cex.axis**, títulos en los ejes: **cex.lab**, el título principal: **cex.main**, y el subtítulo: **cex.sub*** |
| ***col*** | *color de los símbolos ploteados; para controlar el color en otros elementos: **col.axis, col.lab, col.main** y **col.sub*** |
| ***font*** | *un entero para el estilo del texto (**1: normal, 2: cursiva, 3: negrilla, 4: negrilla cursiva)**; como en cex existen**: font.axis, font.lab, font.main y font.sub*** |
| ***las*** | *un entero que controla la orientación de los caracteres en los ejes **(0: paralelo a los ejes, 1: horizontal,2: perpendicular a los ejes, 3: vertical)*** |
| ***lty*** | *un entero o carácter que controla el tipo de las líneas; **(1: solida, 2: quebrada, 3: punteada, 4: punto-línea, 5: línea larga-corta, 6: dos líneas cortas)*** |
| ***lwd*** | *un numero que controla la anchura de las líneas (def=1)* |
| ***mar*** | *un vector con 4 valores numéricos que controla el espacio entre los ejes y el borde de la grafica en la forma **c(inferior, izquierda, superior, derecha)**; los valores por defecto son c(5.1, 4.1, 4.1, 2.1)* |
| ***mgp*** | *un vector con 3 valores numéricos que controla el espacio entre los ejes y las etiquetas (labels) de los ejes (primer numero), o entre los ejes y los números en cada eje (segundo numero).* |
| ***pch*** | *controla el tipo de símbolo, ya sea un entero entre 1 y 25, o un carácter (Ver figura abajo)* |
| ***ps*** | *un entero que controla el tamaño (en puntos) de textos y símbolos* |
| ***xaxt*** | *si **xaxt="n**" el eje x se coloca pero no se muestra (útil en conjunción con **axis**)* |
| ***yaxt*** | *si **yaxt="n"** el eje y se coloca pero no se muestra (útil en conjunción con **axis**)* |

*Tipos de símbolos* con los que pintar los gráficos:




- *Figura 2: Los símbolos gráficos en R (pch=1:25). Los colores se obtuvieron con las opciones col="blue", bg=" yellow"; la segunda opción tiene efecto solo sobre los símbolos 21–25. Se puede usar cualquier carácter (pch="*", "?", ".", . . . ).*

# Algunas direcciones interesantes sobre R

## *R en general*

***Página oficial:***
https://www.r-project.org/
https://cran.r-project.org/

***Quick-R :***
https://www.statmethods.net/

***R-bloggers:***
https://www.r-bloggers.com/

***Gráficos***
https://www.r-graph-gallery.com/

***Páginas de muestra hechas por el profesor con R***
http://172.24.132.61/~eyd/pred_probabilistica/
http://172.24.132.61/~eyd/mes/index.html
http://172.24.132.61/~eyd/estadistica.rayos/