TabulaR - Gráficos profesionales

Elvira Ferre Jaén

Marzo 2018

Índice

1. Todo lo que debes saber para empezar con gráficos en R

1.1. Cómo abrir un gráfico vacío

Podemos abrir un gráfico vacío con la función `plot()`:

```
plot.new()
```

Esta función define un nuevo gráfico sin ejes, etiquetas ni contorno. Esto significa que se abrirá una nueva ventana gráfica si no había ninguna abierta, o en el caso de que la ventana existiera está preparada para alojar un nuevo gráfico.

1.2. Cómo dibujar un gráfico vacío

```
plot(  5, 5,                  # coordenadas puntos
       type = "n",            # no dibujar los puntos
       axes = FALSE,          # no dibujar los ejes
       ann = FALSE,           # sin etiquetas
       xlim = c( 0, 10 ),     # establecemos los límites del gráfico
       ylim = c( 0, 10 ) )
```

Le hemos pasado al gráfico las coordenadas `(5,5)`, sin embargo no muestra nada, pues al poner el argumento `type = "n"` le decimos al R que no dibuje nada.(Ejercicio: prueba a poner el argumento como `"p"` o `"b"` y observa qué ocurre).

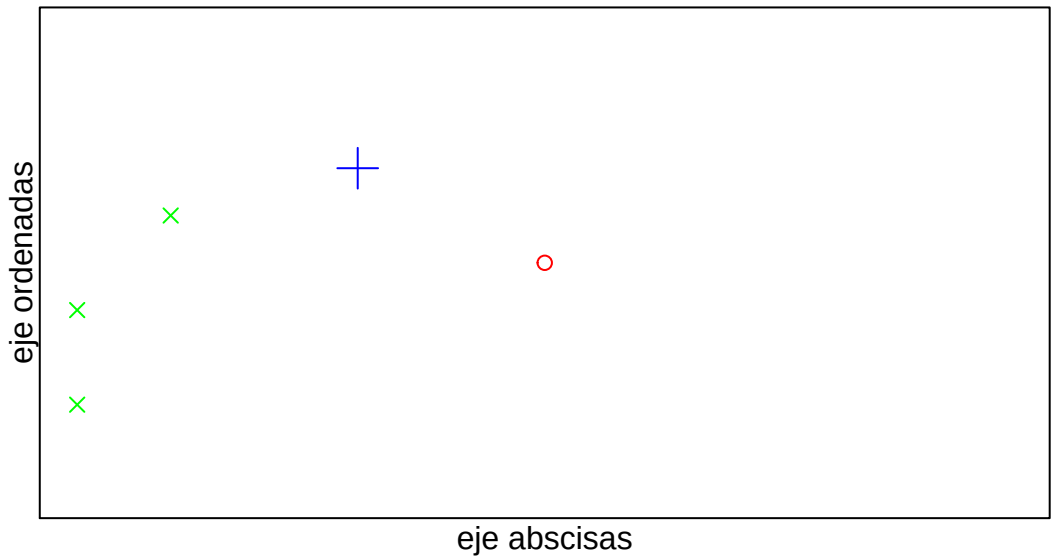
Si os fijais además no tenemos ninguna etiqueta en el gráfico, pero le hemos puesto límites de 0 a 10 en ambos ejes. Una vez tenemos el *device gráfico* lanzado con las dimensiones deseadas, rellenamos el gráfico con los ejes, etiquetas y el título:

```
mtext( "eje abscisas",
      side = 1 ) # Añadir texto a los ejes
mtext( "eje ordenadas", side = 2 )
title( "Un plot en R" ) # Añadir un titulo
```

Podemos dibujar además una caja alrededor de nuestro gráfico usando la función `box()` y añadir algunos puntos con la función `points()`

```
box()      # Dibujar la caja

# Punto rojo en (5,5)
points( 5, 5, col = "red" )

points( 3, 7, col = "blue",
        pch = 3,    # símbolo
        cex = 2 )   # tamaño

points( c( 0, 0, 1 ), c( 2, 4, 6 ), # vector de puntos
        col = "green", pch = 4 )
```

Observemos que podemos introducir en el gráfico las coordenadas `x` e `y` como vectores.



El argumento `pch` te permite elegir el símbolo y con el argumento `cex` puedes asignarle un tamaño. Podemos ver los símbolos disponibles aquí

1.3. Cómo trazar varias líneas o puntos

Para dibujar dos o más funciones dentro de un mismo gráfico, empezamos dibujando función básica en R. Un ejemplo de esto podría ser :

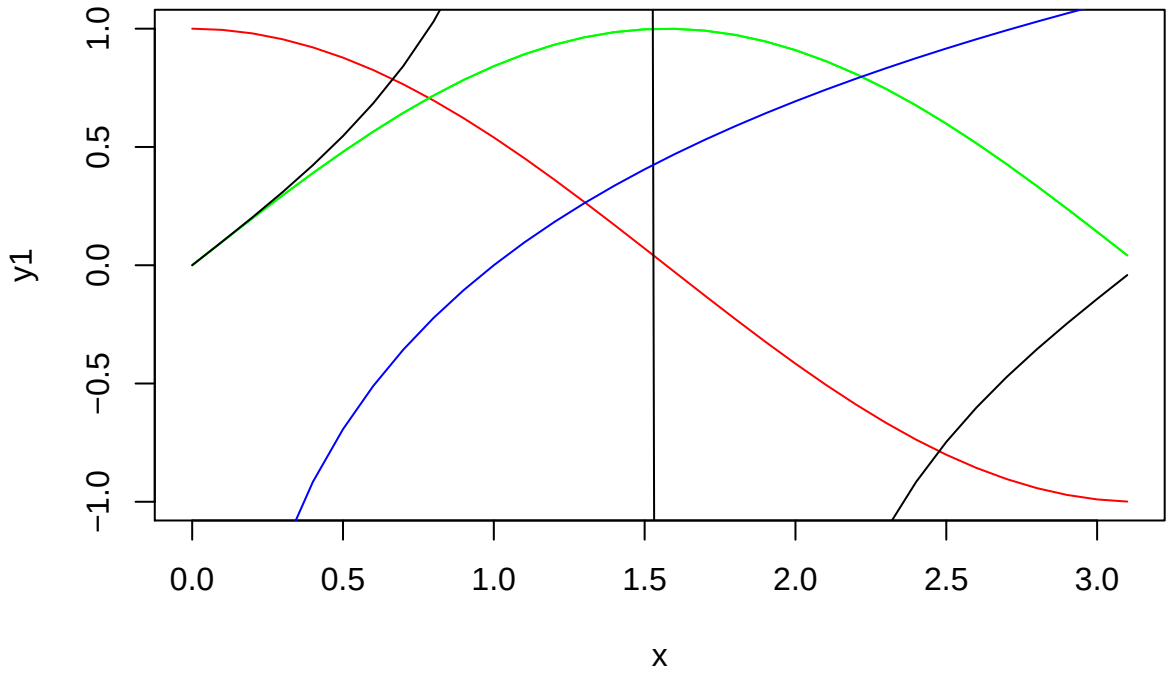
```
x <- seq( 0, pi, 0.1 )
y1 <- cos( x )
plot( x, y1,
     type = "l",    # dibuja una linea
     col = "red"    # de color rojo
     )
```

A continuación, añadimos más líneas o puntos (funciones) a la gráfica. En este caso, se incluyen más líneas, por lo que definimos más valores de y:

```
y2 <- sin( x )
y3 <- tan( x )
y4 <- log( x )
```

Finalmente, se trazan estas líneas usando la función `lines()`:

```
lines( x, y2, col = "green" )
lines( x, y3, col = "black" )
lines( x, y4, col = "blue" )
```

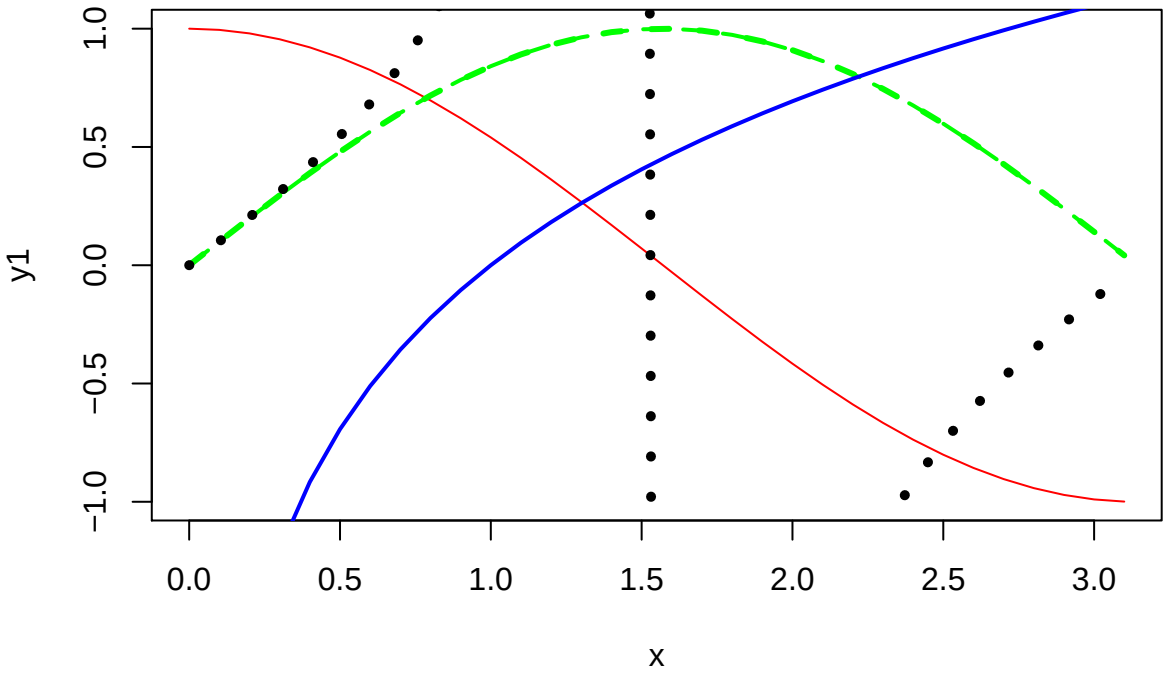


Tenga en cuenta que la función `lines()` toma tres argumentos: el eje X, el eje Y y el color (representado con el argumento `col`). También se pueden incluir las siguientes características:

Característica	Argumento	Valores de entrada
Tipo de línea	`lty`	Entero o string
Anchura de línea	`lwd`	Entero
Tipo de símbolo	`pch`	Entero o caracter
Estilo final linea	`lend`	Entero o string
Estilo linea unión	`ljoin`	Entero o string

Vemos algunos ejemplos

```
lines( x, y2, col = "green", lty = 2, lwd = 3 )
lines( x, y2, col = "green", lty = 5, lwd = 2, pch = 2 )
lines( x, y3, col = "black", lty = 3, lwd = 5, pch = 3,
       lend = 0, ljoin = 2 )
lines( x, y4, col = "blue", lty = 1, lwd = 2, pch = 3,
       lend = 2, ljoin = 1, lmitre = 2)
```



Tenga en cuenta que el argumento `pch` no funciona del todo bien con la función `lines()` y lo mejor es usarlo sólo con `points()`.

```
y5 <- x^3
points( x, y5, col = "yellow")
```

Los argumentos que hemos visto en la tabla anterior son válidos también para la función `points()`. Hay alguno para esta función, estos son:

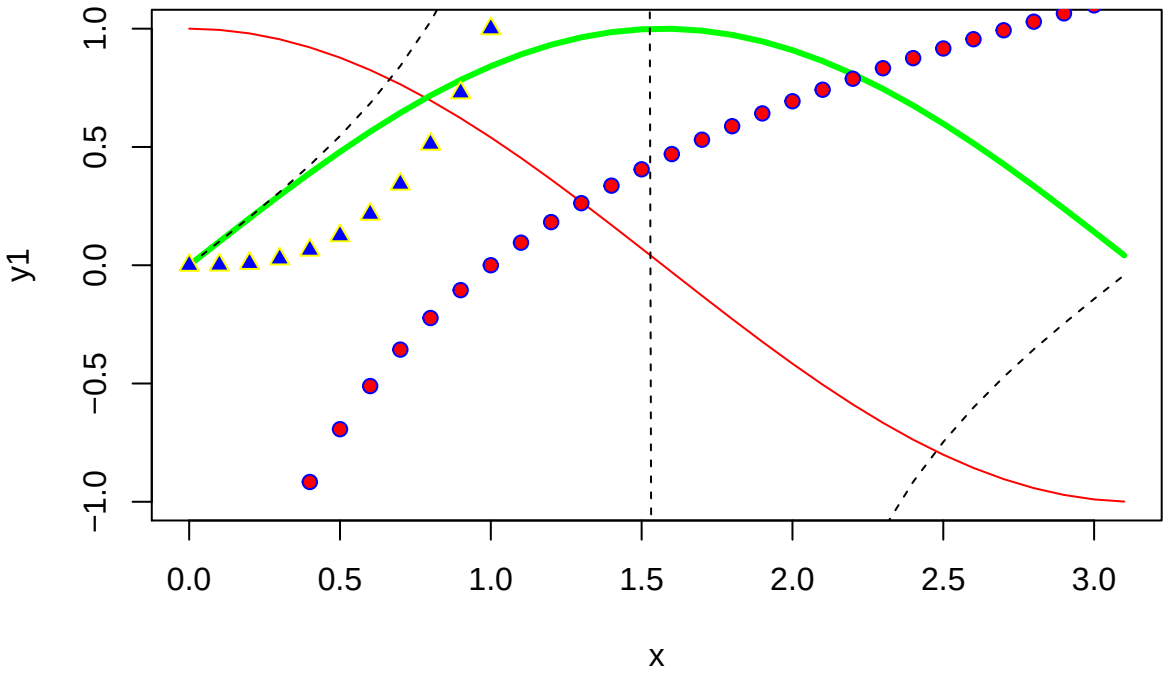
Característica	Argumento	Input
Color de fondo	`bg`	Solo si `pch = 21:25`
Tamaño	`cex`	Vaor entero

Algunos ejemplos:

```
points( x, y4, col = "blue", pch = 21, bg = "red" )
points( x, y5, col = "yellow", pch = 5, bg = "blue" )
```

Incorporando todos estos cambios al gráfico el resultado sería:

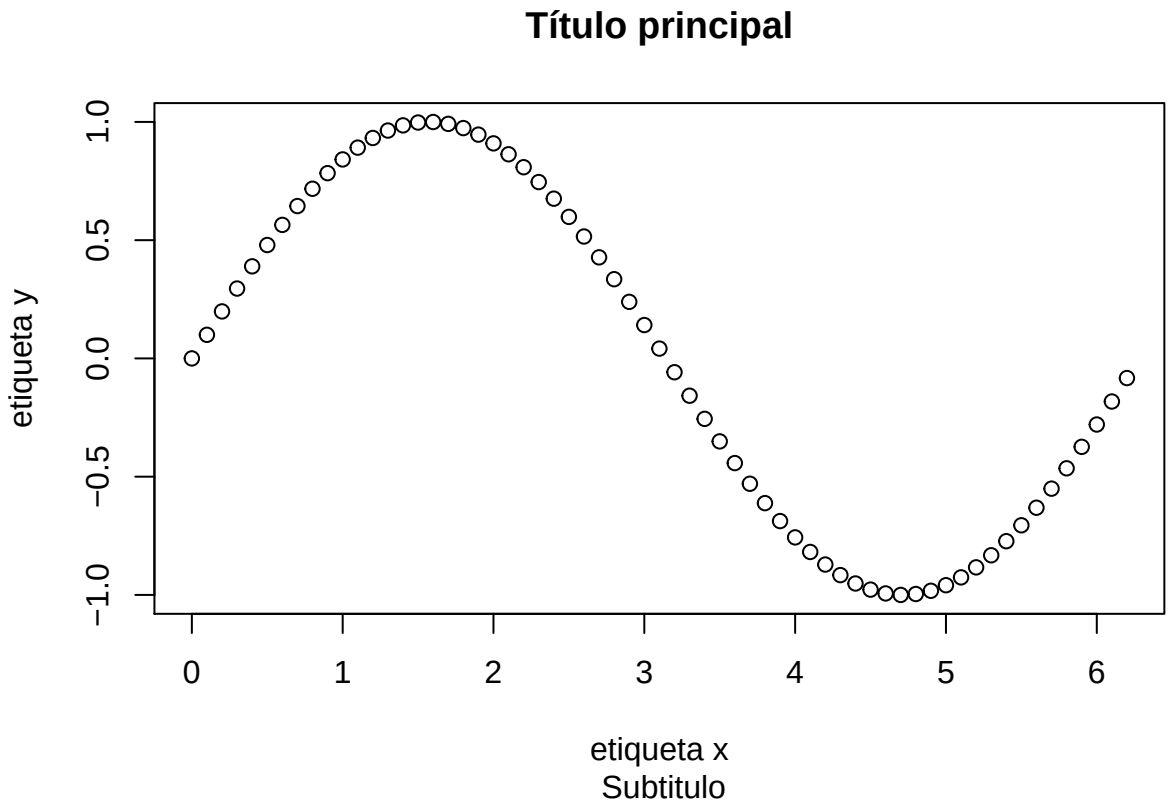
```
plot( x, y1, type = "l", col = "red" ) # gráfico básico

lines( x, y2, col = "green", lty = 1, lwd = 3 )                                  # primera capa
lines( x, y2, col = "green", lty = 3, lwd = 2, pch = 2)                          # segunda capa
lines( x, y3, col = "black", lty = 2, lwd = 1, pch = 3, lend = 0, ljoin = 2)     # tercera capa
points( x, y4, col = "blue", pch = 21, bg = "red" )                              # cuarta capa
points( x, y5, col = "yellow", pch = 24, bg = "blue" )                           # quinta capa
```



1.4. Cómo poner títulos y etiquetas

Podemos añadir a la función `plot()` los argumentos `main` para el título principal, `sub` para el subtítulo y `xlab`, `ylab` para las etiquetas de los ejes.

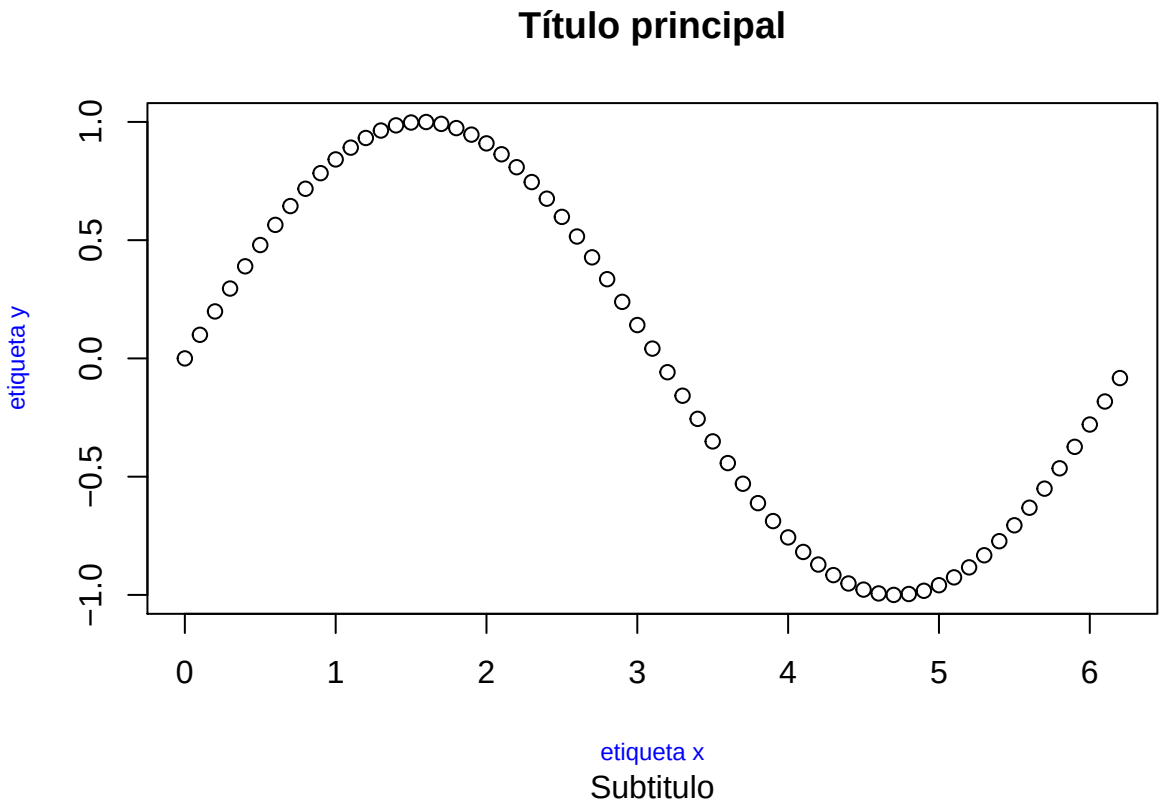
```
# Definimos el vector de puntos para X e Y
x <- seq( 0, 2*pi, 0.1 ) # secuencia
y <- sin( x )

# Dibujamos el gráfico
plot( x, y,
      main = "Título principal",
      sub = " Subtitulo",
      xlab = "etiqueta x",
      ylab = "etiqueta y"
      )
```



1.5. Cómo ajustar el aspecto de las etiquetas

Para ajustar la apariencia de las etiquetas, podemos usar los argumentos `col.lab` y `cex.lab`. El primer argumento para cambiar el color y el segundo para determinar el tamaño de las etiquetas.

```
plot( x, y,
     main = " Título principal ",
     sub = " Subtitulo ",
     xlab = "etiqueta x",
     ylab = "etiqueta y",
     col.lab = "blue",
     cex.lab = 0.75 )
```

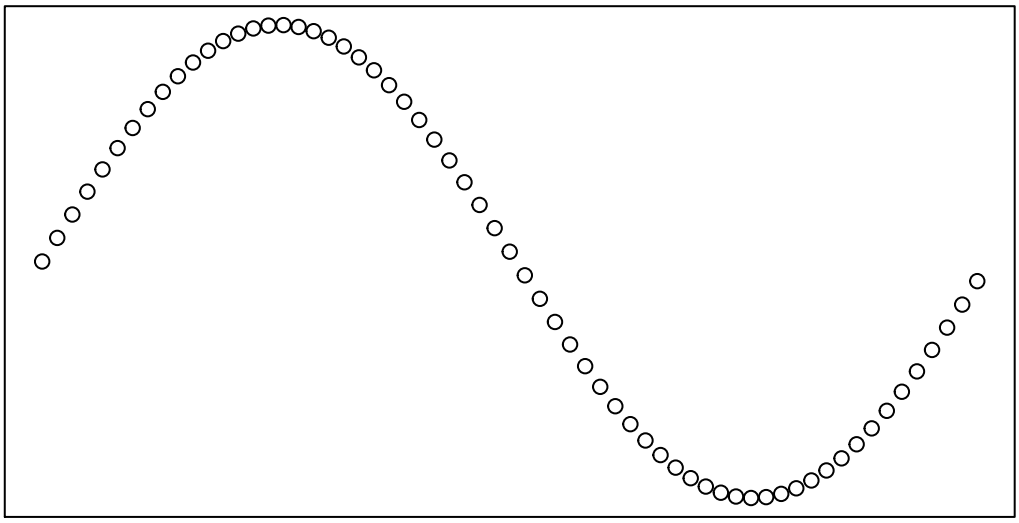


Más información sobre estos argumentos aquí

1.6. Cómo quitar las etiquetas y numeración de los ejes

Para quitar la numeración de los ejes podemos fijar los argumentos `xaxt` y `yaxt` con valor `"n"`. Lo siguiente es poner el argumento `ann = FALSE` para quitar las etiquetas de los ejes.

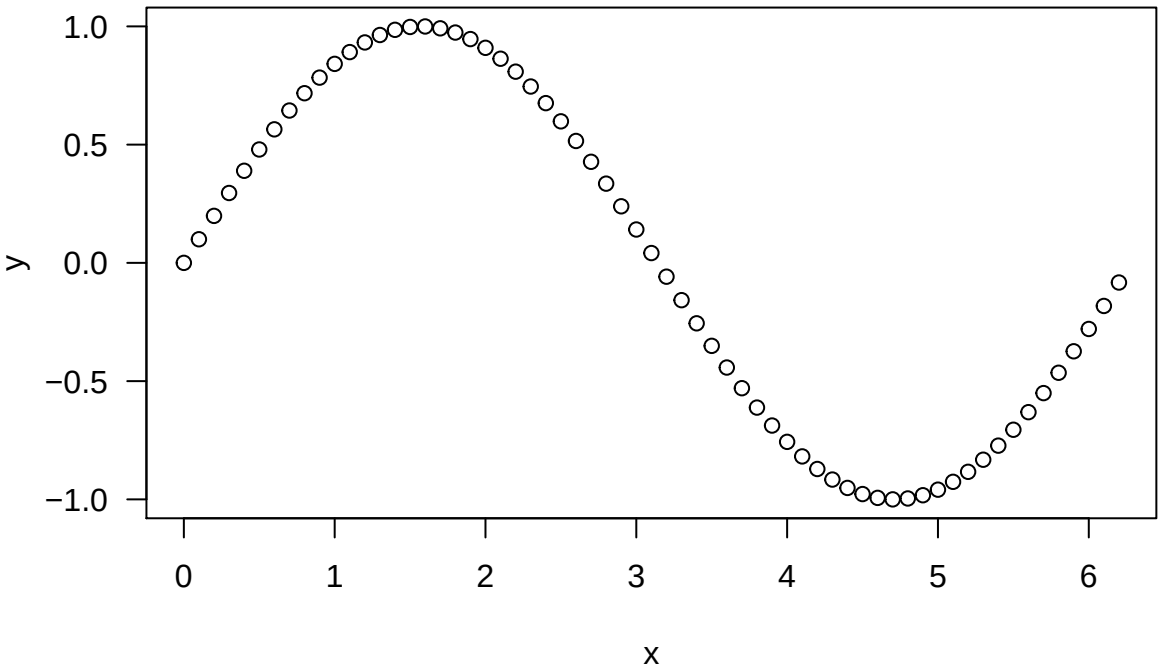
```
plot( x, y,
      xaxt = "n",
      yaxt = "n",
      ann = FALSE )
```



1.7. Cómo rotar las etiquetas de los ejes

Podemos añadir el argumento `las` a la función `axis()` para rotar los números que aparecen en los ejes:

```
plot( x, y, axes = FALSE )
box( )
axis( 1, las = 0 ) # abscisas (1)
axis( 2, las = 2 ) # ordenadas (2)
```



Observamos que es necesario añadir previamente en la función `plot()` el argumento `axes = FALSE`, para indicar que (aún) no hay ejes que dibujar. Los ejes los dibujamos a continuación con la función `axis()`.

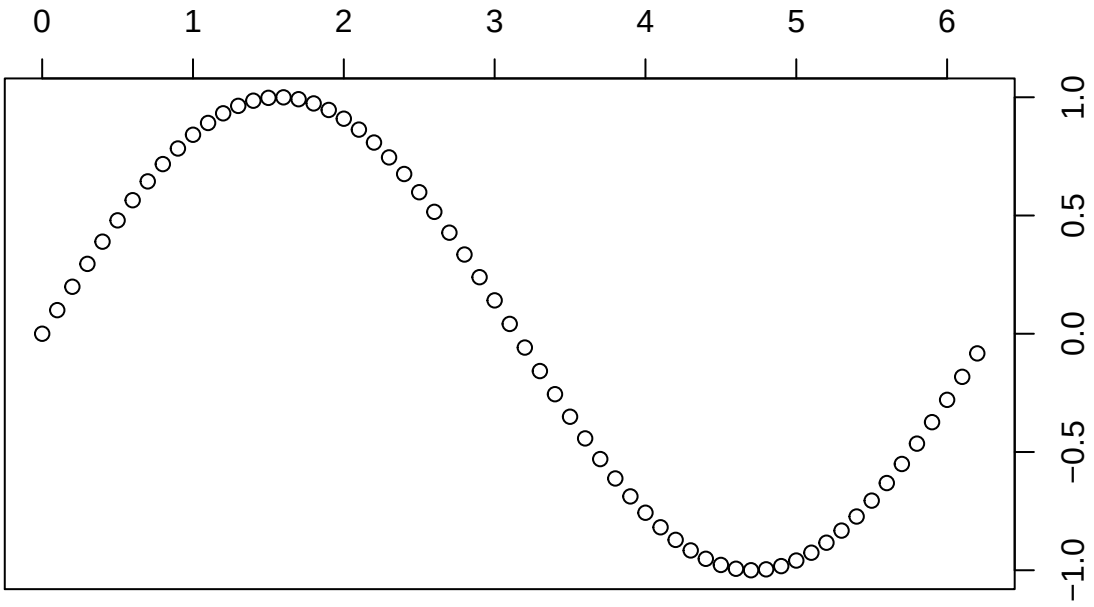
En el gráfico anterior hemos utilizado el argumento `las`. Este puede tener cuatro valores:

- `0`: etiqueta paralela a los ejes (por defecto)
- `1`: etiqueta en horizontal a los ejes
- `2`: etiqueta perpendicular a los ejes
- `3`: etiqueta en vertical a los ejes.

1.8. Cómo mover la posición de los ejes

Podemos hacerlo con la función `axis()`. Los valores por defecto son 1 (eje abscisas abajo) y 2 (eje coordenadas a la izquierda). Para situar los ejes arriba y a la derecha utilizamos los valores 3 y 4.

```
plot( x, y, axes = FALSE, ann = FALSE )
axis( 3 )   # Eje x encima del área del gráfico
axis( 4 )   #  Eje y a la derecha del aŕea del gráfico
box()       # Añadimos la caja
```



Como podemos ver, primero representamos los vectores `x` e `y` dejando fuera los ejes y las etiquetas, y después añadimos los ejes tal cómo queremos y dónde queremos mediante la función `axis()`.

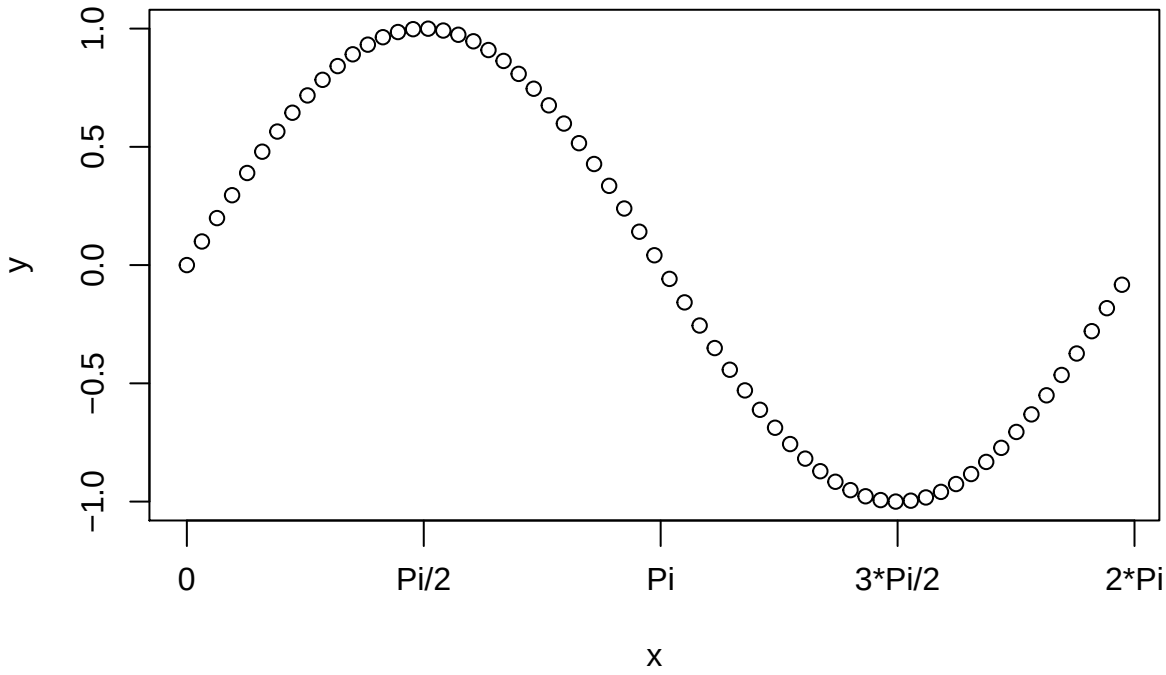
1.9. Cómo cambiar el espaciado de las marcas de los ejes

Hay dos formas de hacerlo:

1. Usando la función `axis()`

Definimos dos vectores `pos` y `etiq` con las posiciones donde queremos colocar las marcas y las etiquetas. Dibujamos el gráfico y le añadimos estas marcas con la función `axis()`.

```
pos <- c( 0, pi/2, pi, 3*pi/2 ,2*pi ) #  definir la posición de los puntos
etiq <- c( "0", "Pi/2", "Pi", "3*Pi/2", "2*Pi" ) # etiquetas de los puntos

x <- seq( 0, 2*pi, 0.1 ); y <- sin( x )
plot( x, y, xaxt = "n" )

axis( side = 1,      # eje inferior
      at = pos,       # posiciones
      labels = etiq,  # etiquetas
      tck = -.05 )
```



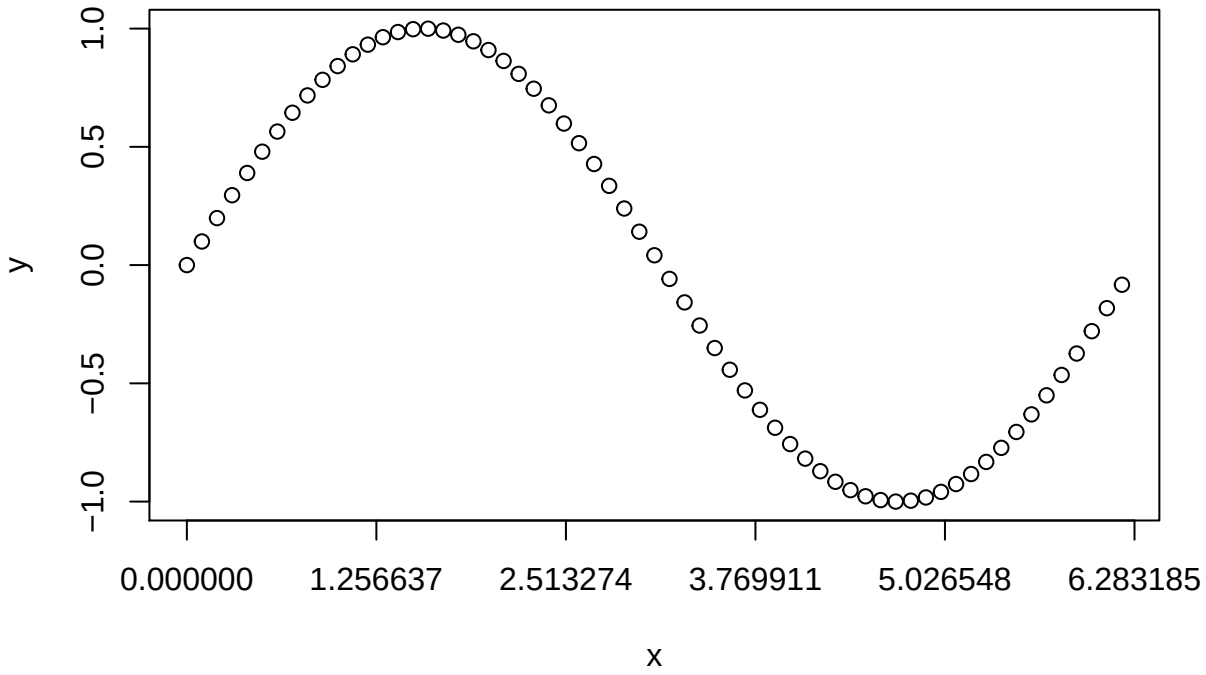
- `side:` valor '1' significa que el eje se dibuja debajo del gráfico, '2' se dibuja a la izquierda, '3' se dibuja arriba y valor '4' se dibuja a la derecha.
- `tck`: ajusta la dirección de los puntos, si proporcionamos un valor negativo, especificamos que las marcas de los puntos ("rallitas") deben aparecer debajo de los ejes.

Podemos ajustar además el tamaño de las marcas mediante el argumento `tcl` y su aspecto con las funciones `cex.axis`, `col.axis` y `font.axis`.

2. Usando el argumento `xaxp`

Mediante el argumento `xaxp` de la función `plot()` podemos indicar las posiciones extremas de las marcas y el número de intervalos que los que queremos dividirlos.

```
x <- seq( 0, 2*pi, 0.1 )
y <- sin( x )
plot( x, y,
     xaxp = c( 0, 2*pi, 5 ) )
```



Obs: este argumento sólo funciona si NO usamos escala logarítmica. Más información aquí

1.10. Cómo crear dos ejes X o Y diferentes

La primera opción es crear un primer gráfico y ejecutar `par( new = TRUE )` para evitar que R limpie del *devide gráfico* al ejecutar el segundo gráfico. Vemos los pasos uno a uno:

1. Dibujamos el primer gráfico

```
set.seed( 101 )
x <- 1:10; y <- rnorm( 10 );
z <- runif( 10, min = 1000, max = 10000 )
plot( x, y )
par( new = TRUE )  # Permite que el siguiente gráfico se dibuje sobre este
```

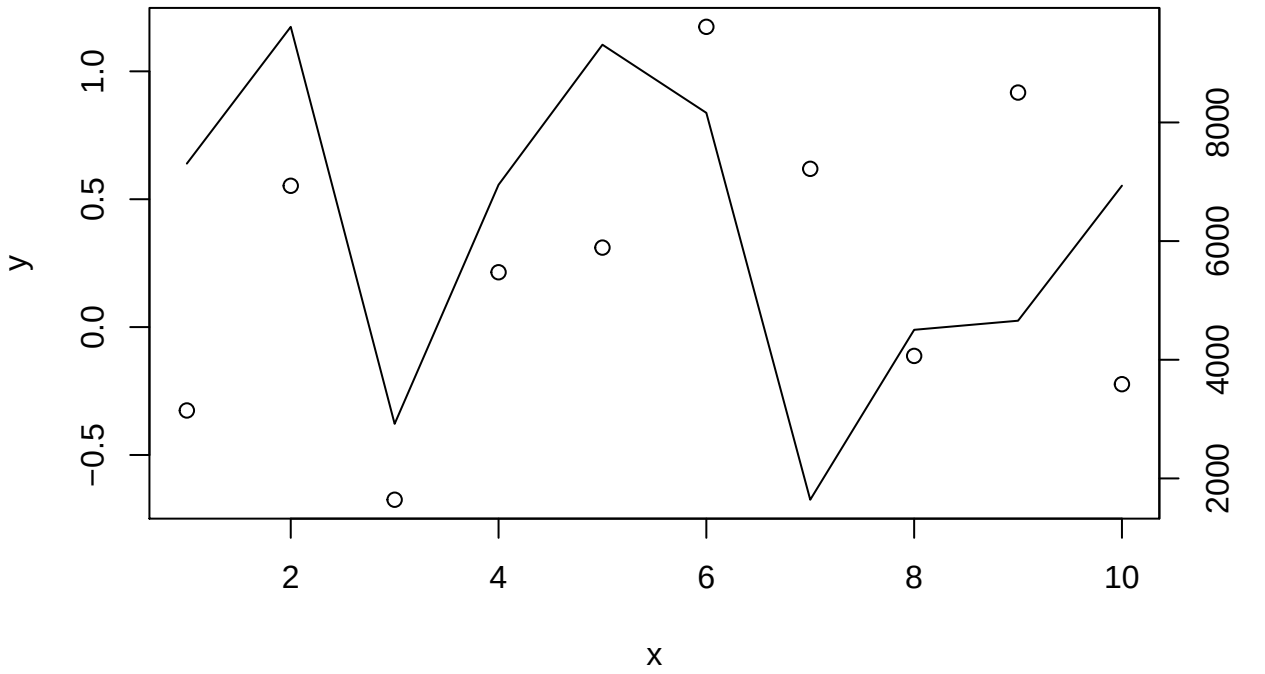
2. Creamos el segundo gráfico llamando de nuevo a la función `plot()`:

```
plot( x, z,
     type = "l",    # tipo de punto: linea
     axes = FALSE,  # No ejes
     bty = "n",     # eliminamos la caja del gráfico
     xlab = "",     # No ponemos etiquetas en los ejes
     ylab = "" )
```

Obs: hemos puesto tanto el argumento `axes = FALSE`, como las etiquetas de `x` e `y` en blanco, evitando sobreescribir los ejes del gráfico anterior.

Si queremos podemos añadir un nuevo eje a la derecha del gráfico ejecutando `axis` con argumento `side = 4`. Para indicar la posición de las marcas usamos `at`.

```
axis( side = 4,
     at = pretty( range( z ) ) # dónde dibujar las marcas

mtext( "z", side = 4,
      line = 3 )
```



Obs: el argumento `side` puede tener cuatro valores asignados:

- 1 para situar el texto debajo del gráfico
- 2 para situarlo a la izquierda
- 3 para situarlo encima
- 4 para poner el texto a la derecha.

Y el argumento `line` sirve para indicar en qué linea del margen empieza el texto.

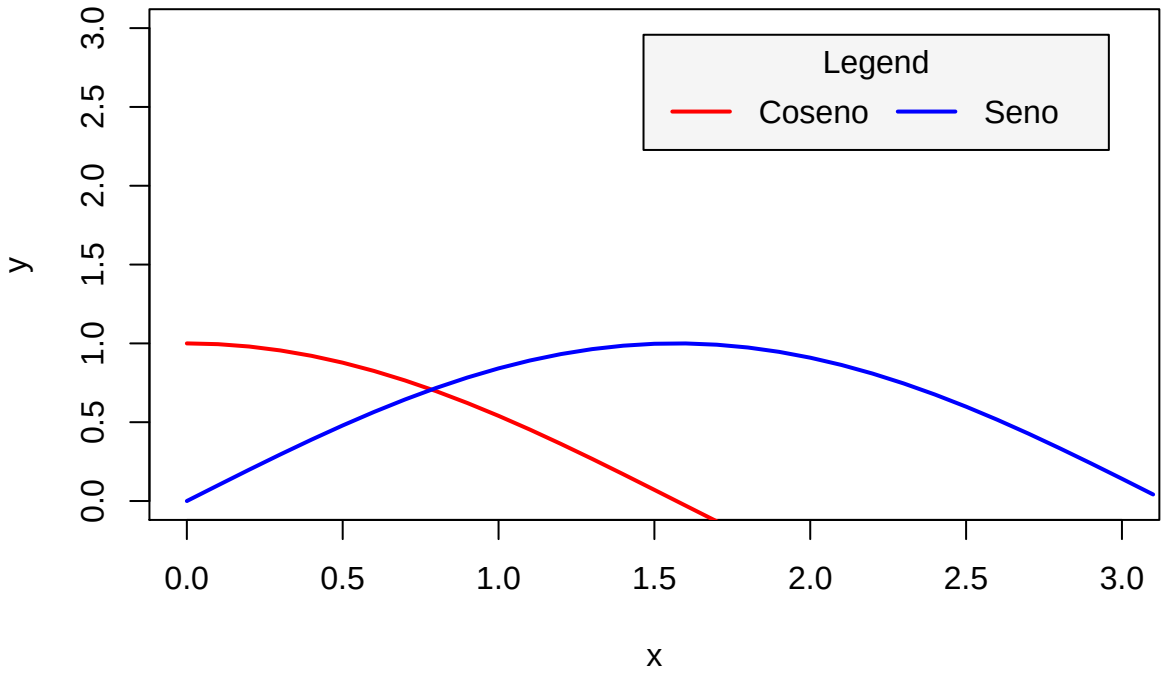
Ejercicio: intenta construir un gráfico con dos ejes de abscisas diferentes. Puedes encontrar la solución a continuación:

```
plot.new( )
set.seed( 101 )
x <- 1:10
y <- rnorm( 10 )
z <- runif( 10, min = 1000, max = 10000 )

par( mar = c( 5, 4, 4, 4) + 0.3 )  # ajustamos los márgenes del gráfico
plot( x, y )
par( new = TRUE )
plot( z, y, type = "l", axes = FALSE, bty = "n", xlab = "", ylab = "")
axis( side = 3, at = pretty( range( z ) ) )
mtext( "z", side = 3, line = 3 )
```

1.11. Cómo añadir una leyenda

Podemos añadirla fácilmente usando la función `legend()`:

```
legend( "topright",                 # posición
       inset = .05,                 # distancia a los ejes
       cex = 1,                     # tamaño
       title = "Leyenda",           # título
       c( "Coseno","Seno"),         # etiquetas
       horiz = TRUE,                # posición
       lty = c( 1, 1 ),             # tipo de linea
       lwd = c( 2, 2 ),             # anchura de la linea
       col = c( "red","blue" ),     # colores de las lineas
       bg = "grey96"                # fondo
       )
```



1.12. Cómo dibujar dos gráficos uno al lado del otro

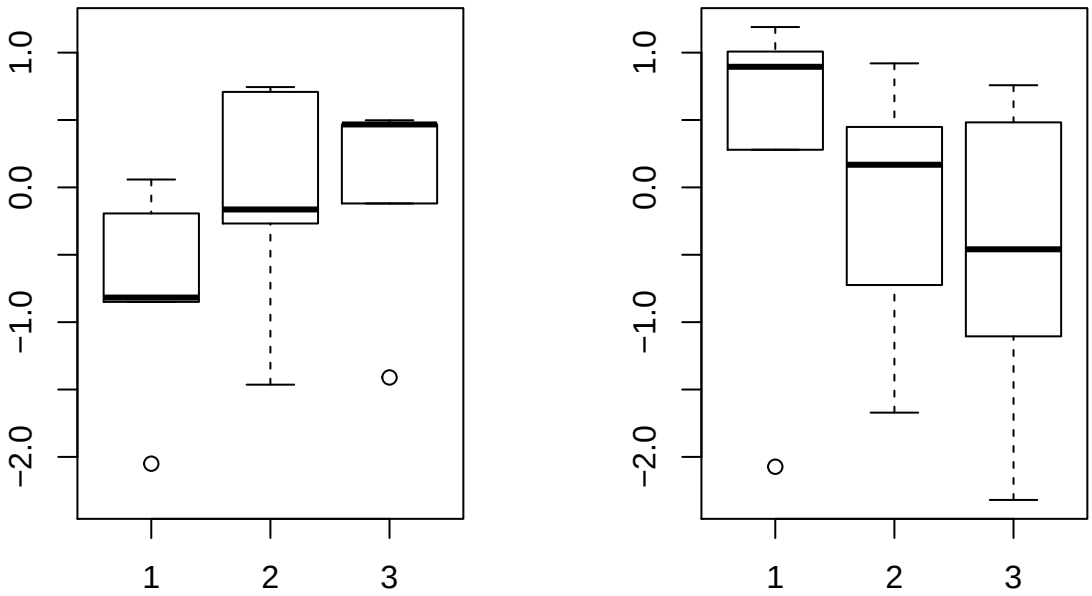
Podemos hacer esto con la función `par()` de R

```
par( mfrow = c( 1, 2 ) )
```

Le hemos pasado como argumento `mfrow = c( 1, 2 )`. Esto significa que los gráficos se dibujaran en una matriz de 1 fila por 2 columnas. Lo vemos

```
par( mfrow = c( 1, 2 ) ) # Dividimos el área gráfica

# Valores que vamos a representar
d0 <- matrix( rnorm( 15 ), ncol = 3 )
d1 <- matrix( rnorm( 15 ), ncol = 3 )

limits <- range(d0,d1) # limites de los boxplot
```

```r
# Boxplot
boxplot( d0, ylim = limits )
boxplot( d1, ylim = limits )
```

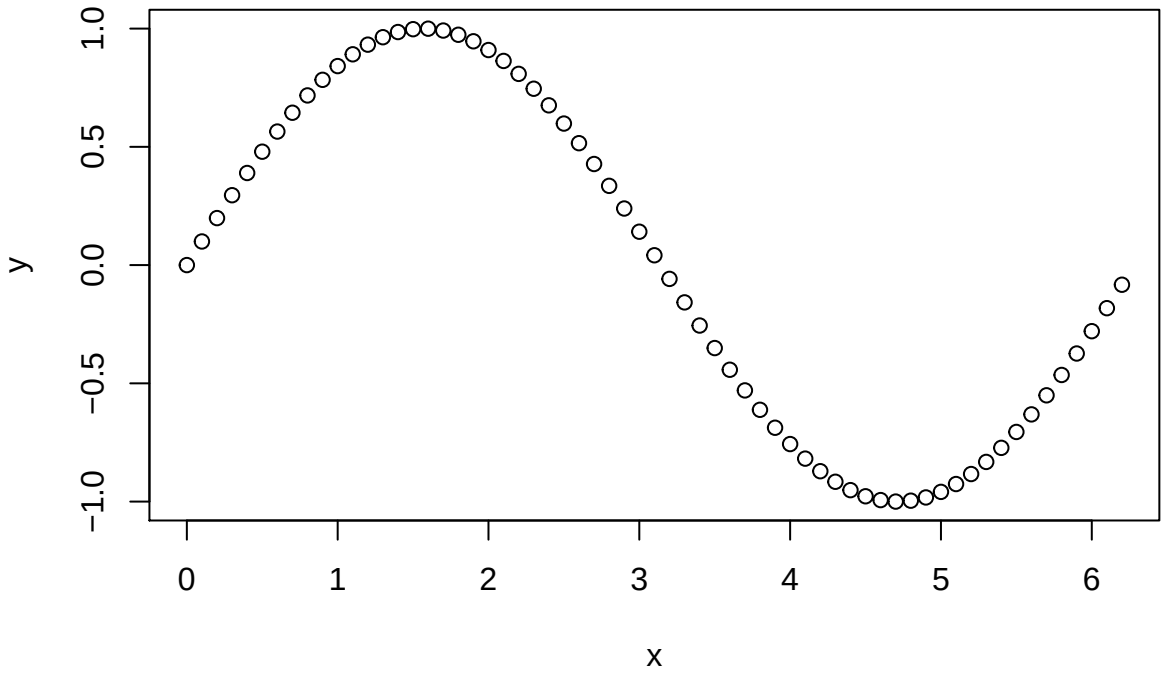


1.13. Cómo controlar las escalas de tus gráficos

Para que los ejes del gráfico sean proporcionales a su tamaño, podemos añadir el argumento `asp` de la función `plot()` y asignarle un valor entero.

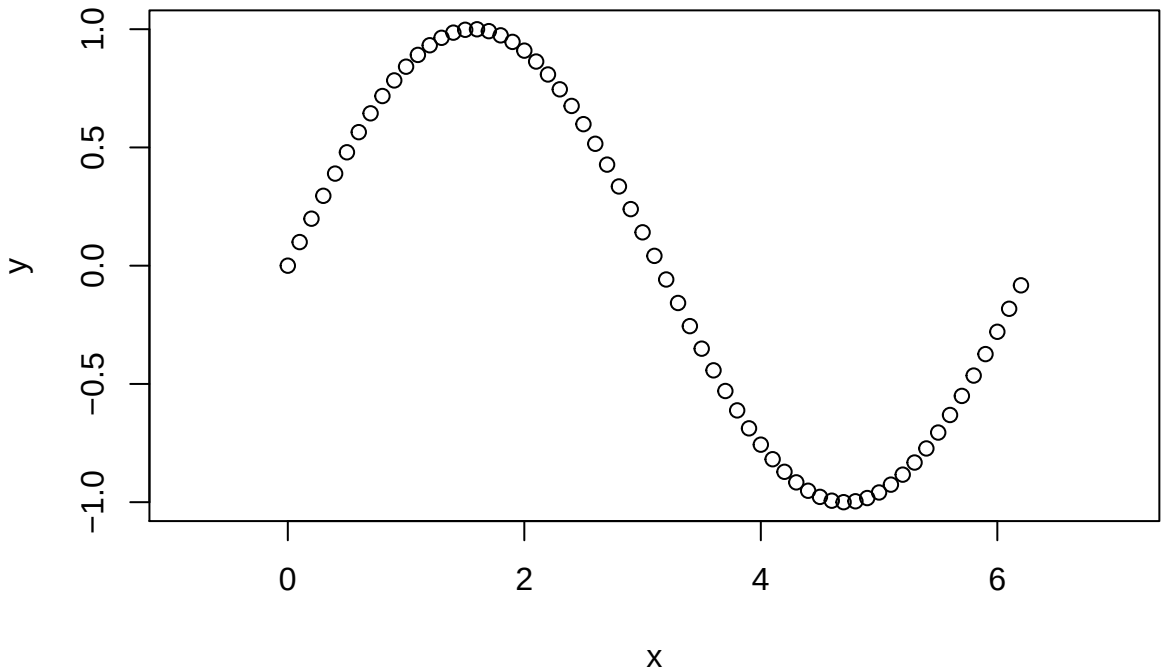
Mira este primer ejemplo sin una relación de aspecto definido:

```r
x <- seq( 0, 2*pi, 0.1 )
y <- sin( x )
plot( x, y )
```



Y compáralo con este gráfico dónde la relación de aspecto ha sido definida con el argumento `asp`:

```
x <- seq( 0, 2*pi, 0.1 )
y <- sin( x )
plot( x, y,
     asp = 2 )
```



R por defecto busca la máxima dispersión a la hora de representarlos. ¿Representar siempre así? ¿O hay veces que prefiero utilizar mis escalas?

R podemos indicar la relación entre los ejes con el argumento `asp`

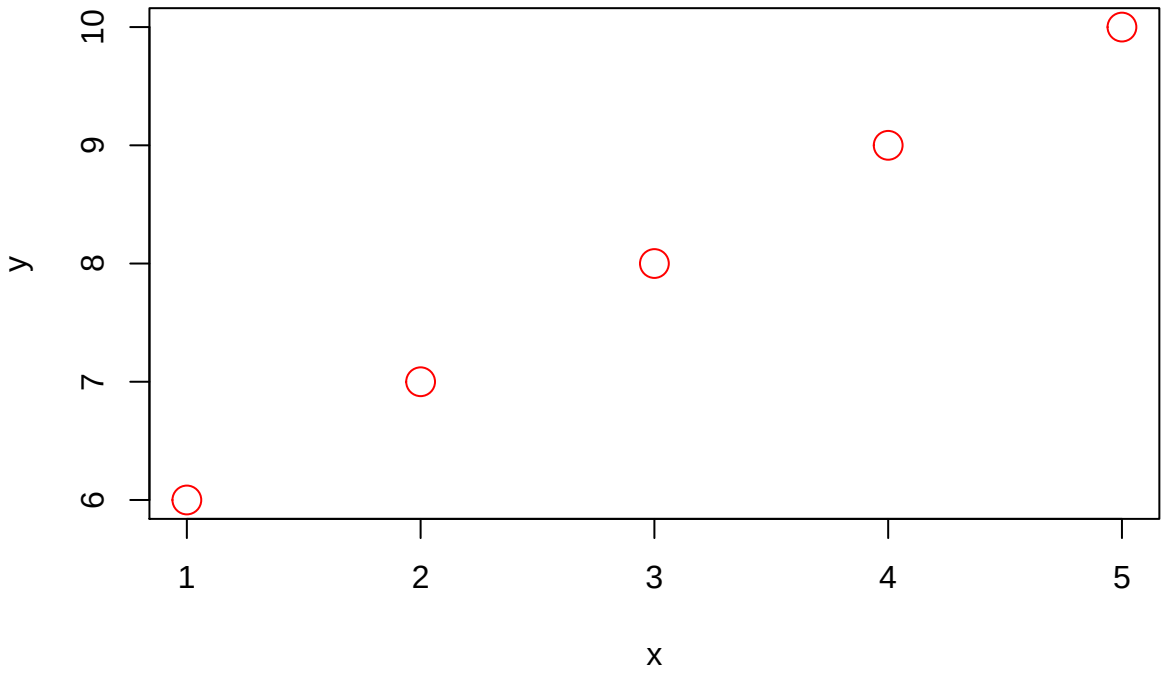
- `asp = 1`, el mismo los dos

- asp = 0.5
- asp = 2

1.14. Cómo ajustar el tamaño de los puntos

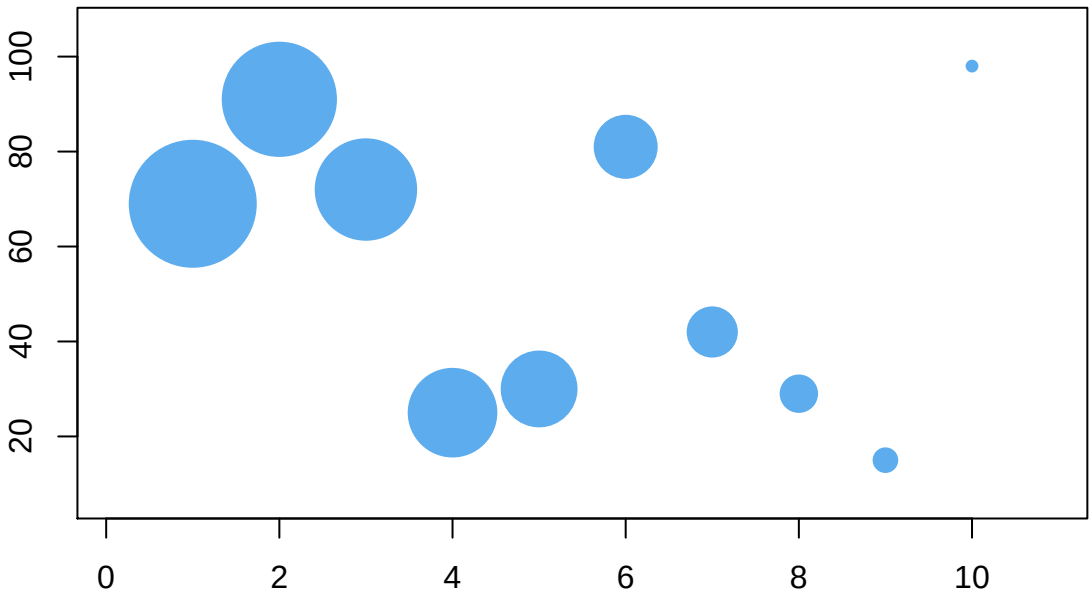
Para ajustar el tamaño de los puntos podría simplemente utilizarse el argumento `cex`:

```
x <- c( 1, 2, 3, 4, 5 )
y <- c( 6, 7, 8, 9, 10 )
plot( x, y,
      cex = 2 ,
      col = "red" )
```



Además, R permite tener mucho más control sobre sus símbolos a través de la función `symbols()`:

```
df <- data.frame( x = 1:10, y = sample( 10:99, 10 ),
                  rad = 10:1 )

symbols( df$x, df$y,
        circles = df$rad,       # radio de los círculos
        inches = 1/3,           # controla el tamaño
        ann = FALSE,
        bg = "steelblue2",      # color de fondo
        fg = NULL               # color del borde
        )
```



Los círculos del gráfico anterior reciben los valores de `df$rad` como el radio, mientras que el argumento `inches` controla el tamaño de los símbolos. Cuando este último argumento recibe como valor un número positivo, los símbolos se escalan para hacer más grande la dimensión de este tamaño en pulgadas .

1.15. Cómo ajustar una curva a nuestros datos

La función `loess()` utiliza un ajuste locar para determinar una superficie polinomial según uno o más predictores numéricos. Supongamos que se tienen los siguientes puntos:

```
x <- 1:10
y <- c( 2, 4, 6, 8, 7, 12, 14, 16, 18, 20 )
```

Y usamos la función `loess()` para relacionar `x` e `y`. Al realizar esto, especifamos la variable respuesta y los predictores.

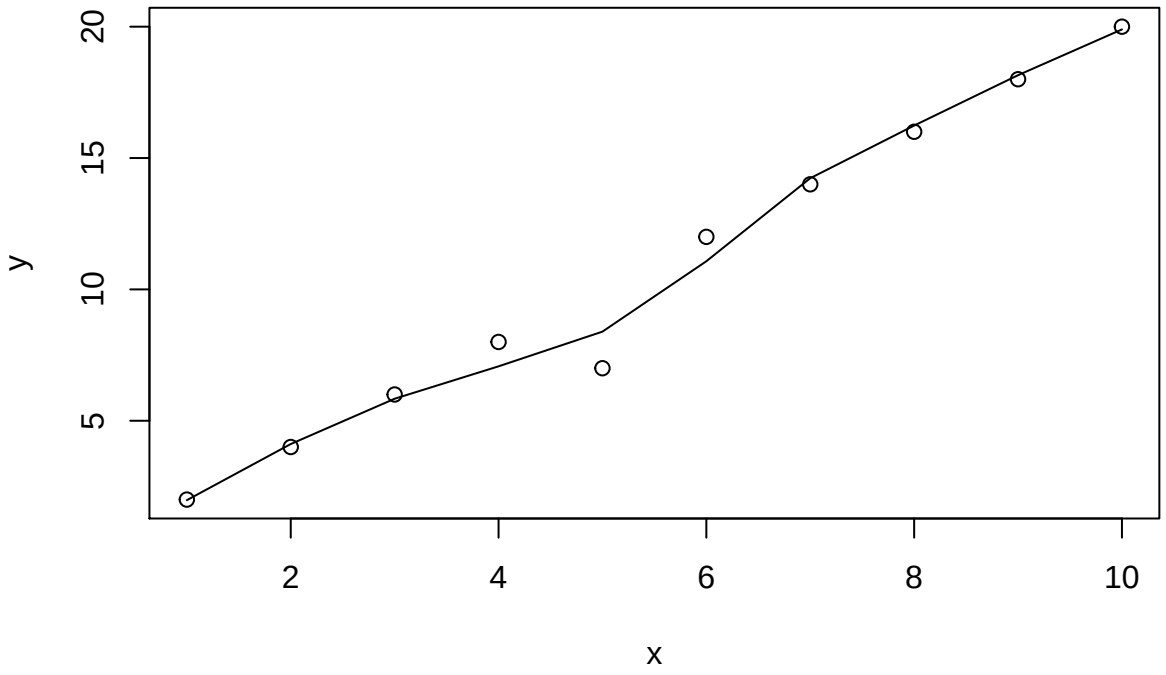
```
lo <- loess( y ~ x ) # estimaciones entre los datos
```

Representamos `x` e `y`

```
plot( x, y )
```

Y añadimos las líneas donde se predicen los valores de `lo`

```
lines( predict( lo ) )
```



1.16. Cómo exportar un gráfico a como imagen

Se puede utilizar la función `dev.copy()` para exportar el gráfico a una ruta especificada.

```
x <- seq( 0, 2*pi, 0.1 )
y <- sin( x )
plot( x, y )
dev.copy( jpeg,
          filename = "/home/elvira/prueba.jpg");
dev.off( )   # cerramos el device
```

```
x <- seq(0,2*pi,0.1)
y <- sin(x)
plot(x,y)
dev.copy(jpeg,
          filename="/home/elvira/prueba.jpg");
dev.off()
```