R: Graphics

140.776 Statistical Computing

August 21, 2011

Run R program from a file

Create a file ex1.R that contains two lines:

```
x<-rnorm(1000)
hist(x)
```

Run R program from a file

```
>source("ex1.R")
```

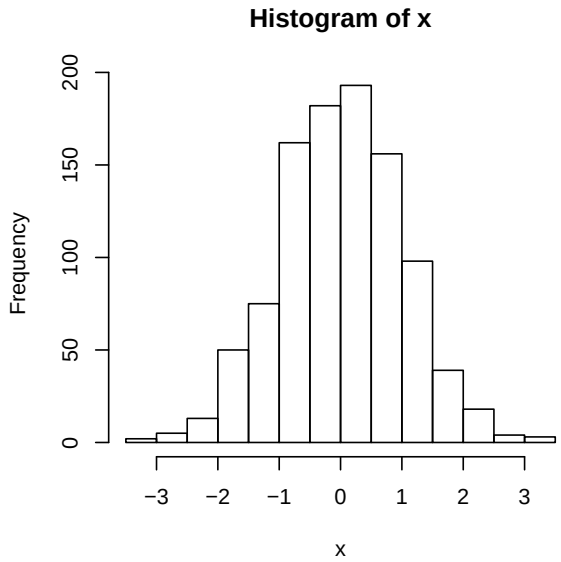
Basic graphics

Plotting in R is easy. There are many functions for plotting your data:

- plot(): 2-D graphics
- boxplot(): box plot
- hist(): histogram
- qqplot(): QQ plot
- ...

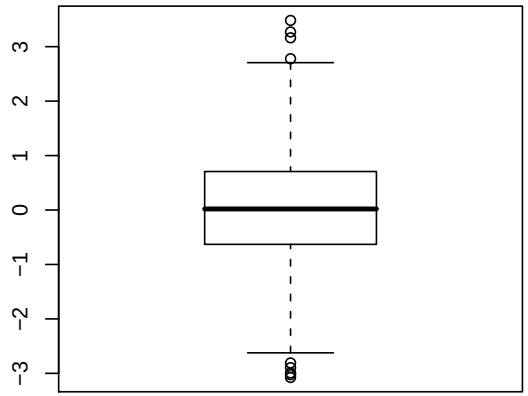
Histogram

```
> x<-rnorm(1000)
> hist(x)
```



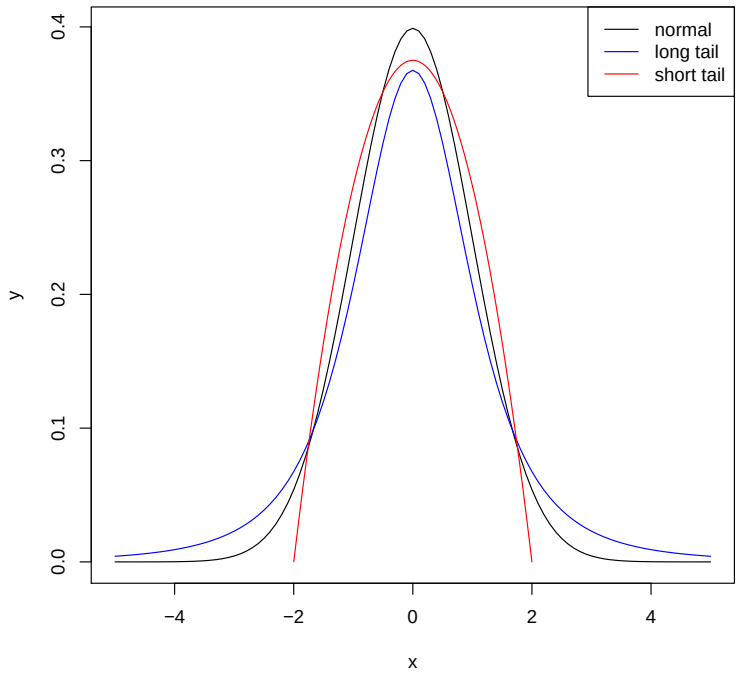
Box plot

```
> boxplot(x)
```



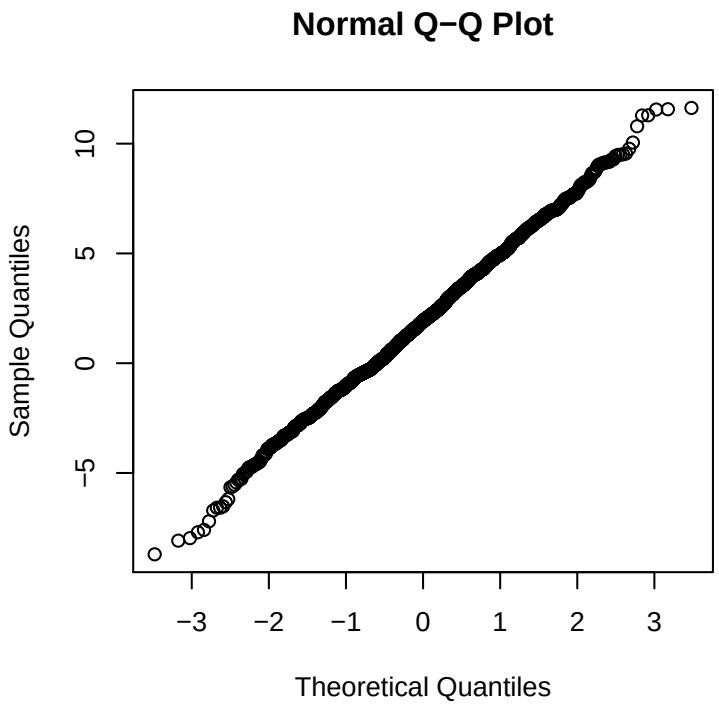
Normal QQ plot

You can use qqnorm() to check whether data are collected from a normal, a long tail, or a short tail distribution



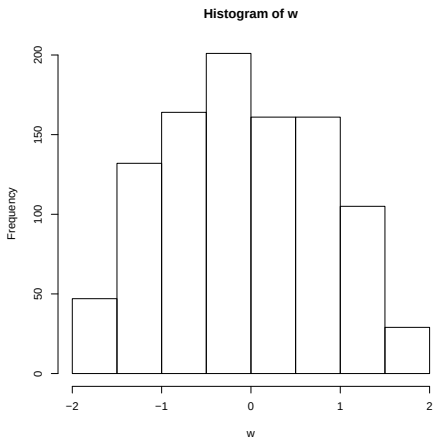
Normal QQ plot

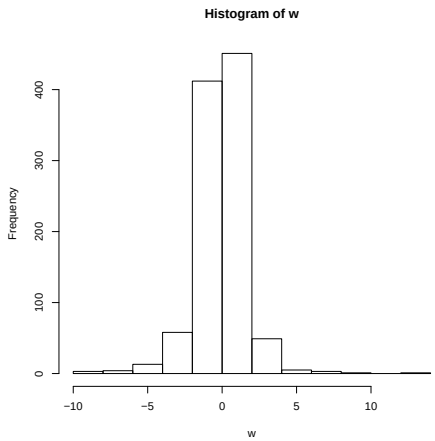
```
> y<-rnorm(2000, mean=2, sd=3)
> qqnorm(y)
```

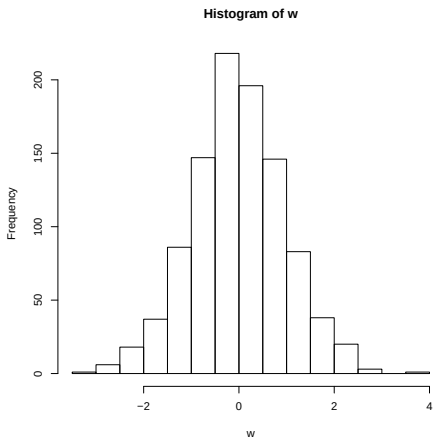


Exercise

Which one of the following is long tail? Short tail? Normal?

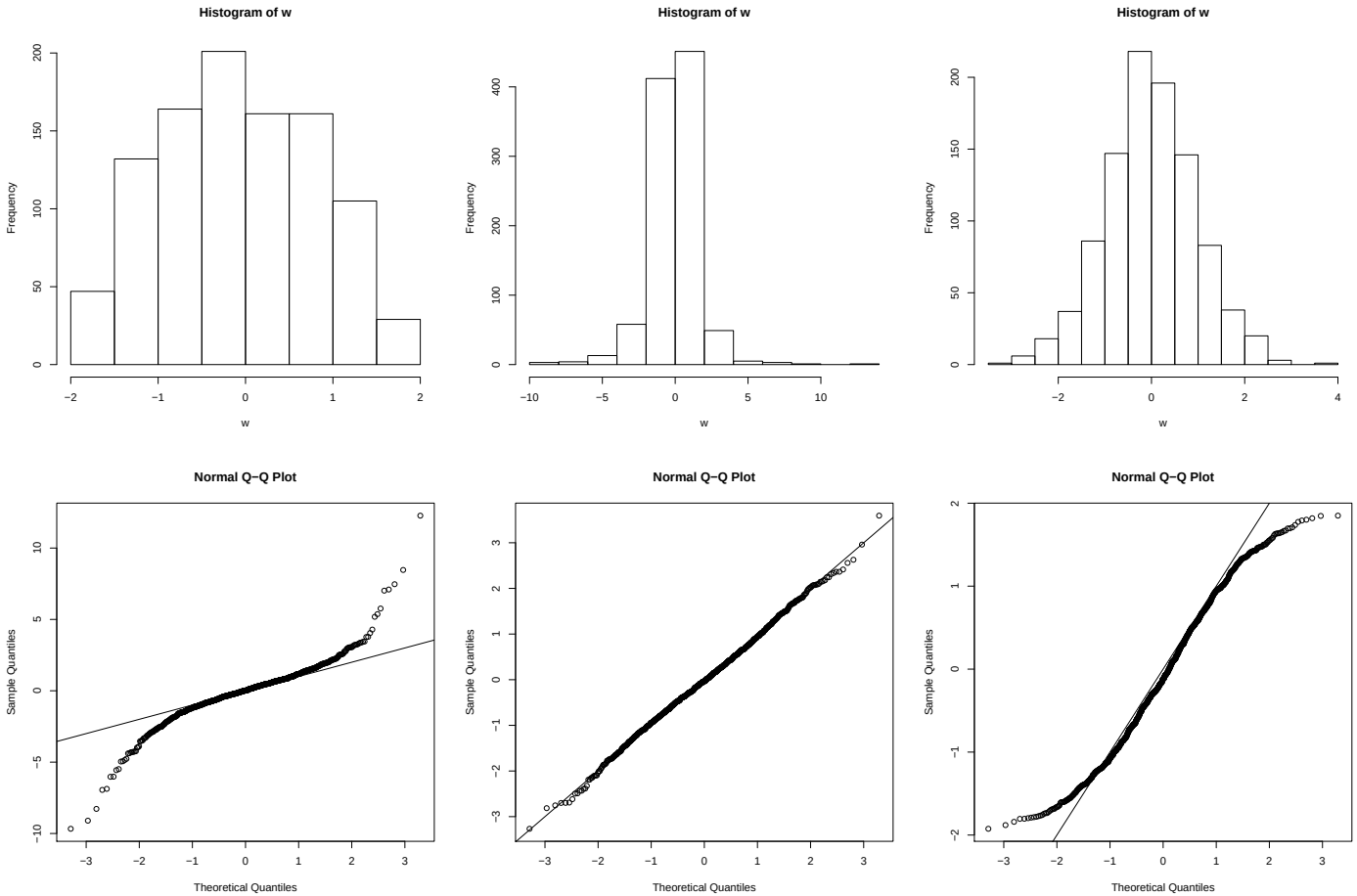






Exercise

Find the corresponding normal QQ plot for each histogram:



Normal QQ plot

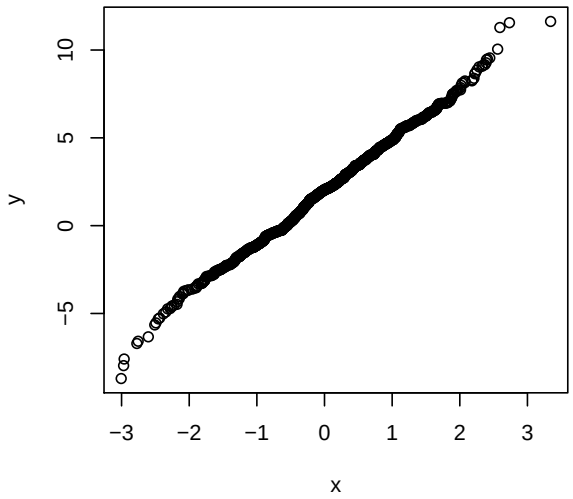
If you want to find out, you can try t-distribution (a long tail distribution)

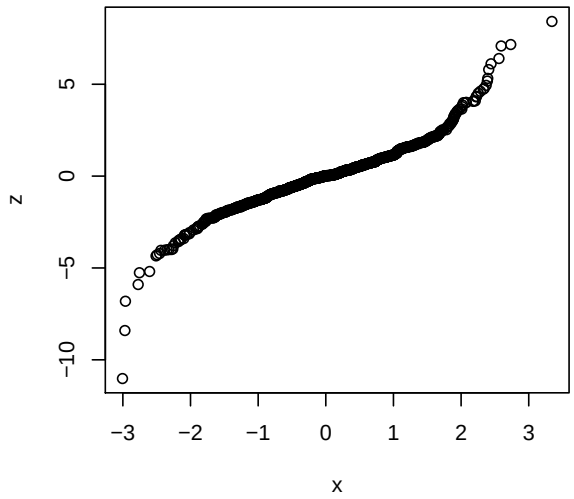
```
> w<-rt(1000,df=3)
> hist(w)
> qqnorm(w)
```

QQ plot

qqplot() allow you compare two distributions:

```
> x<-rnorm(1000)
> y<-rnorm(2000, mean=2, sd=3)
> z<-rt(1000,df=3)
> qqplot(x,y)
> qqplot(x,z)
```

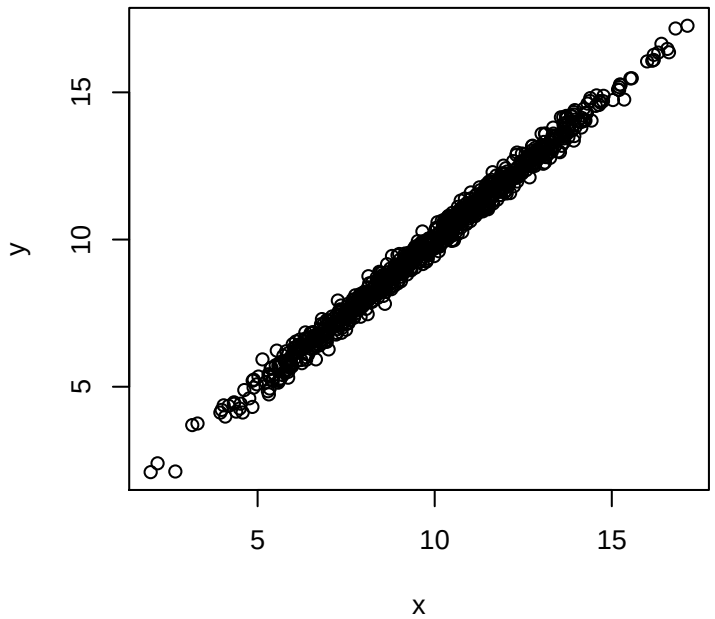




Plot

plot() is perhaps the most frequently used plotting function in R. Let us study $Y = X + \epsilon$, where $X \sim N(10, 2.5^2)$ and $\epsilon \sim N(0, 0.25^2)$.

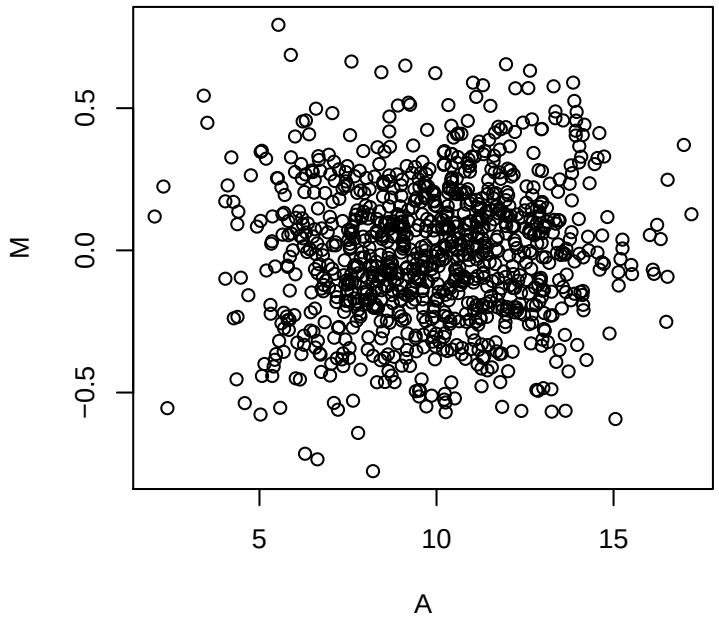
```
> x<-rnorm(1000,mean=10,sd=2.5)
> y<-x+rnorm(1000,mean=0,sd=0.25)
> plot(x,y)
```



Plot

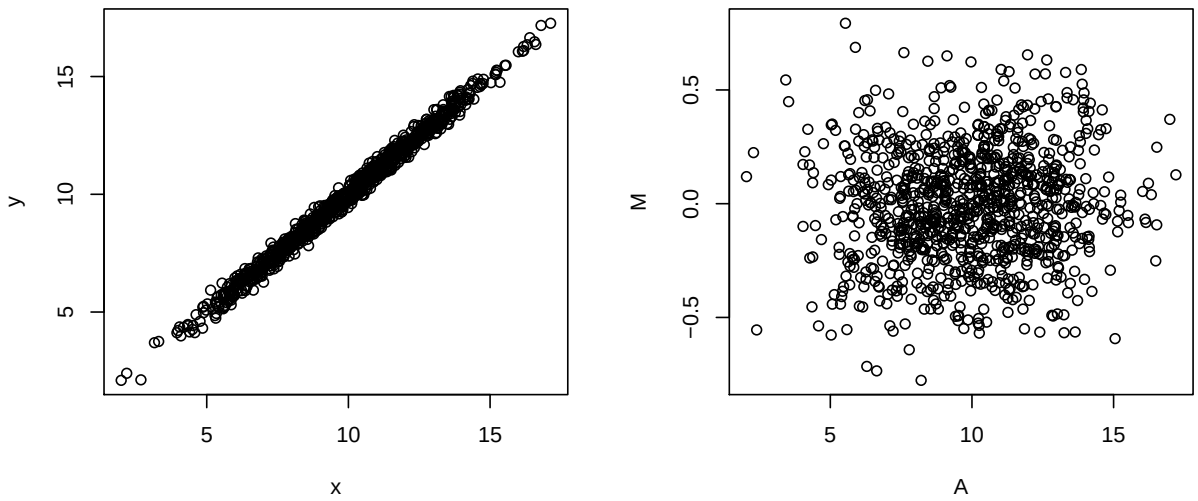
Now let us rotate the plot 45° and plot (y-x) vs. (y+x)/2. This is the so called "M-A plot".

```
> M<-y-x
> A<-(y+x)/2
> plot(A,M)
```



Setting graphical parameters

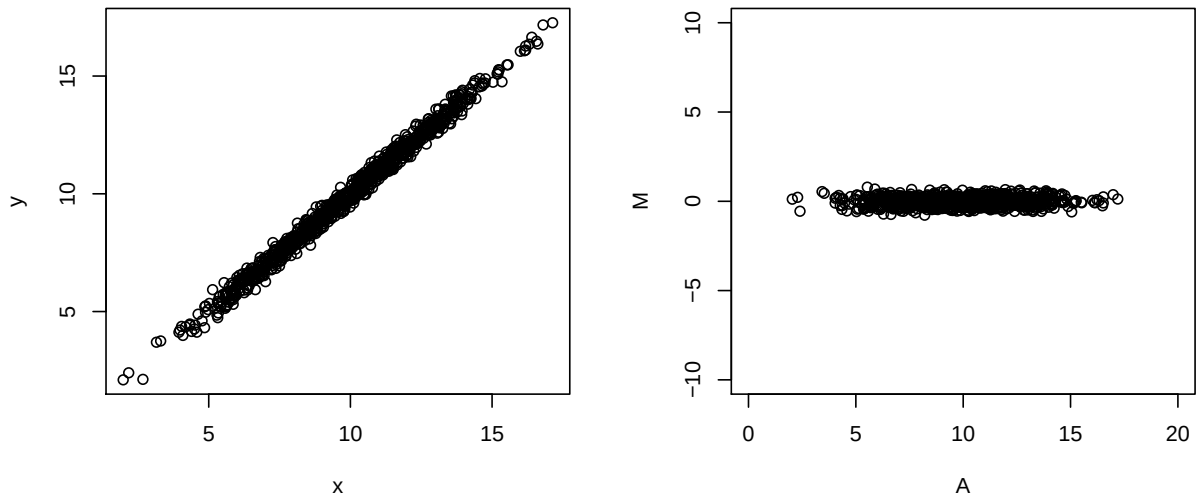
The MA plot looks quite different from the original plot. Why?



Plot

If you want to have the two figures on a similar scale, you can use the xlim and ylim options of the plot() function:

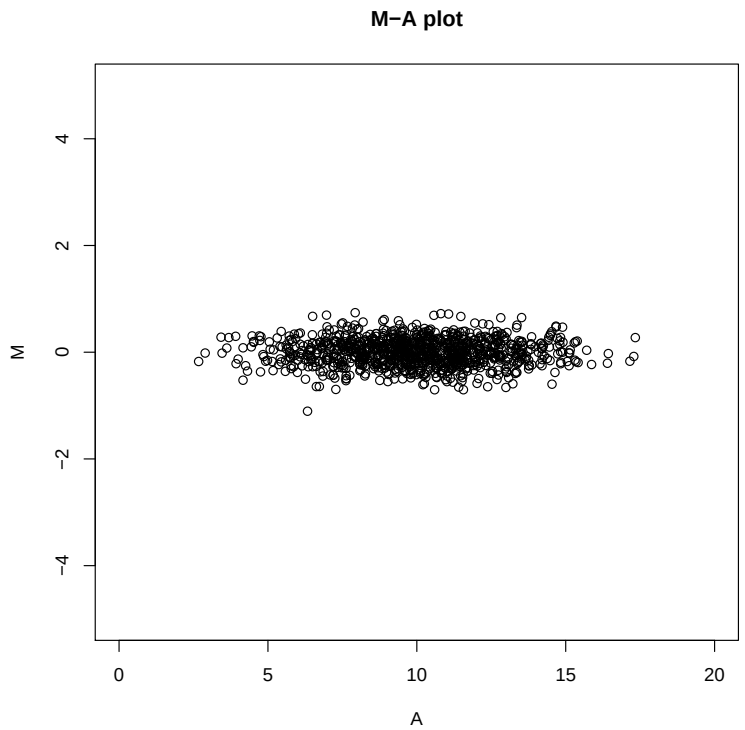
```
> plot(A,M, xlim=c(0,20), ylim=c(-10,10))
```



Plot

Indeed, there are a lot of parameters you can adjust.

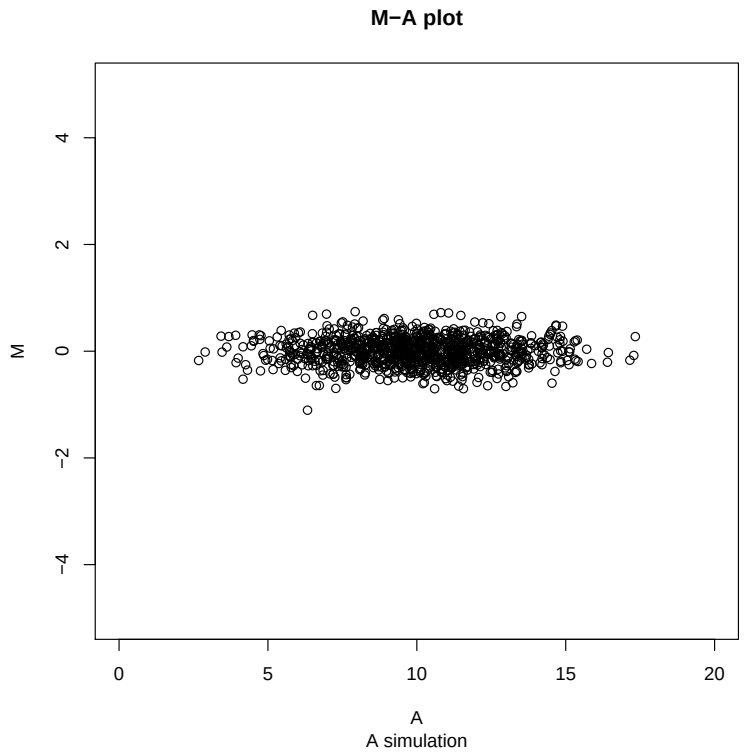
```
> plot(A,M, xlim=c(0,20), ylim=c(-5,5), main="M-A plot")
```



Plot

Indeed, there are a lot of parameters you can adjust.

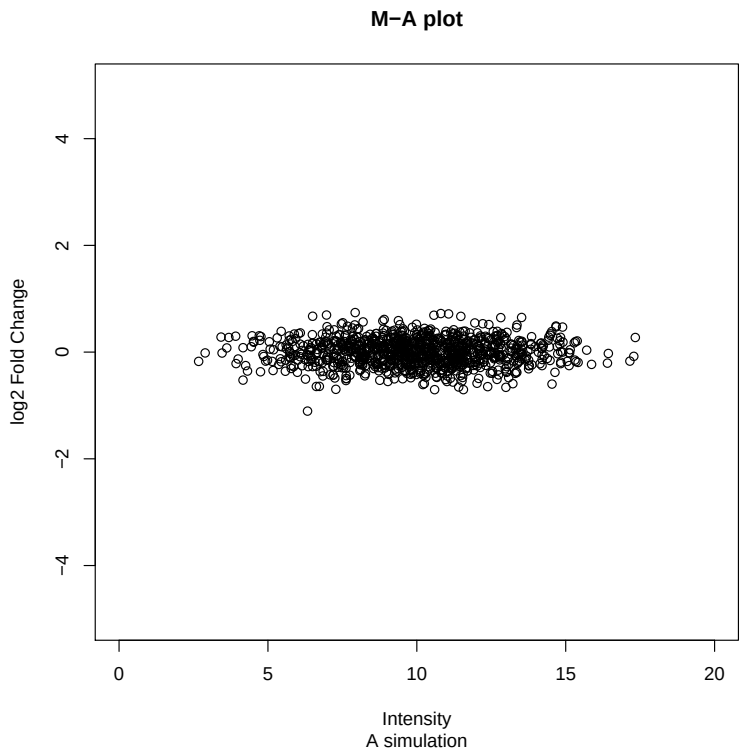
```
> plot(A,M, xlim=c(0,20), ylim=c(-5,5), main="M-A plot",
+ sub="A simulation")
```



Plot

Indeed, there are a lot of parameters you can adjust.

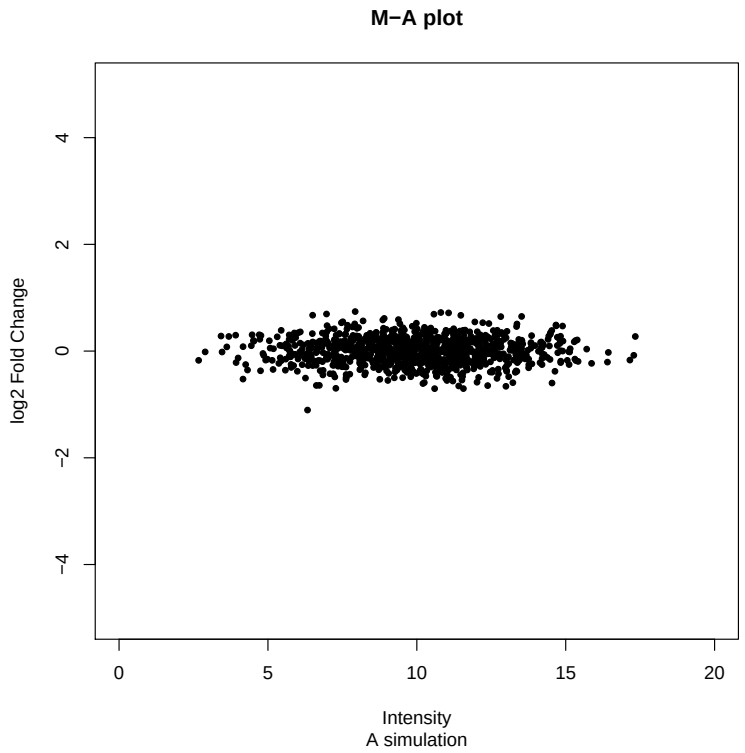
```
> plot(A,M, xlim=c(0,20), ylim=c(-5,5), main="M-A plot",
+ sub="A simulation",
+ xlab="Intensity", ylab="log2 Fold Change")
```



Plot

Indeed, there are a lot of parameters you can adjust.

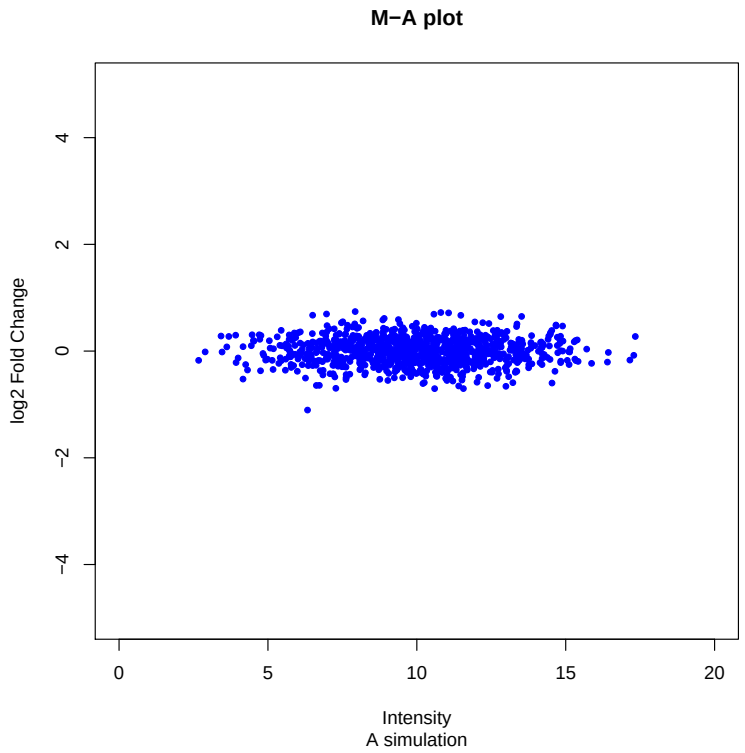
```
> plot(A,M, xlim=c(0,20), ylim=c(-5,5), main="M-A plot",
+ sub="A simulation",
+ xlab="Intensity", ylab="log2 Fold Change"),
+ pch=20)
```



Plot

Indeed, there are a lot of parameters you can adjust.

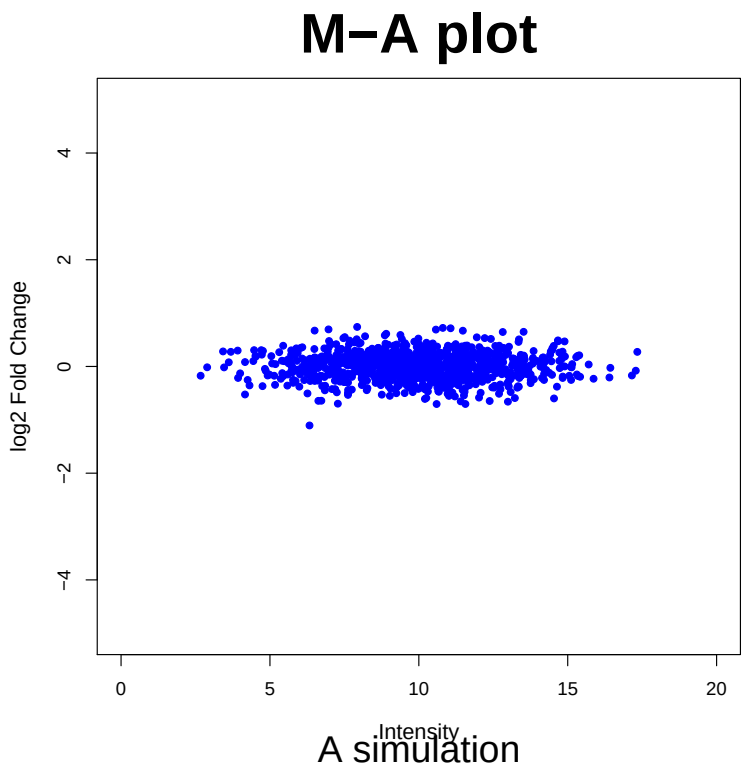
```
> plot(A,M, xlim=c(0,20), ylim=c(-5,5), main="M-A plot",
+ sub="A simulation",
+ xlab="Intensity", ylab="log2 Fold Change"),
+ pch=20, col="blue")
```



Plot

Indeed, there are a lot of parameters you can adjust.

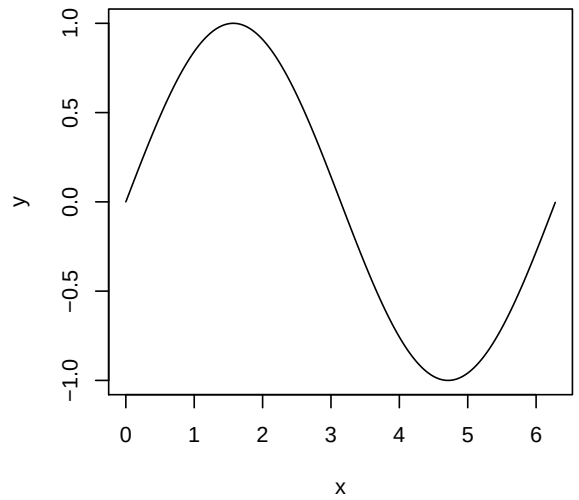
```
> plot(A,M, xlim=c(0,20), ylim=c(-5,5), main="M-A plot",
+ sub="A simulation",
+ xlab="Intensity", ylab="log2 Fold Change"),
+ pch=20, col="blue",
+ cex=1.2, cex.lab=1.2, cex.main=3, cex.sub=2)
```



Plot

Another example (draw lines instead of points):

```
> x<-seq(0, 2*pi, by=0.01)
> y<-sin(x)
> plot(x,y,type="l")
```



par()

You can also access and modify the list of graphics parameters for the current graphics device using the function par():

- par() returns a list of all graphics parameters.
- par(c("col","lty")) returns only the named graphics parameters.
- par(col=4,lty=2) sets the value of the named parameters, returns the old values as a list.

For example:

```
> par(c("col","lty"))
$col
[1] "black"
$lty
[1] "solid"
> oldpar<-par(col=4,lty=2)
> par(oldpar) ## restores the original setting
```

par()

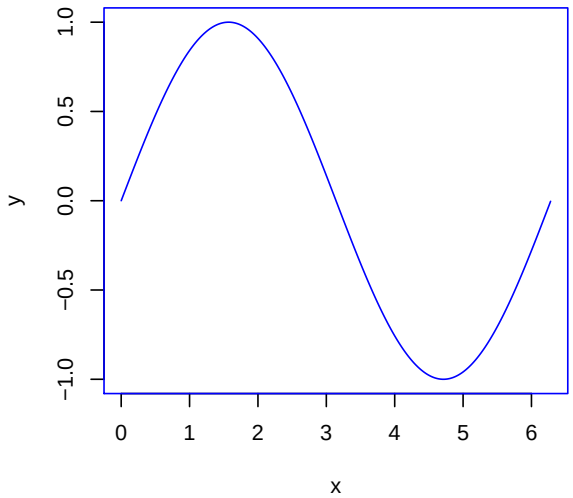
Differences between par() and setting parameters in plot() (and other high-level plotting functions):

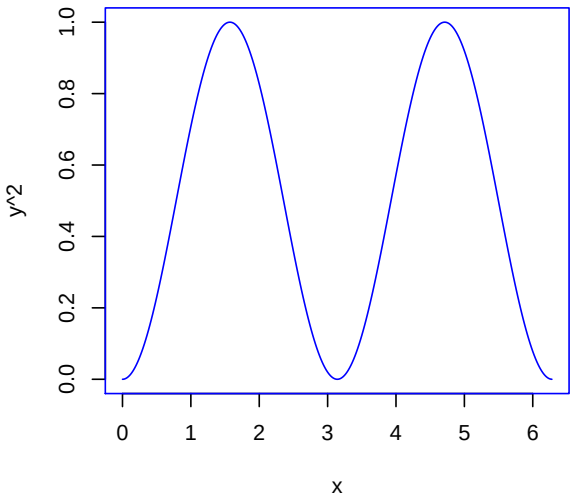
- Setting parameters using par() result in *permanent* changes of the values for the current graphics device.
- Parameter values set in plot() etc. are only effective when executing that particular command.

par()

For example:

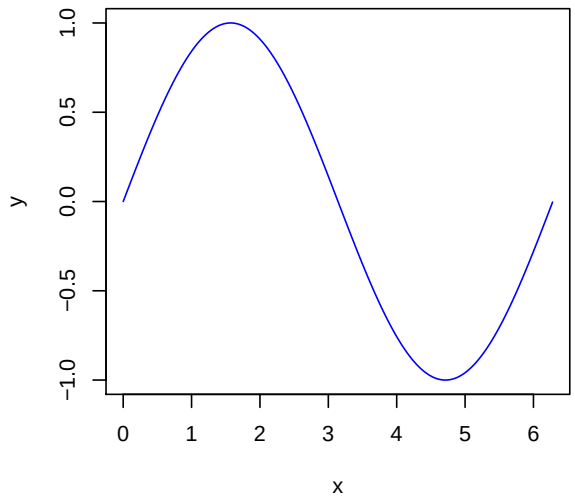
```
> x<-seq(0, 2*pi, by=0.01)
> y<-sin(x)

> oldpar<-par(col="blue")
> plot(x,y,type="l")
> plot(x,y^2,type="l")
> par(oldpar)
```

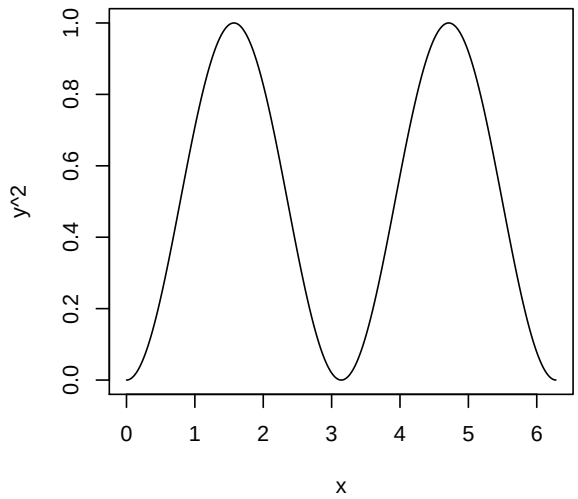




par()

```
> plot(x,y,type="l",col="blue")
> plot(x,y^2,type="l")
```





Types of plotting commands

plot(), hist(), etc. are *high-level* plotting functions. Sometimes, you want to add points or lines to an existing plot. To do this, you need *low-level* plotting functions.

In general, there are three types of plotting commands:

- **High-level**: create a new plot on the graphics device, with axes, labels, titles etc.
- **Low-level**: add information to an existing plot.
- **Interactive**: interactively add or extract information to or from an existing plot using a pointing device (e.g. mouse)

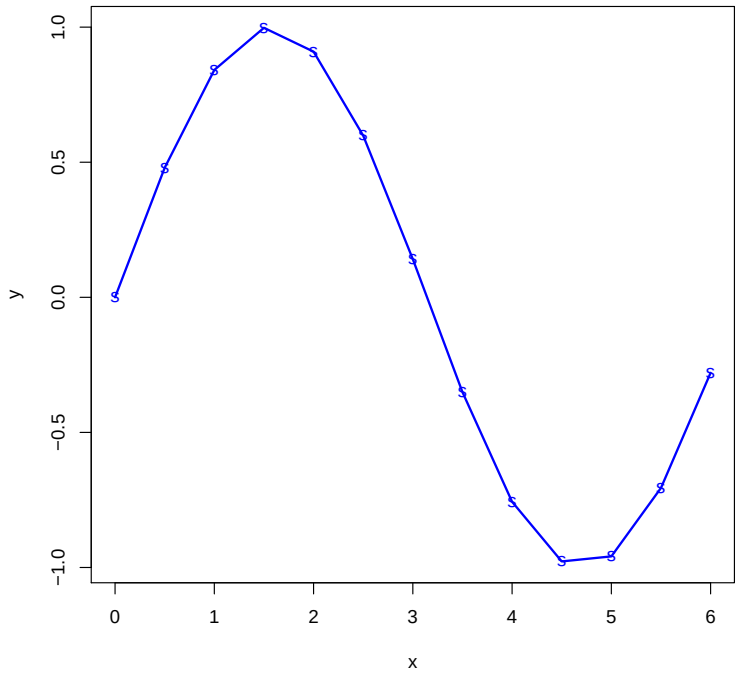
Low-level plotting functions

Examples are:

- points(): add points
- lines(): add connected lines
- text(): add texts
- abline(): add straight lines
- legend(): add legend
- title(): add titles
- ...

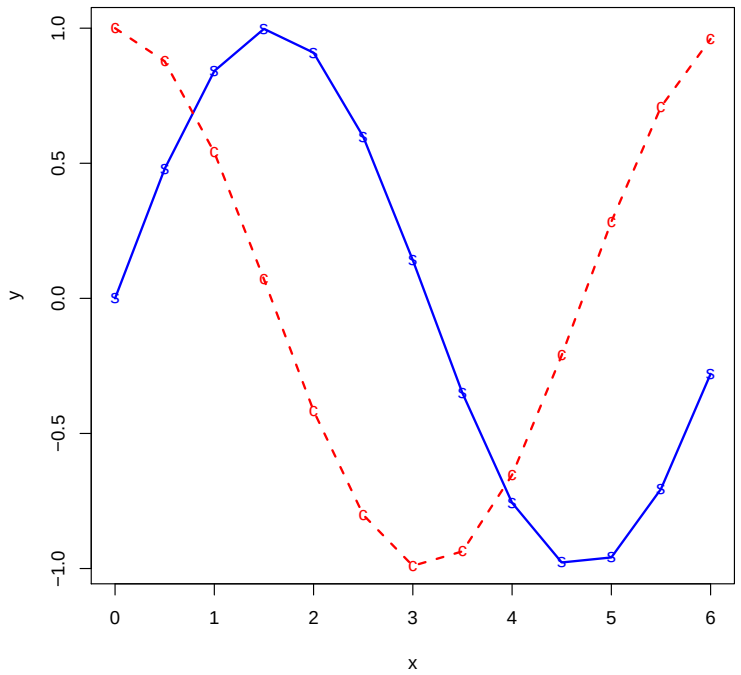
Low-level plotting functions

```
> x<-seq(0,2*pi,by=0.5)
> y<-sin(x)
> z<-cos(x)
> plot(x,y,type="o",col="blue",lwd=2,pch="s")
```



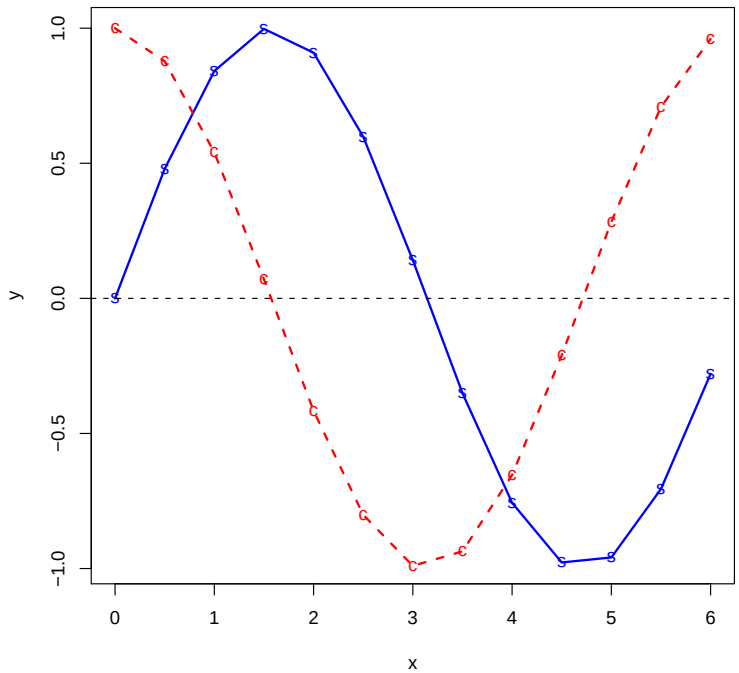
Low-level plotting functions

```
> lines(x,z,type="o",col="red",lty=2,lwd=2,pch="c")
```



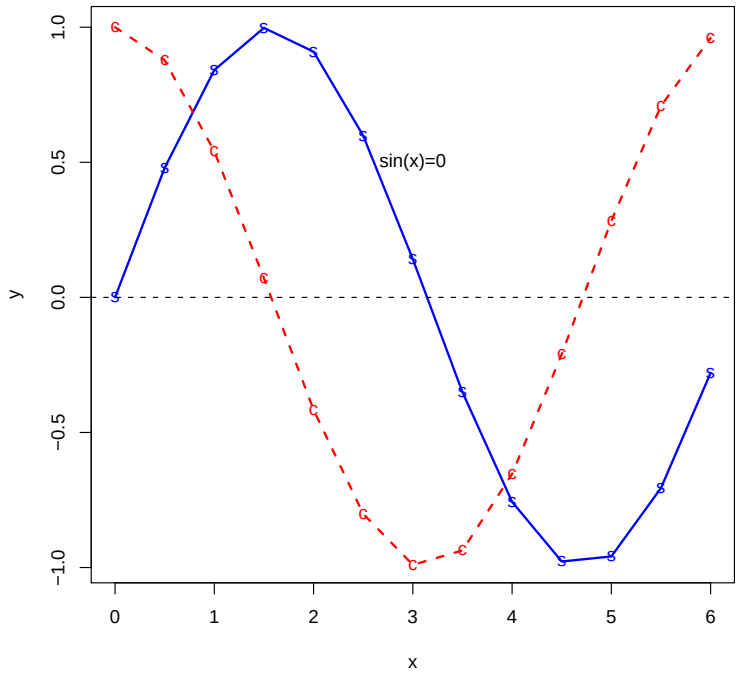
Low-level plotting functions

```
> abline(h=0, lty=2)
```



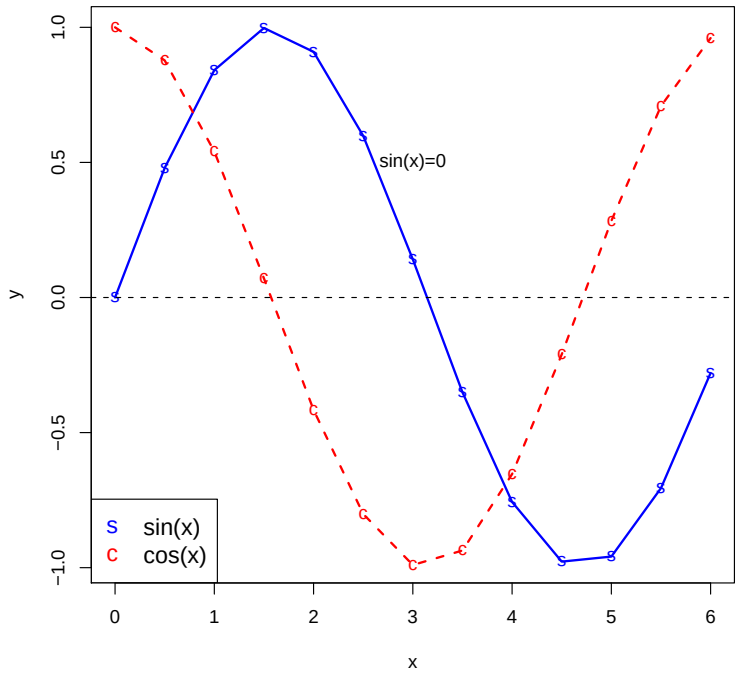
Low-level plotting functions

```
> text(3,0.5,"sin(x)=0")
```



Low-level plotting functions

```
> legend("bottomleft", cex=1.25,
+legend = c("sin(x)", "cos(x)"), pch = c("s", "c"),
+ col=c("blue","red"))
```



Interactive plotting functions

Try the locator() function:

```
> plot(x,y)
> text(locator(1),"y=sin(x)")
```

Exercise