Tema 5:

# Gráficos

Antes de generar un gráfico y representarlo, es necesario tener abierto un dispositivo que los admita (y que éste sea el dispositivo al que van a dirigirse los resultados gráficos, lo que se denomina como dispositivo gráfico actual, de entre los varios que pueden estar abiertos).

## 5.1.Dispositivos gráficos

La orden **?Devices** indica qué dispositivos gráficos están disponibles en el sistema en que trabajamos. En la versión de MSW, puede utilizar las funciones **x11**. En cada caso, se abre el dispositivo gráfico y no se devuelve nada a R.

### 5.1.1. Función x11

La función **x11** abre una (nueva) ventana gráfica a la que irán dirigidos los resultados gráficos desde ese momento. Los argumentos de la función, todos opcionales, son width, height y pointsize.

- **width** corresponde a la anchura lógica del eje X, medida en pulgadas, que automáticamente se hace corresponder a la anchura de la ventana. Ello importa para el momento en que este gráfico se pasa a impresora.
- **height** hace el mismo papel respecto de la altura.
- **pointsize** hace referencia al tamaño del punto de la fuente.

### 5.1.2. Función win.print

La función **win.print** genera gráficos directamente sobre el administrador de impresión del sistema. Podrá elegir en qué impresora de las instaladas y de qué modo se realizará la impresión. Así pues, si se ha instalado una impresora Postscript, podrá también realizar la impresión en un archivo Postscriptpor por este procedimiento.

### 5.1.3. Función pictex

La función **pictex** genera gráficos que pueden utilizarse en TeX y en LaTeX. Su forma es

**pictex(file = "Rplots.tex", width = 5, height = 4, debug = FALSE, bg = "white", fg
"black")**

Los argumentos son los siguientes:

- **file** es el nombre del archivo donde se almacenará el gráfico. Su valor predeterminado es Rplots.tex".
- **width** es la anchura, medida en pulgadas. Su valor predeterminado es 5.
- **height** es la altura, medida en pulgadas. Su valor predeterminado es 4.
- **debug** indica si debe imprimirse información para depuración. Su valor predeterminado es FALSE.
- **bg** es el color del fondo. Su valor predeterminado es "white".
- **fg** es el color de la tinta. Su valor predeterminado es "black".

Para utilizarlo en LaTeX, debe incluir la biblioteca pictex, escribiendo la orden

```
\usepackage{pictex}
```

tras lo cual puede incluir el gráfico, por ejemplo, como una figura flotante con las órdenes

```
\begin{figure}[htp]
\centerline{\input{Rplots.tex}}
\end{figure}
```

Si desea utilizarlo en TeX, deberá incluir la biblioteca pictex, escribiendo la orden

```
\input pictex
```

tras lo cual puede incluir el gr áfico, por ejemplo, centrado, con la orden

```
$$ \input Rplots.tex $$
```

### 5.1.4. Funciones bmp, jpeg, png

Cada una de estas funciones genera un gráfico en el formato que indica su nombre. Debe tener en cuenta que para el formato jpeg}, R realiza una compresión del archivo, de tal modo que el gráfico resulta modificado, por lo que debe comprobar siempre si el mismo es correcto.

La forma de las funciones es

```
bmp(filename="Rplot.bmp", width=480, height=480, pointsize=12)
jpeg(filename="Rplot.jpg", width=480, height=480, pointsize=12, quality=75)
png(filename="Rplot.png", width=480, height=480, pointsize=12)
```

- **filename** es el nombre del archivo donde se almacenará el gráfico.
- **width** es la anchura del gráfico, medida en pixels.
- **height** es la altura del gráfico, medida en pixels.
- **pointsize** es el tamaño del punto para dibujar textos.
- **quality** sólo se aplica a la función **jpeg**. En este formato se realiza una compresión y este parámetro indica, mediante un porcentaje, la calidad del gráfico. Cuanto menor es el valor, más se comprime la imagen, y previsiblemente se obtiene peor calidad.

### 5.1.5. Funciones win.metafile

La función **win.metafile** imprime el gráfico en un archivo. Su forma es

```
win.metafile(filename = "", width = 7, height = 7, pointsize = 12)
```

- **filename** es el nombre del archivo de salida, que contendrá un archivo en formato 'Microsoft Windows metafile', y cuya extensión habitualmente es `.emf' o `.wmf'. Si el valor de este argumento es una cadena vacía, el gráfico se pasa al portapapeles, desde donde podrá copiarse a otro programa.

- **width** corresponde a la anchura lógica del eje X, medida en pulgadas, que automáticamente se hace corresponder a la anchura de la ventana. Ello importa para el momento en que este gráfico se pasa a impresora.
- **height** hace el mismo papel respecto de la altura.
- **pointsize** hace referencia al tamaño del punto de la fuente.

### 5.1.6. Función postscript

La función **postscript** genera gráficos que pueden utilizarse en cualquier dispositivo que acepte el lenguaje postscript. En particular es útil para impresoras postscript, y para incluir el gráfico en un procesador de textos. También puede utilizarlo en un intérprete postscript, como Ghostscript y Gsview.

Sus argumentos son

- **file** es el nombre del archivo donde se almacenará el gráfico
- **paper** es el tamaño del papel. Las posibilidades son ''a4'', ''letter'', ''legal'', ''executive'', y ''special''. En este último caso, será necesario definir el tamaño (anchura y altura) del papel.
- **horizontal** es la orientación del papel. Si es T, valor predeterminado, la orientación será apaisada; si es F, será vertical.
- **width, height** son la anchura y altura, medidas en pulgadas. Los valores predeterminados son las medidas de la página, menos un borde de 0.25 pulgadas.
- **family** es la familia de fuentes a utilizar. Puede seleccionar ''AvantGarde'', ''Bookman'', ''Courier'', ''Helvetica'', ''Helvetica-Narrow'', ''NewCenturySchoolbook'', ''Palatino'' o ''Times''.
- **pointsize** es el tamaño de punto a utilizar.
- **bg** es el color del fondo.
- **fg** es el color de la tinta.
- **onefile** si es T, valor predeterminado, permite almacenar varias figuras en un archivo, si es F, sólo permite almacenar una, y lo hace en formato EPS, que es el que debe utilizar para poder insertar la imagen en un procesador de textos.
- **pagecentre** si es T, valor predeterminado, centra el gráfico en la página.

### 5.1.6. Ejemplos

Las siguientes órdenes crean tres gráficos, en tres ventanas distintas, con las realizaciones de tres procesos no lineales a través de las correspondientes funciones estudiadas anteriormente. Los resultados se muestran en las figuras \ref{Dong1}, \ref{Dong2} y \ref {Dong3}.

```
> dong1.datos<-dong1(10000)
> dong2.datos<-dong2(10000)
> dong3.datos<-dong3(10000)
> win.graph()
> plot(dong1.datos,col="yellow")
> win.graph()
```

```
> plot(dong1.datos,col="blue")
> win.graph()
> plot(dong3.datos,col="red")
```

#### 5.1.7. Ejemplos

Las siguientes órdenes crean tres gráficos, en tres ventanas distintas, con las realizaciones de tres procesos no lineales a través de las correspondientes funciones estudiadas anteriormente. Los resultados se muestran en las figuras \ref{Dong1}, \ref{Dong2} y \ref {Dong3}.

```
> dong1.datos<-dong1(10000)
> dong2.datos<-dong2(10000)
> dong3.datos<-dong3(10000)
> win.graph()
> plot(dong1.datos,col="yellow")
> win.graph()
> plot(dong1.datos,col="blue")
> win.graph()
> plot(dong3.datos,col="red")
```

### 5.2. Función graphics.off

La función **graphics.off()** cierra todos los dispositivos gráficos. Esta función, como efecto adicional, termina de dibujar los gráficos pendientes, como puede ocurrir al mandar un gráfico a **win.print**.

A continuación veremos algunas funciones de las mcuhas disponibles en el libro **graphics**.

### 5.3. Función plot

**plot** es una función genérica que crea un gráfico en el dispositivo gráfico actual. Además existen funciones específicas en las que funciona de modo especial, como por ejemplo para **data.frame**, **lm**, etc. La forma de uso habitual es **plot(x, ...)**, donde **x** es uno o varios objetos. Además es posible utilizar diferentes parámetros, lo que se hace con la función **par**.

Por ejemplo, las dos órdenes siguientes representan, respectivamente, un tramo de la parábola y=x^2, y todas las parejas de gráficos bidimensionales de las variables incluidas en la hoja de datos **h.datos2** , definida anteriormente.

```
> win.graph()
> plot(1:100,(1:100)^2, type="l")
> win.graph()
> plot(h.datos2)
```

### 5.4. Función text

Esta función añade texto a un gráfico existente. Su sintaxis es

**text (x, y = NULL, labels = seq(along = x), adj = NULL, pos = NULL, offset**
**0.5,vfont= NULL, cex = 1, col = NULL, font = NULL, ...)**

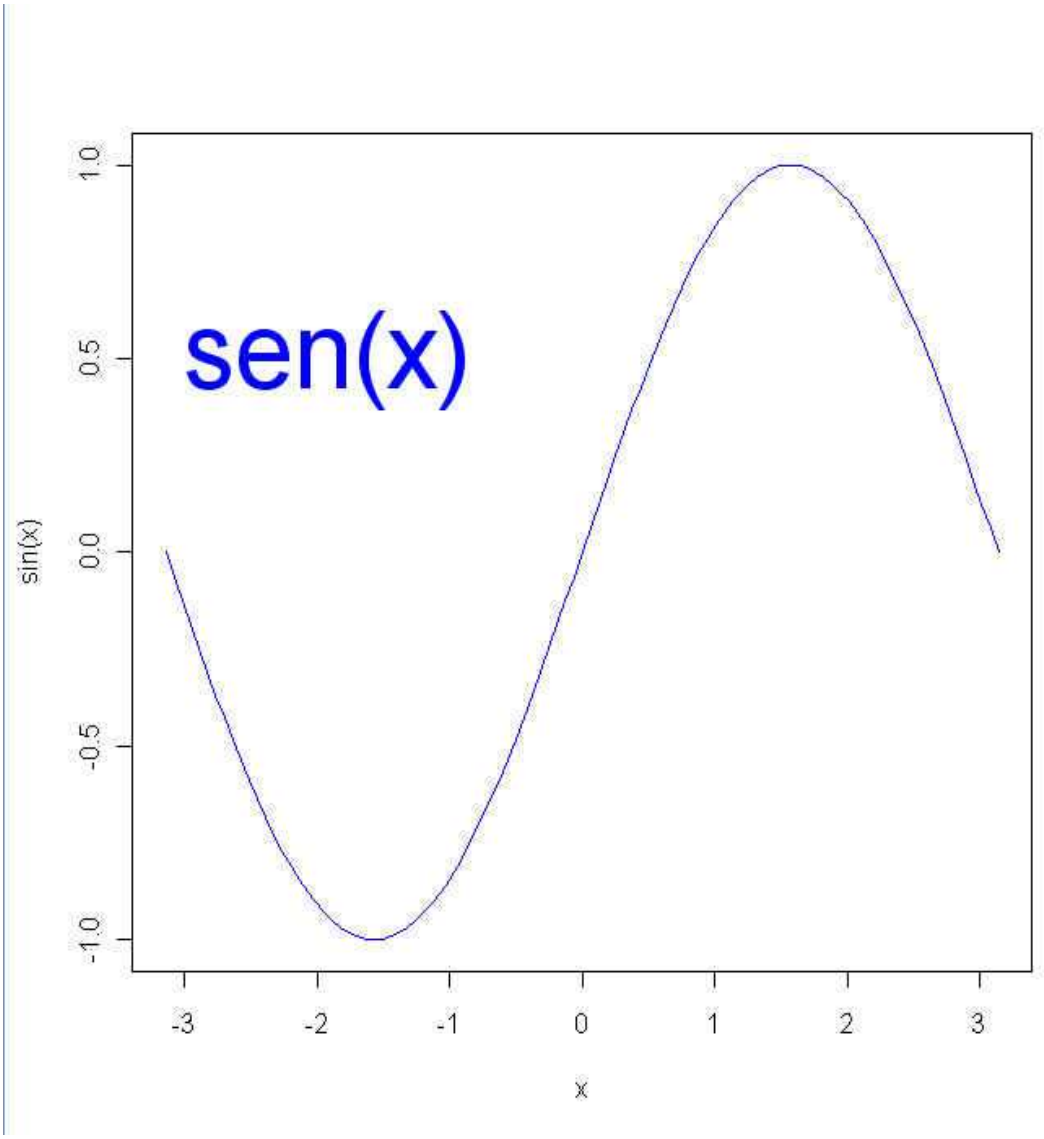
siendo los argumentos más importantes los siguientes:

- **x** e **y** son vectores numéricos de coordenadas en que deben ser escritas las etiquetas de texto.
- **labels** es un vector de tipo carácter o una expresión que especifica el texto que se desea escribir.
- **adj** es uno o dos valores del intervalo [0,1] que especifican el ajuste de las etiquetas en x e y respectivamente. El valor predeterminado es c(0.5,0.5) que significa centrado en ambas coordenadas.
- **pos** toma valores 1,2,3 o 4 para indicar que el texto debe estar debajo, a la izquierda, encima o a la derecha de las coordenadas especificadas.
- **cex** es el factor de expansión de los caracteres.
- **col** y **font** son el color y la fuente.

El texto puede rotarse respecto del centro definido por **adj** mediante el parámetro gráfico **srt**

Por ejemplo, las órdenes siguientes producen el gráfico de la figura.

```
curve(sin(x),-pi,pi,col="blue")
text(-3,0.5,"sen(x)",col="blue",cex=4,adj=0)
```




## 5.5. Función symbols

Esta función permite dibujar círculos, cuadrados, rectángulos, estrellas, termómetros y cajas en una posición determinada de un gráfico, indicando además el tamaño que deben tener. Su sintaxis es

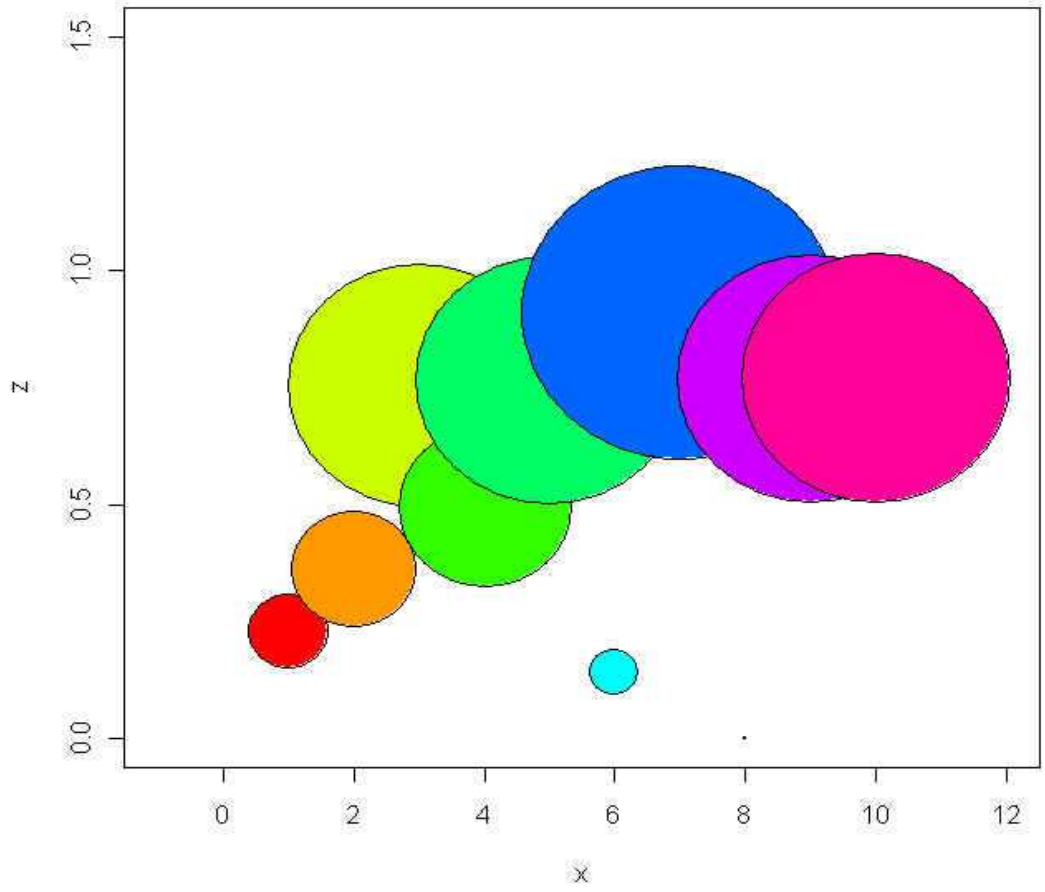
**symbols(x, y = NULL, circles, squares, rectangles, stars, thermometers, boxplo**
**inches = TRUE, add = FALSE, fg = par("col"), bg = NA, xlab = NULL, ylab = NU**
**main = NULL, xlim = NULL, ylim = NULL, ...)**

Los valores de **circles, squares, rectangles, stars, thermometers, boxplots**, son las medidas de los símbolos que se desea representar, **fg** y **bg** corresponden, respectivamente, al color de tinta y el de fondo.

Las siguientes órdenes generan 10 valores de la distribución uniforme entre 0 y 10 y los representan mediante círculos de colores, como puede observarse en la figura.

```
n = 10
x = 1:n
palette(rainbow(n))
```

```
z = runif(10)
symbols(x,z,circles=z,xlim=c(-1,12),ylim=c(0,1.5),bg=1:n)
```




## 5.6. Función fourfoldplot

Esta función realiza una representación de tipo *fourfold* de k tablas de contingencia 2x2. La sintaxis es

```
fourfoldplot(x, color = c("#99CCFF", "#6699CC"), conf.level = 0.95, std =
("margins", "ind.max", "all.max"), margin = c(1, 2), space = 0.2, main = NULL,
mfrow = NULL, mfcol = NULL)
```

## 5.7. Función hist

Esta función genera y devuelve un histograma de los datos suministrados y además lo dibuja si se indica el parámetro **plot=TRUE**. La sintaxis es

```
hist(x, breaks = "Sturges", freq = NULL, probability = !freq, include.lowest = TRU
right = TRUE, density = NULL, angle = 45, col = NULL, border = NULL,
main = paste("Histogram of" , xname), xlim = range(breaks), ylim = NULL,
```

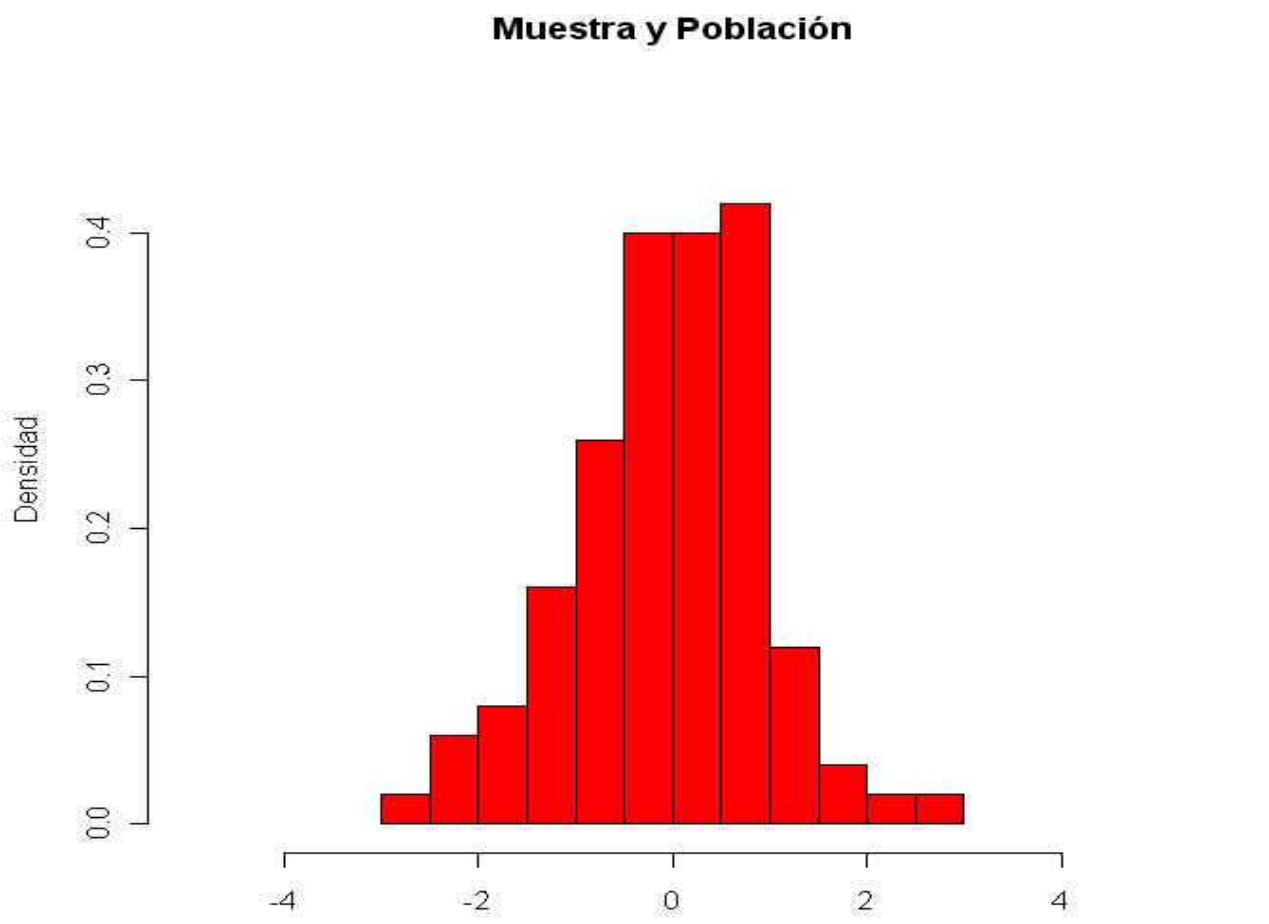
**xlab = xname, ylab, axes = TRUE, plot = TRUE, labels = FALSE,**
**nclass = NULL, ...)**

- **x** es el vector que contiene los valores para los que se realizará el histograma.
- **breaks** son los puntos de corte que definen los intervalos y pueden venir dados directamente, o por el número de intervalos, o por una cadena de caracteres que indican uno de los varios algoritmos existentes para construir los intervalos, o por una función que calcule el número de intervalos.
- **freq** es una variable de tipo lógico que indica si se representan frecuencias absolutas (TRUE) o relativas (FALSE).
- **include.lowest** es una variable de tipo lógico que indica en que intervalo se incluyen los puntos que coincidan con el extremo del intervalos.
- **right** es una variable de tipo lógico que indica si los intervalos son cerrados por la derecha.
- **density** es el número de líneas por pulgada para hacer sombreados.
- **angle** es la pendiente en grados para las líneas de sombreado.
- **col** es el color con que se rellenan las barras.
- **border** es el color con que se dibujan las barras.

Las siguientes órdenes generan una muestra pseudoaleatoria de una normal tipificada, epresentan el histograma de los valores obtenidos y superponen el gráfico de la normal tipificada, como puede observarse en la figura \ref{fig:HistogramaNormal}.

```
hist(rnorm(100),col="red",freq=F,xlim=c(-5,5), ylim=c(0,dnorm(0)*1.2),xlab='
ylab="Densidad",main="Muestra y Población")

curve(dnorm(x),-5,5,add=T,col="blue")
```




## 5.8. Función polygon

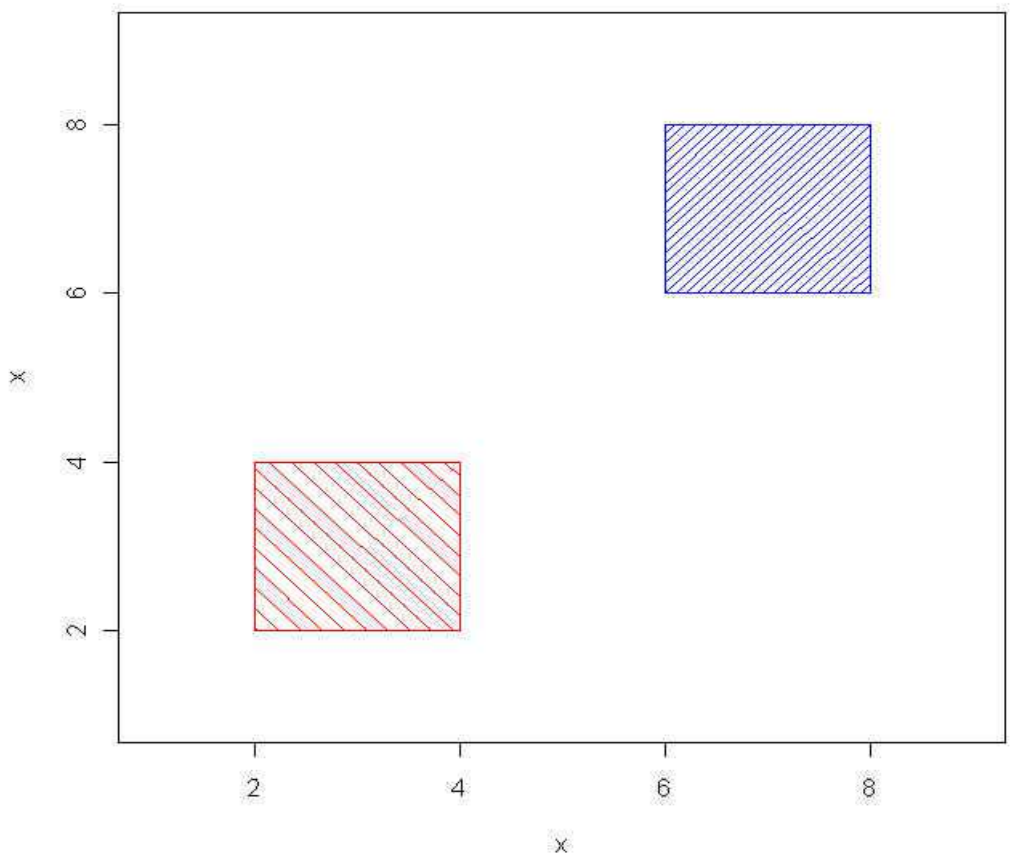
Esta función dibuja polígonos definidos por sus vértices, sobre un gráfico ya existente. La sintaxis de la orden es

**polygon(x, y = NULL, density = NULL, angle = 45, border = NULL, col = NA, lty**
**par("lty"), ...)**

- **x** e **y** son vectores que contienen las coordenadas de los vértices del polígono. El polígono se cierra uniendo el último punto con el primero. En las coordenadas puede haber valores NA en cuyo caso se realizarán varios polígonos.
- **density** es el número de líneas por pulgada para realizar el sombreado.
- **angle** es la pendiente en grados de las líneas de sombreado.
- **border** es el color con que se dibuja el borde del polígono. Si es \textbf{NA} no se dibujan.
- **col** es el color del relleno del polígono.
- **lty** es el tipo de línea que se usa.

Por ejemplo, las órdenes siguientes producen el gráfico de la figura.

```
x=c(1,9)
plot(x, x, type="n")
polygon(c(2,4,4,2,NA,6,8,8,6), c(2,2,4,4,NA,6,6,8,8),
density=c(10, 20), angle=c(-45, 45),col=c("red","blue"))
```




## 5.9. Función curve

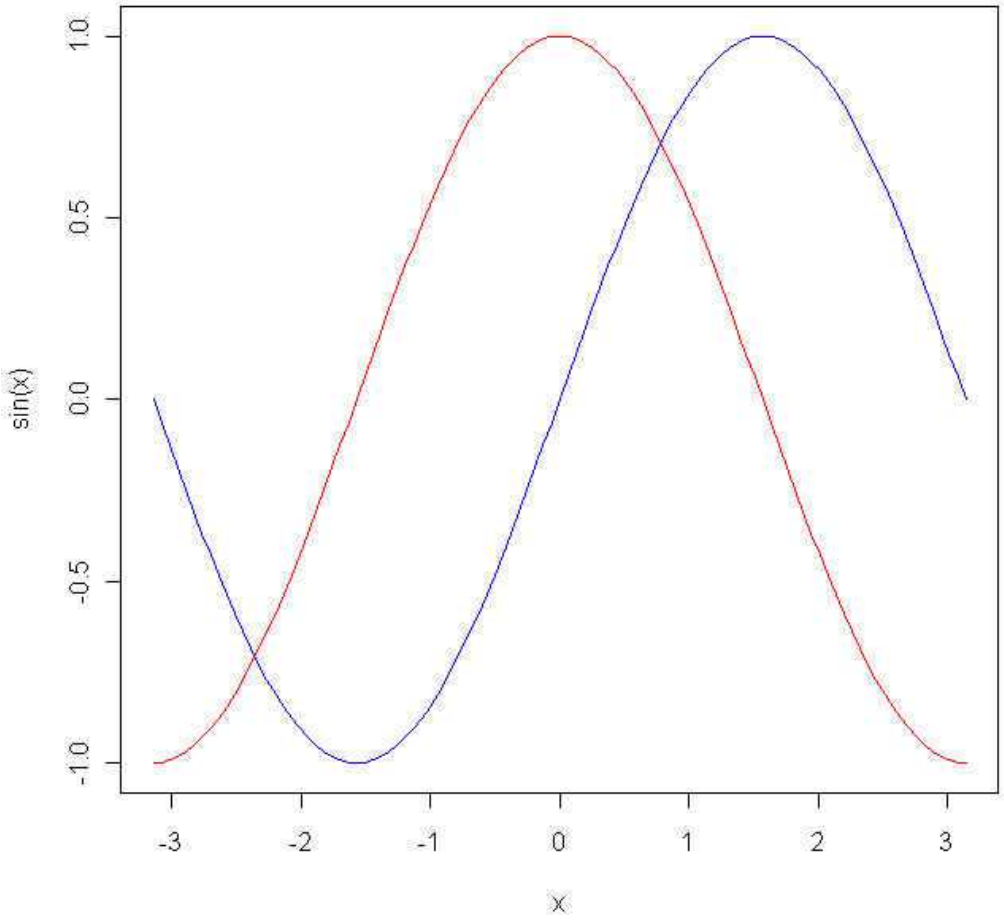
Esta función dibuja la curva correspondiente a una función dada en un intervalo concreto. Sus argumentos son:

- **expr** es una expresión que corresponde a una función de $x$ o el nombre de una función.
- **from**, **to** determinan el intervalo en que se realizará la gráfica.
- **n** = 101 es el número de puntos que se dibujará.
- **add** = FALSE indica si el gráfico se añade o no al gráfico actual.
- **type** = "l" indica el tipo de gráfico, predeterminadamente es líneas.
- **xlim** = NULL es un vector numérico de longitud 2. Si se especifica indica los límites del gráfico.
- **...** son argumentos adicionales que se pasan

También pueden pasarse etiquetas de los ejes y parámetros gráficos.

Así, por ejemplo, las órdenes siguientes producen el gráfico de la figura.

```
curve(sin(x),-pi,pi,col="blue")
curve(cos(x),-pi,pi,add=TRUE, col="red")
```




## 5.10. Funciones lines y points

Con estas funciones genéricas, puede añadir puntos o líneas al gráfico actual. Recuerde que debe haber realizado un gráfico, el cual define las medidas, aunque puede hacerlo con **type=n** si no desea representar ningún punto. La forma de uso es

```
lines(x, y, type="l")
points(x, y, type="p")
```

donde **x** e **y** son las coordenadas de unos puntos. Las coordenadas pueden expresarse mediante dos vectores, una matriz de dos columnas, una lista con dos componentes llamados **x** e **y**, un vector complejo, o una serie temporal univariante.

Si alguno de los puntos a representar es **NA**, no se dibuja. Si está dibujando líneas, se producirá una ruptura.

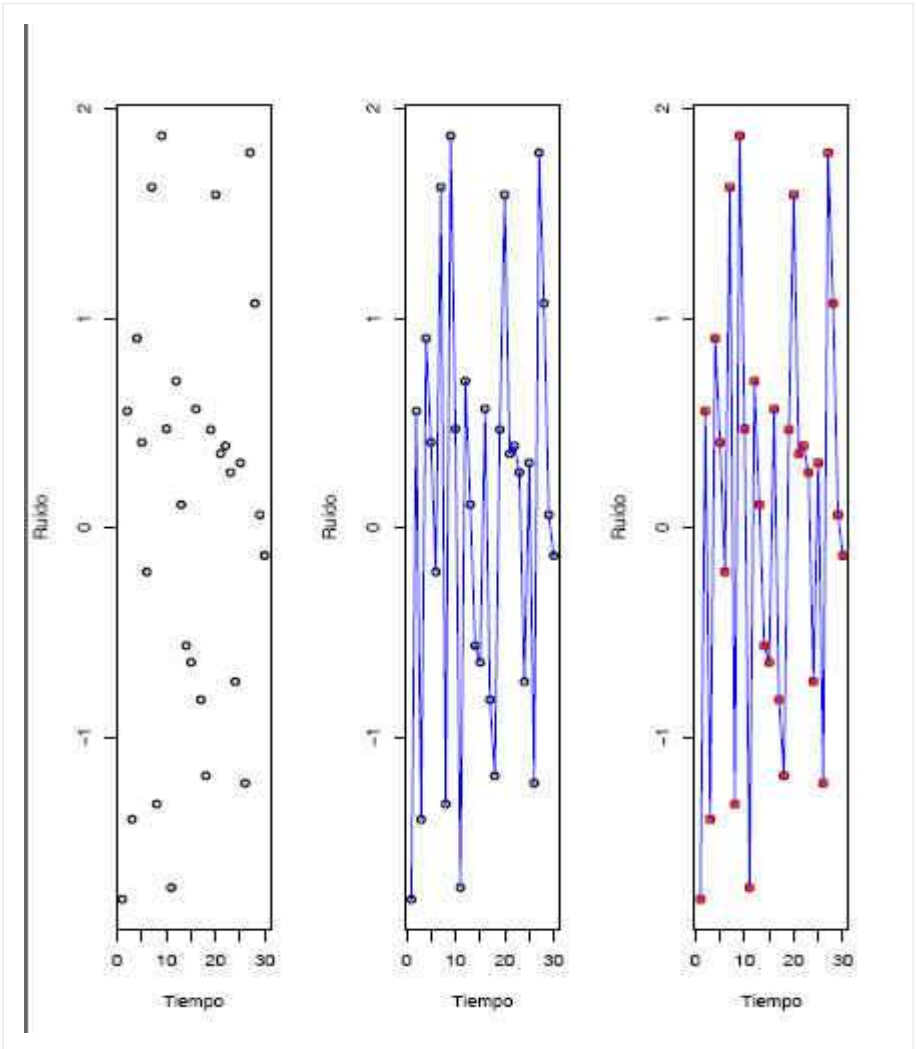
También puede pasar parámetros gráficos de la función **par**. Por ejemplo, con el parámetro pch, puede seleccionar símbolos especiales para representar cada punto.

## Ejemplos

A continuación generamos 30 valores pseudoaleatorios de una normal tipificada, los representamos y añadimos líneas y puntos.

```
x=rnorm(30)
opar=par(mfrow=c(1,3))
plot(x,xlab="Tiempo",ylab="Ruido")
plot(x,xlab="Tiempo",ylab="Ruido")
lines(x,col="blue")
plot(x,xlab="Tiempo",ylab="Ruido")
lines(x,col="blue")
points(x,col="red",pch=4)
par(opar)
```

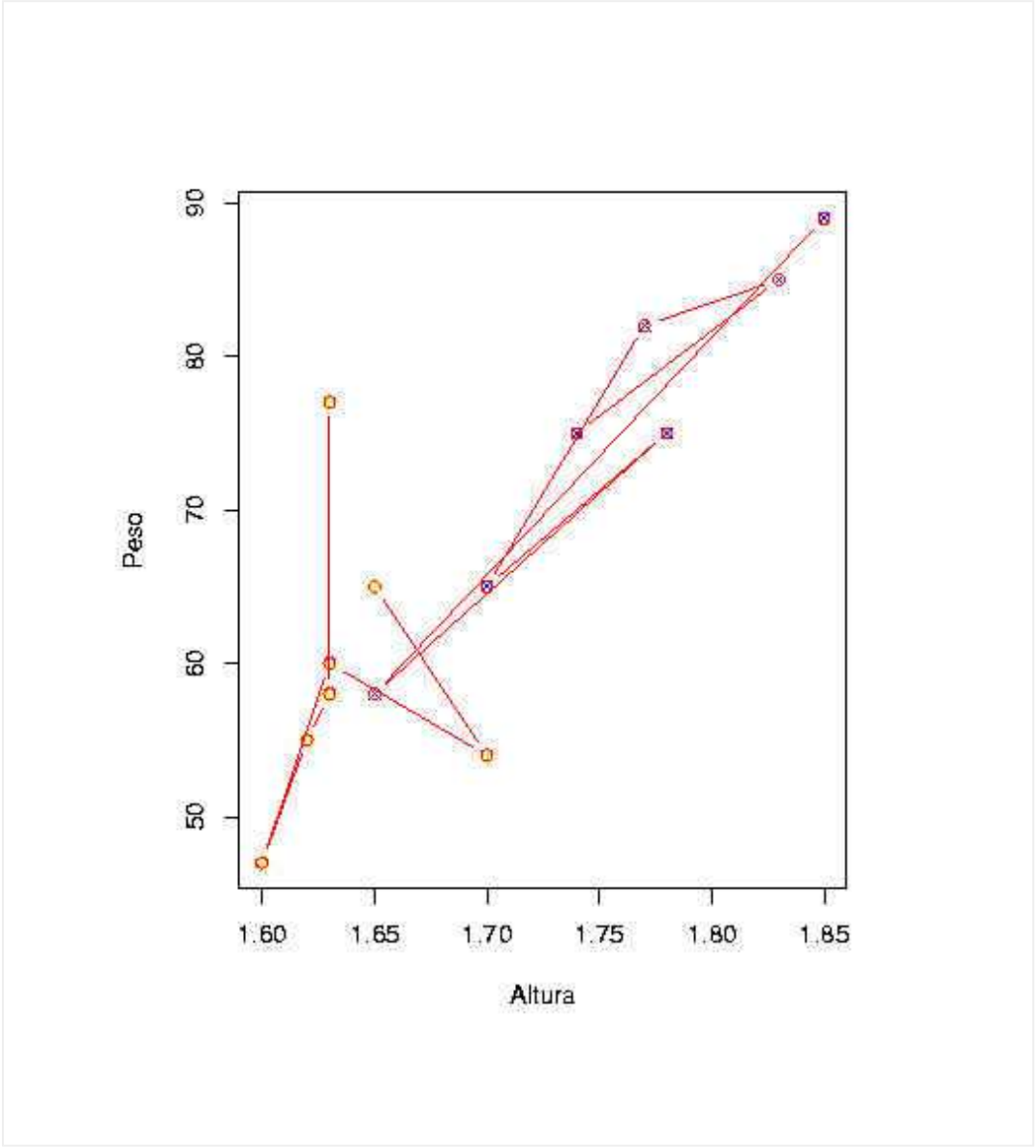
El resultado puede observarse en la siguiente figura.




En este segundo ejemplo, utilizando los datos almacenados en el objeto h.datos2,

seleccionamos en primer lugar las mujeres. A continuación representamos los valores de altura frente a peso, marcamos de modo distinto los hombres y mujeres y unimos con tramos rectos los elementos de cada grupo. El resultado se muestra en la figura \ref {hombreymujer}.

```
> attach(h.datos2)
> Mujer<-Sexo=="M"
> plot(Altura,Peso)
> points(Altura[Mujer], Peso[Mujer],
+ type="p",pch=3,col="yellow")
> points(Altura[!Mujer],Peso[!Mujer],
+ type="p",pch=4,col="blue")
> points(c(Altura[Mujer],NA,Altura[!Mujer]),
+ c(Peso[Mujer], NA,Peso[!Mujer]),type="b",col="red")
> detach(h.datos2)
```




## 5.11. Función par

Mediante la función **par** se controlan los parámetros gráficos que afectan al resultado de un gráfico. Todos sus argumentos son optativos y, por tanto, deben darse mediante el nombre. Si los argumentos de esta función se especifican en el interior de una de las funciones gráficas de nivel alto que hemos estudiado, sus efectos son temporales, en tanto que utilizados en la función **par**, son permanentes. La función **par** devuelve los valores actuales de sus parámetros. Por tanto, estos pueden almacenarse para ser recuperados posteriormente.

Puede comprobar como las órdenes siguientes producen dos gráficos distintos, representados en las figuras \ref{pchoriginal} y \ref{pch3}.

```
> attach(h.datos2)
> x11()
> plot(Edad,Altura)
> x11()
> par.antiguo <- par(pch=3)
> # Almacena los valores originales
> # y selecciona un nuevo símbolo
> # para representar los puntos
> plot(Edad,Altura)
> par(par.antiguo)
> # Restablece los valores originales
> detach(h.datos2)
```

\begin{figure}[thpb]
\begin{center}
\caption{pch original}
\label{pchoriginal}
\includegraphics[width=5cm]{pchoriginal.eps}
\end{center}
\end{figure}

\begin{figure}[hptb]
\begin{center}
\caption{pch 3}
\label{pch3}
\includegraphics[width=5cm]{pch3.eps}
\end{center}
\end{figure}

Existen muchos argumentos, que puede consultar utilizando **?par**, pero los más comunes son los siguientes. Puede conocer los valores predeterminados con la orden **par()**.

- **no.readonly** Existen dos tipos de parámetros, unos modificables y otros no. Si **no.readonly** es el único argumento y vale FALSE, se devuelven todos los parámetros, si es TRUE, sólo se devuelven los modificables.
- **ask** Si debe preguntar antes de borrar un gráfico. Los valores son TRUE y FALSE.
- **bg** Color de fondo del gráfico. Vea el parámetro **col**.
- **col** Indica el color. Puede hacerse mediante el nombre del color. La función

**colors** devuelve los nombres de colores disponibles. También puede definirse mediante una cadena de caracteres de la forma ''#RRGGBB'' donde cada pareja, `RR', `GG', `BB', son dos dígitos hexadecimales que varían de `00' a `FF', definiendo un color posible.

- **fg** Color de la tinta del gráfico.
- **font** Fuente del texto. Si es posible, el 1 corresponde a letra normal, el 2 a negrita, el 3 a cursiva, y el 4 a negrita cursiva.
- **lwd** Anchura de las líneas.
- **lty** Es el tipo de línea. Los tipos de línea se pueden indicar, en su forma elemental, mediante un entero (0=blanco, 1=continuo, 2=guiones, 3=puntos, 4=punto y guión, 5=guión largo, 6=dos guiones) o mediante una cadena de caracteres, respectivamente, de las siguientes: ''blank'', ''solid'', ''dashed'', ''dotted'', ''dotdash'', ''longdash'', o ''twodash''. El tipo 0 corresponde a no dibujar la línea. Además es posible especificar una cadena de hasta ocho caracteres, tomados de entre ''0123456789ABCDEF'', para indicar la longitud, en hexadecimal, de los segmentos que se dibujan y saltan, alternativamente, para realizar la línea. Por ejemplo, ''44'', corresponde a guiones, y ''13'', a puntos.
- **new** Valor lógico que indica si la siguiente orden gráfica de alto nivel debe comenzar un nuevo gráfico o añadirse al existente.
- **pch** Símbolo con que se dibuja. Puede ser un entero (que indica el número del carácter especial a utilizar) o un carácter (que se utiliza literalmente).
- **ps** Es un entero que indica el tamaño en puntos de los símbolos.
- **type** Tipo de dibujo para representar los puntos. Los valores son:
  - **p** puntos
  - **l** líneas
  - **b** puntos y líneas, sin superponerse
  - **o** puntos y líneas, superponiéndose
  - **n** nada
  - **s, S** líneas en escalera, comenzando hacia la derecha o hacia arriba.
  - **h** líneas verticales desde el eje X
- **xlab** Cadena de caracteres para etiquetar el eje X.
- **xlog** Si es TRUE, se utiliza una escala logarítmica para el eje X, en caso contrario, se utiliza una escala lineal.
- **ylab** Cadena de caracteres para etiquetar el eje Y.
- **ylog** Si es TRUE, se utiliza una escala logarítmica para el eje Y, en caso contrario, se utiliza una escala lineal.

## Ejemplos

Si escribe, par(ask=T) , desde ese momento, cada vez que se genera un gráfico, aparece un cuadro de diálogo antes de borrar el anterior.

Si escribe

```
> win.graph()
```

```
> plot(1:10,(1:10)^2)
> win.graph()
> plot(1:10,(1:10)^2,xlab="Eje X",ylab="Y=X^2")
```

el gráfico se genera, en el segundo caso, con nuevas etiquetas en los ejes.

## 5.12. Desplazamiento entre dispositivos gráficos

Cuando existen varios dispositivos gráficos, es posible saber cual es el activo, cerrar uno concreto (con lo que se termina de lanzar el gráfico correspondiente, si está pendiente) o pasar de uno a otro. Para ello se utilizan las siguientes funciones:

**dev.cur** devuelve el número y nombre del dispositivo gráfico actual.

**dev.list** devuelve el número y nombre de todos los dispositivos gráficos activos.

**dev.next** devuelve el número y nombre del siguiente dispositivo gráfico de la lista activos.

**dev.prev** devuelve el número y nombre del anterior dispositivo gráfico de la lista de activ

**dev.set** hace que sea activo el dispositivo indicado y, además, devuelve el número nombre del mismo. Si no se le da valor, cambia al siguiente.

**dev.off** cierra el dispositivo actual, pasa al siguiente, y devuelve su número.

Siempre hay un dispositivo gráfico abierto con el número 1, *null device*, aunque no sirve para realizar ningún gráfico efectivo.

### Ejemplo

Considere las siguientes órdenes:

```
> graphics.off()
> dev.cur()
null device
1
> x11()
> x11()
> dev.cur()
windows
3
> dev.next()
windows
2
> dev.set(dev.next())
windows
2
> dev.prev()
windows
3
```

```
> dev.set(dev.prev())
windows
3
> plot(1:10)
```

En primer lugar, cierra cualquier posible dispositivo gráfico abierto. Por tanto, el dispositivo actual será el 1; a continuación abre dos ventanas, por tanto, el dispositivo actual será el 3. El siguiente, será el 2. A continuación hace que sea este el dispositivo actual. El anterior será el 3, y si lo hace actual, la orden última realizará un gráfico sobre el mismo.

## 5.13. Impresión de gráficos

Para imprimir un gráfico, ya generado, puede utilizar la función

```
dev.print(device=postscript, ...)
```

que copia el gráfico del dispositivo actual en el dispositivo creado por la función especificada en **device** y a continuación lo cierra.

Para generar un gráfico directamente sobre el administrador de impresión, puede utilizar la función **win.print**.

## 5.14. Identificación interactiva

Cuando se ha realizado un gráfico, es posible identificar interactivamente puntos con la función **identify**. Al marcar un punto con el botón primario se escribe en pantalla el valor del identificativo. El botón secundario sale de este modo y la función devuelve los puntos marcados.

```
> plot(h.datos2[,3],h.datos2[,2])
> identify(h.datos2[,3],h.datos2[,2],h.datos2[,4])
[1] 11 13 9 12 3 1 10 7 14 8
```

## 5.15. Función locator

La función **locator** devuelve las coordenadas de una posición de un gráfico. Su sintaxis es

```
locator(n = 512, type = "n")
```

El argumento **n** indica el máximo número de puntos que leer, aunque puede finalizar antes pulsando el botón secundario. El argumento **type** permite dibujar utilizando las posiciones seleccionadas, con el mismo significado que en las otras funciones gráficos, por tanto, su valor predeterminado es no dibujar. La función devuelve las coordenadas de las posiciones seleccionadas, en una lista con dos componentes, **x** e **y**.

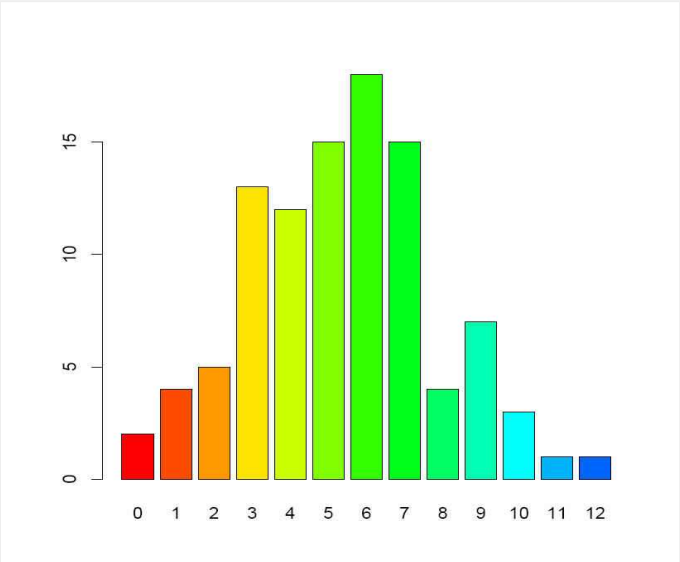
Esta función suele utilizarse sin argumentos. Es particularmente útil para seleccionar interactivamente posiciones para elementos gráficos, tales como etiquetas, en que es difícil conocer previamente dónde deben colocarse. Por ejemplo, para colocar un texto informativo en una posición arbitraria de un gráfico, puede utilizar

```
> text(locator(1), "Aquí", adj=0)
```

## 5.16. Función barplot

Esta función genera gráficos de barras en muy diversas formas. Utilice la función **example (barplot)** para hacerse una idea de sus posibilidades. De allí hemos tomado el siguiente ejemplo, que produce el gráfico de la figura. Se generan 100 números pseudoaleatorios de una distribución de Poisson de parámetro lambda=5 se realiza un conteo de frecuencias y se representa con barras de diferentes colores.

```
tN <- table(Ni <- rpois(100, lambda=5))
r <- barplot(tN, col=rainbow(20))
```

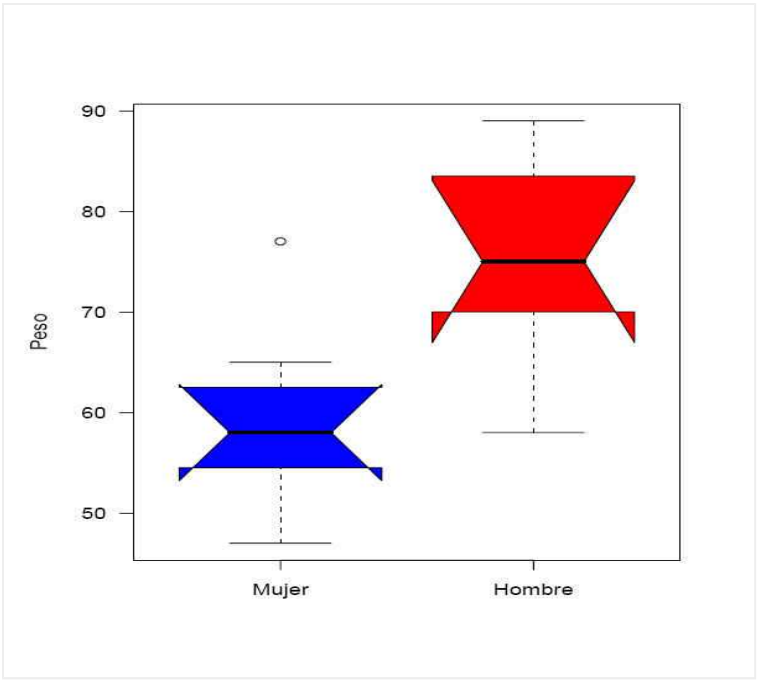



## 5.17. Función boxplot

Esta función realiza gráficos de caja con bigotes (*box-and-whisker*). Utilice la función **example(boxplot)** para hacerse una idea de sus posibilidades.

El siguiente ejemplo produce el gráfico de la figura.

```
attach(hoja)
boxplot(Peso[Sexo=="M"],Peso[Sexo=="H"],
notch=T,names=c("Mujer","Hombre"),
ylab="Peso",col=c("blue","red"))
```



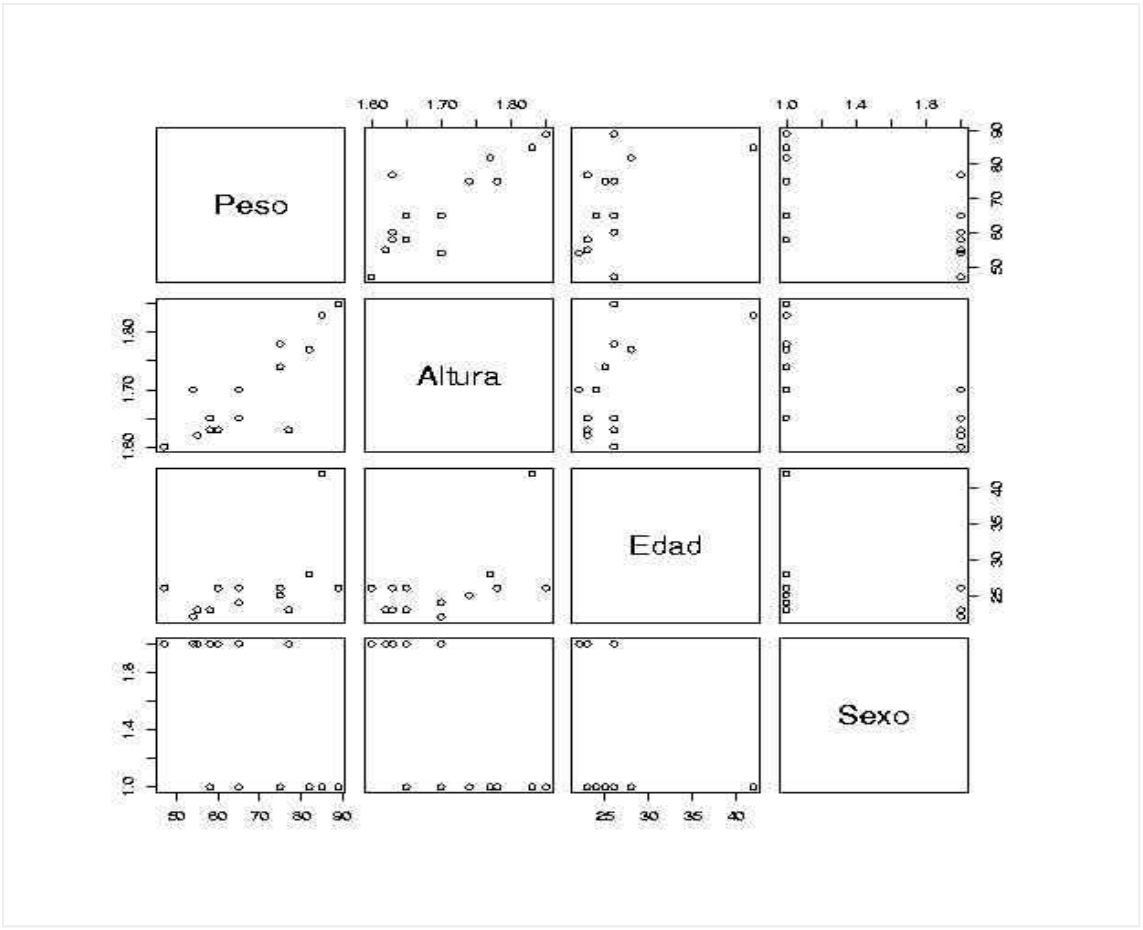

## 5.18. Función pairs

Esta función, genérica, crea una figura que contiene todos los diagramas de dispersión de cada variable frente a las restantes. La función está definida para las clases **matrix**, **data.frame** y **formula**. La sintaxis es

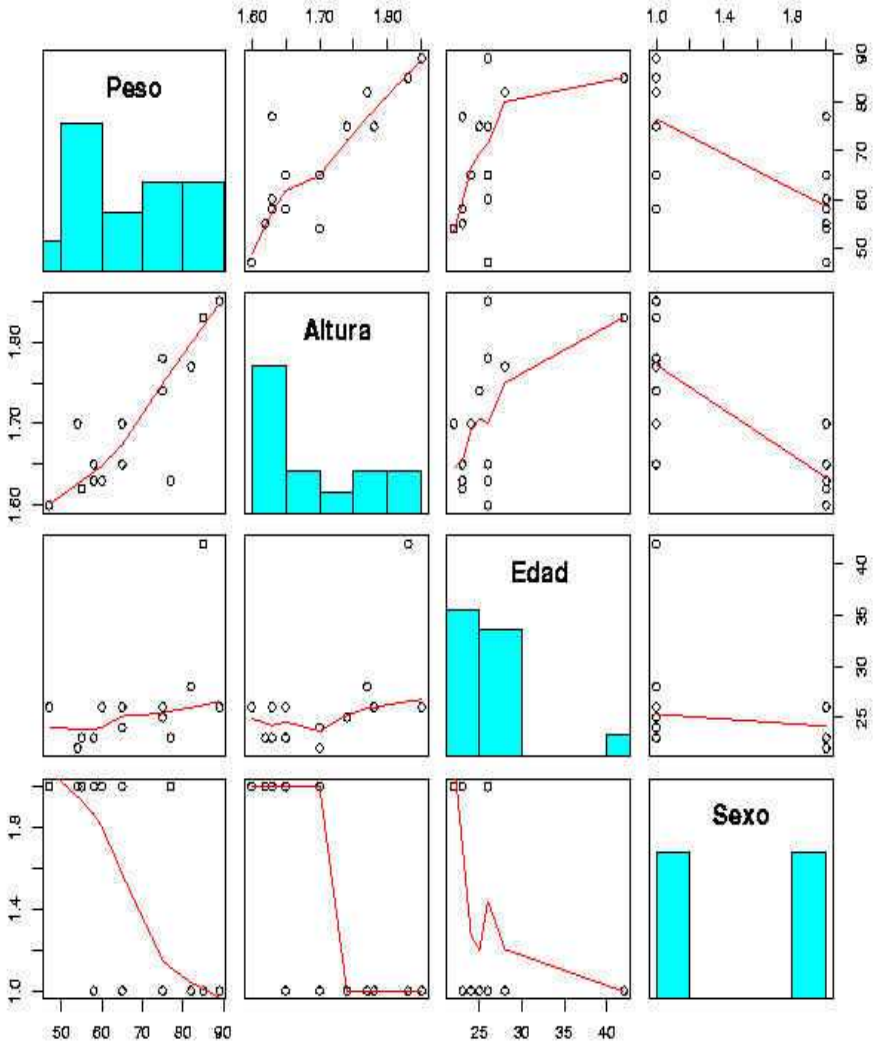
**pairs(x, (x, labels = colnames(x), panel = points, ...,)**

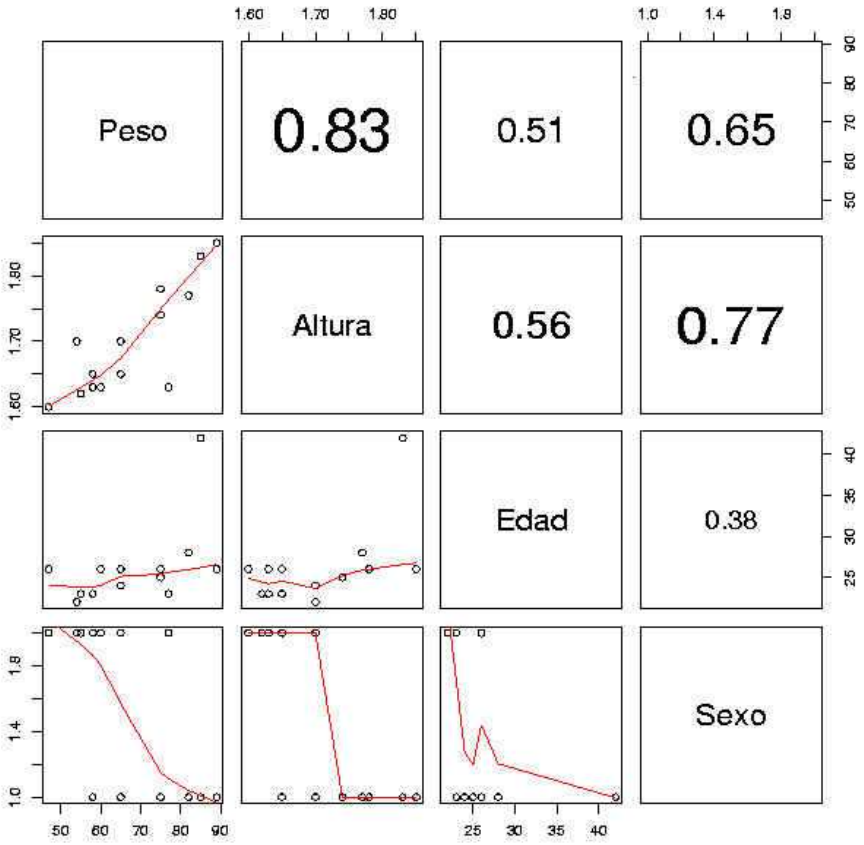
donde **x** es un objeto, **labels** son caracteres que etiquetan a las variables, y **panel** es `function(x,y,...)', que es la función que se utiliza para representar cada panel del gráfico. También pueden darse parámetros gráficos generales. Es posible referirse a la diagonal, el triángulo inferior y el superior del conjunto de paneles mediante **diag.panel**, **lower.panel** y **upper.panel** respectivamente.

Un ejemplo de utilización es **pairs(h.datos2)**, que produce:




Puesto que puede definir la función de representación, puede construir gráficos con modificaciones a partir de esta función, como se indica en la ayuda de la función. De este modo, las órdenes siguientes permiten representaciones como las de las figuras.


Peso
Altura
Edad
Sexo
1.60
1.70
1.80
1.0
1.4
1.8
50
60
70
80
90
25
30
35
40



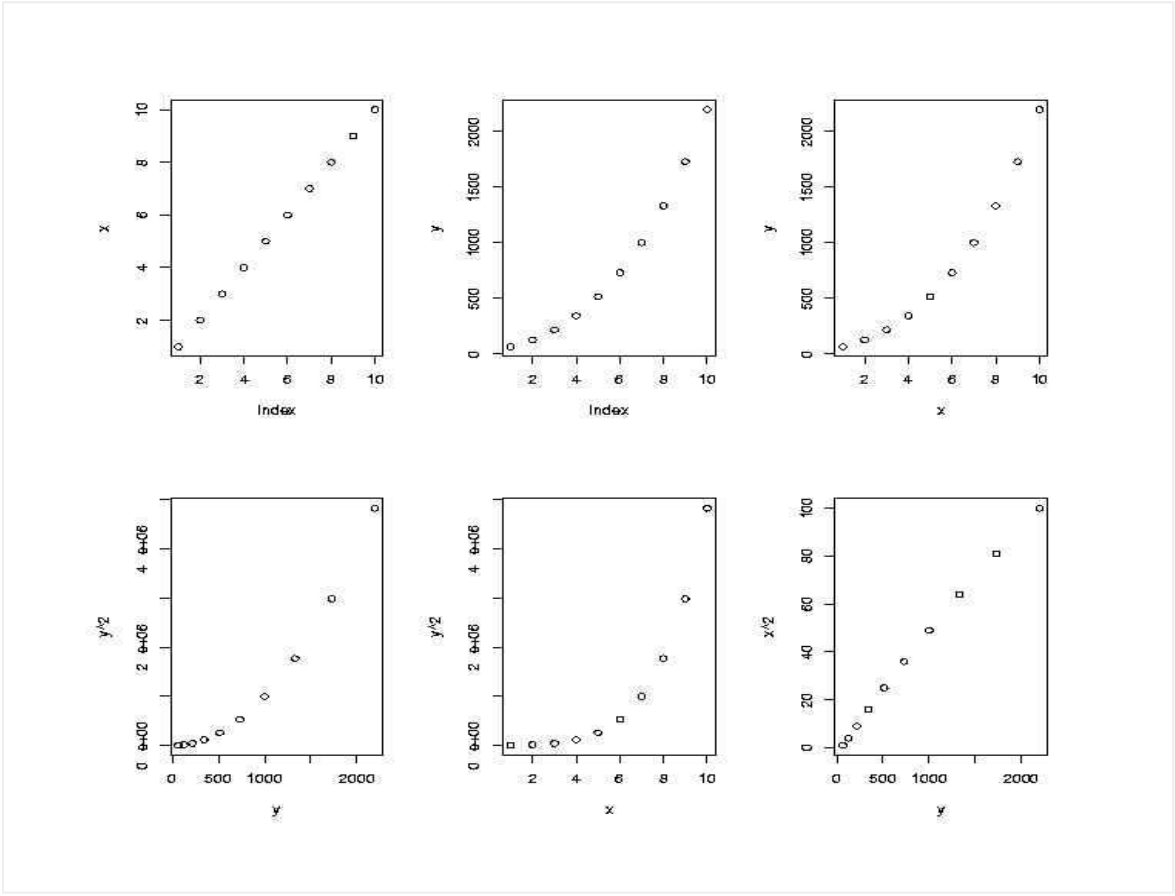

```
> ## incluye histogramas en la diagonal
> panel.hist <- function(x, ...)
> {
> usr <- par("usr"); on.exit(par(usr))
> par(usr = c(usr[1:2], 0, 1.5) )
> h <- hist(x, plot = FALSE)
> breaks <- h$breaks; nB <- length(breaks)
> y <- h$counts; y <- y/max(y)
> rect(breaks[-nB], 0, breaks[-1], y, col="cyan", ...)
> }
> pairs(h.datos2[1:4], panel=panel.smooth,
+ diag.panel=panel.hist, cex.labels=1.5,
+ font.labels=2)
> ## escribe correlaciones absolutas en los
> ## paneles superiores, de tamaño proporcional
> ## a los valores.
> panel.cor <- function(x, y, digits=2, prefix="", cex.cor)
> {
> usr <- par("usr"); on.exit(par(usr))
> par(usr = c(0, 1, 0, 1))
> r <- abs(cor(x, y))
> txt <- format(c(r, 0.123456789), digits=digits)[1]
> txt <- paste(prefix, txt, sep="")
```

```
> if(missing(cex.cor)) cex <- 0.8/strwidth(txt)
> text(0.5, 0.5, txt, cex = cex * r)
> }
> pairs(h.datos2, lower.panel=panel.smooth,
+ upper.panel=panel.cor)
```

## 5.19. Gráficos múltiples

Entre los argumentos de la función **par** hay algunos que permiten presentar gráficos múltiples. Así, el argumento **mfcol=c(m,n)**, divide el dispositivo gráfico en m por n partes iguales, que se rellenan por columnas. Significado análogo tiene **mfrow=c(m,n)**, aunque en este caso se rellenan por filas.

Si escribe las órdenes siguientes, el dispositivo gráfico actual se dividirá en seis partes y se realizarán seis gráficos, en el orden que se indica, por filas.




```
> par(mfrow=c(2,3))
> x<-(1:10)
> y<-(4:13)^3
> plot(x)
> plot(y)
> plot(x,y)
```

```
> plot(y,y^2)
> plot(x,y^2)
> plot(y,x^2)
```

Si desea rellenarlo por columnas tendría que sustituir la primera orden del ejemplo anterior por

```
>par(mfcol=c(2,3))
```

## 5.20. Funciones de uso de pantalla

Una forma versátil de dividir un dispositivo gráfico en partes es utilizar la función **split.screen,** que divide el dispositivo gráfico en partes, que pueden, hasta cierto punto, ser tratadas como dispositivos gráficos distintos. Estas partes pueden a su vez dividirse, lo que permite crear gráficos complejos. Las funciones **screen**, **erase.screen** y **close.screen**, permiten, respectivamente, seleccionar la pantalla actual, borrarla o cerrarla. La sintaxis de estas funciones es

```
split.screen(figs, screen =, erase = TRUE)
screen(n =, new = TRUE)
erase.screen(n =)
close.screen(n =, all = TRUE)
```

donde los parámetros se interpretan así:

- **figs** es, bien un vector de la forma **c(filas,columnas)** que indica la división inicial de la pantalla, bien una matriz Nx4, donde N es el número de pantallas y cada fila especifica la posición de la pantalla correspondiente dentro del dispositivo gráfico en términos de coordenadas relativas, considerando que el dispositivo total se representa por el vector c(0,1,0,1). Estas coordenadas se dan así: las dos primeras componentes representan las dos abscisas (izquierda y derecha) y las dos segundas las dos ordenadas (inferior y superior).
- **n** es el número de pantalla, utilizado tanto por **screen**, **erase.screen**, como **close.screen**. El valor predeterminado es el correspondiente a la pantalla activa.
- **screen** es el número de la pantalla que se va a dividir. El valor predeterminado es la pantalla activa, que inicialmente es el dispositivo completo, numerado como 0.
- **erase** valor lógico que indica si debe borrarse la pantalla en que se va a dibujar.
- **new** valor lógico que indica si debe borrarse la pantalla.
- **all** valor lógico que indica si deben cerrarse todas las pantallas.

## 5.21. Diagrama de sectores

La función **pie** permite crear un diagrama de sectores. Su sintaxis es

```
pie(x, labels=names(x), shadow=FALSE, edges=200, radius=0.8, fill=NU
main=NULL, ...)
```

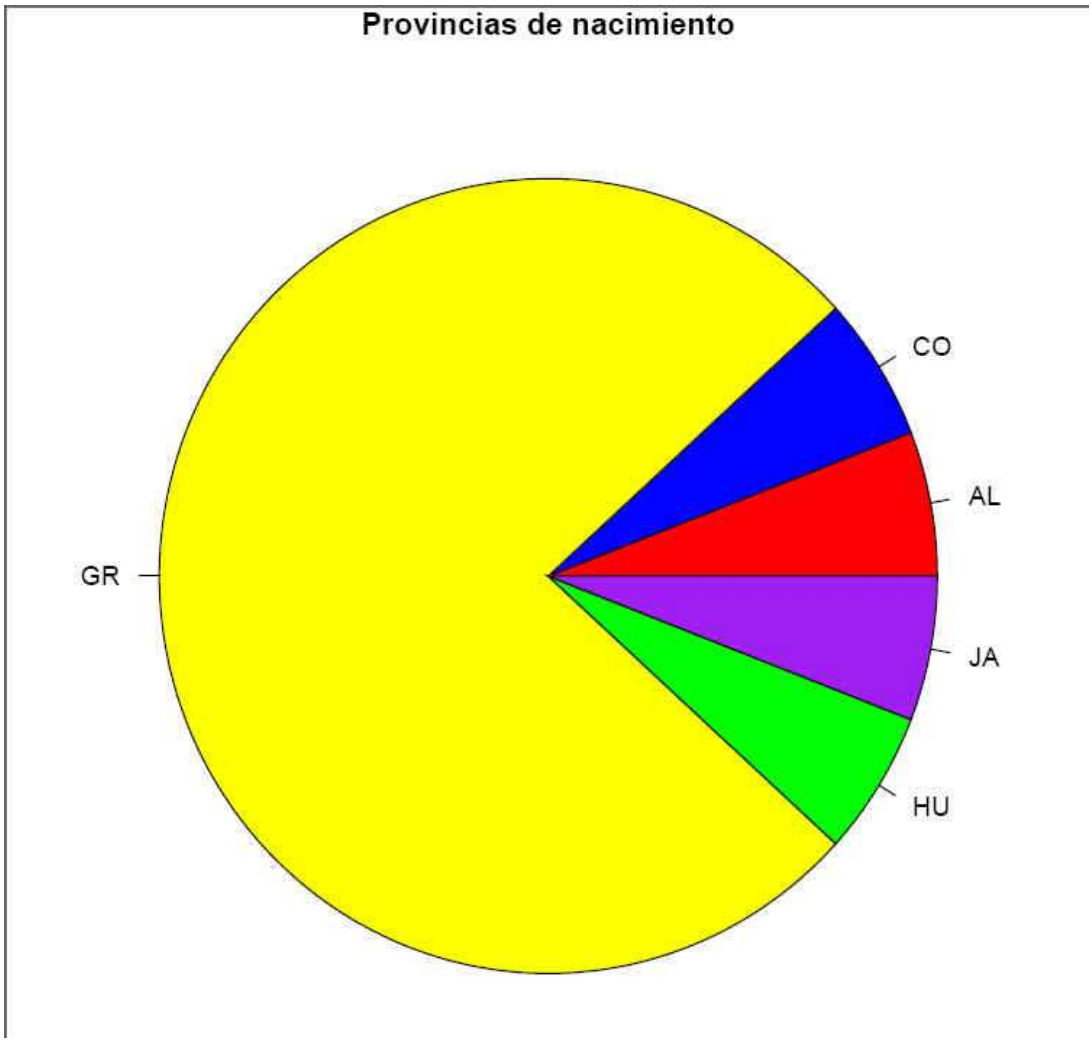
siendo los parámetros los siguientes:

- **x** es un vector de valores proporcionales al tamaño de cada sector.
- **labels** es un vector de etiquetas de los sectores.
- **shadow** es un vector lógico que indica qué efecto de sombreado debe aplicarse a un gráfico con colores.
- **edges** número de vértices de un polígono que aproxima al círculo.
- **radius** radio del círculo respecto del tamaño del gráfico.
- **col** vector de colores para cada sector.
- **main** título del gráfico.
- **...** parámetros gráficos adicionales de la función **par**.

## Ejemplos

Considere el siguiente vector de caracteres que representa las provincias de nacimiento de las personas de h.datos2. Puede realizar un diagrama de sectores mediante la función **pie** aplicada al conteo de frecuencias y obtendrá la figura inferior.

```
> graphics.off()
> x11()
> opar<- par(mai=c(0,0,0.2,0))
> Provincia<-c("GR","CO","GR","GR","HU","GR","GR",
+ "GR","AL","GR","GR","GR","GR","GR","JA","GR","GR")
> Provincia
[1] "GR" "CO" "GR" "GR" "HU" "GR"
[7] "GR" "GR" "AL" "GR" "GR" "GR"
[13] "GR" "GR" "JA" "GR" "GR"
> table(Provincia)
Provincia
AL CO GR HU JA
1 1 13 1 1
> pie(table(Provincia),
col=c("red","blue","yellow","green","purple"))
> title("Provincias de nacimiento")
> par <- opar
```




## 5.22. Diagramas de estrella

La función **stars** realiza un diagrama de estrellas, una por individuo, con información de todas las variables. La sintaxis es

**stars(x, full = TRUE, scale = TRUE, radius = TRUE, labels = dimnames(x)[[1**
**locations = NULL, xlimit = NULL, ylimit = NULL,len = 1, colors = NULL,**
**key.loc = NULL, key.labels = NULL,**
**draw.segments = FALSE, draw.axes = FALSE, ...)**

cuyos argumentos más importantes tienen el siguiente significado:

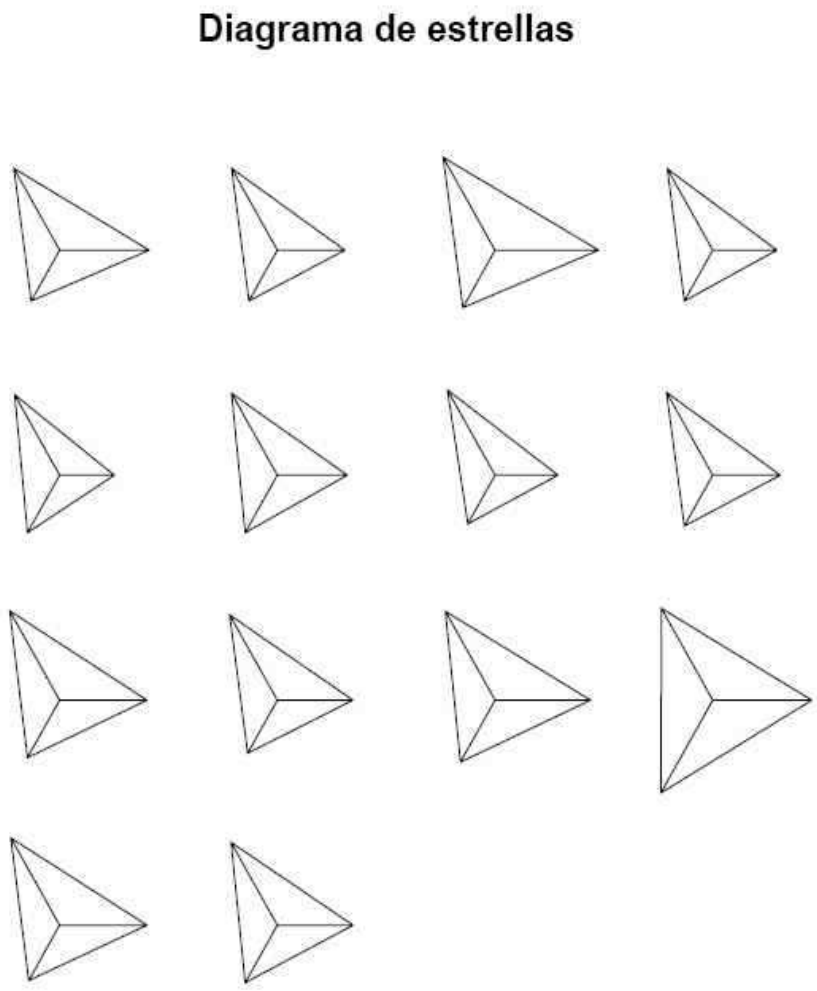
- **x** es una matriz de datos.
- **full** valor lógico que indica si los símbolos deben ocupar un círculo completo o sólo el semicírculo superior.
- **scale** valor lógico que indica si las columnas deben homogeneizarse al intervalo (0,1).
- **radius** valor lógico que indica si deben dibujarse los radios.
- **labels** etiquetas de los dibujos. El valor predeterminado es el nombre del individuo.

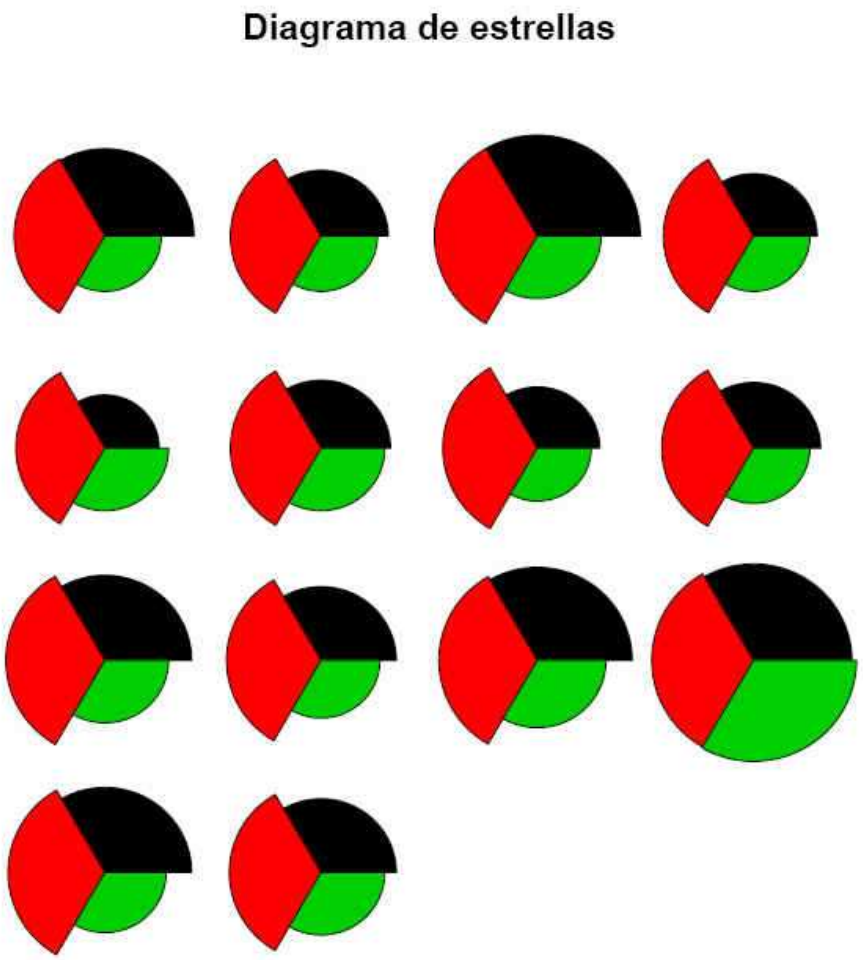
**...** parámetros gráficos adicionales de la función **par**.

## Ejemplos

Las siguientes órdenes crean dos diagramas de estrella, recogidos en los gráficos que se muestran a continuación

```
> attach(h.datos2)
> stars(matrix(c(Peso,Altura,Edad),ncol=3), main="Diagrama de estrellas")
> stars(matrix(c(Peso,Altura,Edad),ncol=3),
+ main="Diagrama de estrellas", draw.segments = TRUE)
```






A continuación se define la función **Tarta**, para eliminar la necesidad de realizar el conteo de frecuencias. Para ilustrar la realización de varios gráficos, se repite el gráfico cuatro veces.

```
> Tarta<-function(var = "")
+ {
+ if(var == "")
+ stop("Necesitas una variable")
+ x11()
+ opar<-par(mfrow = c(2, 2))
+ for(i in 1:(2 * 2))
+ piechart(table(var), col=c("red","blue","yellow","green","purple"))
+ par<-opar
+ }
> Tarta(Provincia)
```

Se puede modificar la función para que represente un número arbitrario de veces el gráfico, así como para comprobar que los parámetros son válidos.

```
> Tarta<-function(var = "",filas=2,columnas=2)
+ {
+ if(var == "")
+ stop("Necesitas una variable")
+ filas <-as.integer(filas)
+ columnas<-as.integer(columnas)
```

```
+ if(filas<1) stop("El numero de filas es < 1")
+ if(columnas<1) stop("El numero de columnas es < 1")
+ x11()
+ opar<-par(mfrow = c(filas, columnas))
+ for(i in 1:(filas * columnas))
+ piechart(table(var), col=c("red","blue","yellow","green","purple"))
+ par<-opar
+ }
> graphics.off()
> Tarta(Provincia,3)
> Tarta(Provincia,3,3)
> Tarta(Provincia,,1)
> Tarta(Provincia,1,1)
> Tarta(Provincia,-1,1)
Error in Tarta(Provincia, -1, 1) : El numero de filas es < 1
```

## 5.23. Representaciones tridimensionales

Es posible realizar gráficos tridimensionales mediante las funciones **persp**, **contour** e **image**. Las tres utilizan los siguientes parámetros:

- **x** vector de puntos del eje X, en orden ascendente, para los que se hará la representación.
- **y** vector de puntos del eje Y, en orden ascendente, para los que se hará la representación.
- **z** matriz de valores de la función a representar. Si este es el primer valor se calculan **x** e **y**.

La función **persp** crea un gráfico en perspectiva. Su sintaxis es

```
persp(x = seq(0, 1, len = nrow(z)), y = seq(0, 1, len = ncol(z)),
z, xlim = range(x), ylim = range(y), zlim = range(z, na.rm = TRUE),
xlab = NULL, ylab = NULL, zlab = NULL, theta = 0, phi = 15, r = sqrt(3), d = 1,
scale = TRUE, expand = 1, col = NULL, border = NULL,
ltheta = -135, lphi = 0, shade = NA, box = TRUE,
axes = TRUE, nticks = 5, ticktype = "simple", ...)
```

Los parámetros más utilizados son:

- **x, y** posiciones de la rejilla definida por puntos del eje X y del eje Y, en orden ascendente, para los que se ha medido el valor de 'z'. El valor predeterminado son valores equiespaciados entre 0 y 1. Si **x** es una lista, deberá tener dos componentes, x$x y x$y, que se utilizarán como valores de **x** e **y** respectivamente.
- **z** matriz de valores de la función a representar.
- **xlim, ylim, zlim** Extremos de x, y y z.
- **xlab, ylab, zlab** Etiquetas de los ejes.
- **theta, phi** Ángulos que definen la dirección de visión.
- **r** Distancia desde el lugar de punto de vista al centro del gráfico.

- **box** Valor lógico que indica si debe dibujarse la caja que delimita el dibujo. Su valor predeterminado es TRUE.
- **...** parámetros gráficos adicionales.

La función **contour** realiza un mapa de nivel. Su sintaxis es

**contour(x = seq(0, 1, len = nrow(z)), y = seq(0, 1, len = ncol(z)), z,**
**nlevels = 10, levels = pretty(zlim, nlevels), labels = NULL,**
**xlim = range(x, finite = TRUE), ylim = range(y, finite = TRUE),**
**zlim = range(z, finite = TRUE),**
**labcex = 0.6, drawlabels = TRUE, method = "flattest",**
**vfont = c("sans serif", "plain"),**
**col = par("fg"), lty = par("lty"), lwd = par("lwd"), add = FALSE, ...)**

Los parámetros más utilizados son:

- **x, y** posiciones de la rejilla definida por puntos del eje X y del eje Y, en orden ascendente, para los que se ha medido el valor de 'z'. El valor predeterminado son valores equiespaciados entre 0 y 1. Si x es una lista, deberá tener dos componentes, x$x y x$y, que se utilizarán como valores de **x** e **y** respectivamente.
- **z** matriz de valores de la función a representar.
- **nlevels** número de niveles de contorno, sólo en el caso de que no se especifique levels.
- **levels** vector numérico de los niveles en que se dibujarán líneas de contorno.
- **labels** vector de etiquetas de las líneas de contorno. Si su valor es `NULL' se utilizan los propios niveles como etiquetas.
- **drawlabels** Valor lógico que indica si deben etiquetarse las líneas de contorno.
- **xlim, ylim, zlim** Extremos de x, y y z.
- **col** Color de las líneas.
- **lty** Tipo de línea que se dibujará.
- **lwd** Anchura de las líneas.
- **add** Valor lógico que indica si debe añadirse al gráfico actual.
- **...** parámetros gráficos adicionales.

Por último, la función **image** dibuja una representación utilizando un código de color o una escala de grises. La sintaxis es

**image(x, y, z, zlim=range(z), add=F)**

- **zlim** vector que contiene el mínimo y el máximo de z.
- **add** valor lógico que indica si la imagen debe añadirse al gráfico actual.

También es posible añadir parámetros gráficos.

## Ejemplos

La función **Dibuja** realiza los tres tipos de representación. A continuación la utilizamos con

dos funciones diferentes, una definida por su nombre y otra definida directamente.

```
> Seno<-function(x,y) sin(x*y)
> Dibuja <- function(x = NA, y = NA, f = Seno)
+ {
+ close.screen(all = T)
+ win.graph()
+ split.screen(c(1, 3))
+ screen(1)
+ image(x, y, outer(x, y, f))
+ screen(2)
+ contour(x, y, outer(x, y, f))
+ screen(3)
+ persp(x, y, outer(x, y, f))
+ }
> x<- -5:5
> Dibuja(x,x)
> Dibuja(x,x,function(x,y) sin(x)*cos(y))
```