# R: Graphics

140.776 Statistical Computing

August 21, 2011

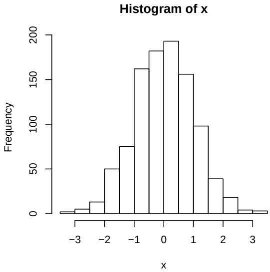
# Basic graphics

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
- ...

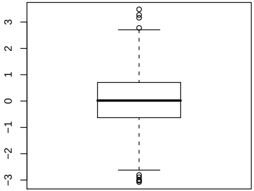
# Histogram

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



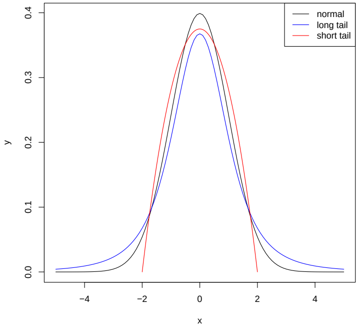
# Box plot

```
> boxplot(x)
```



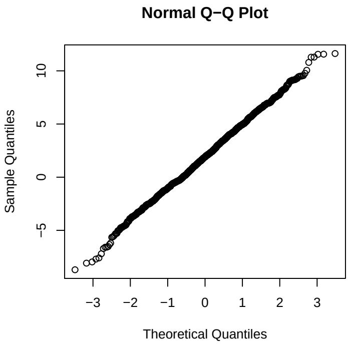
# Normal QQ plot

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



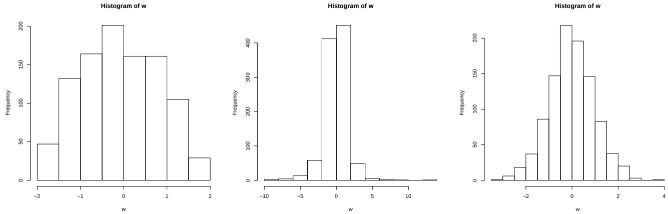
# Normal QQ plot

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



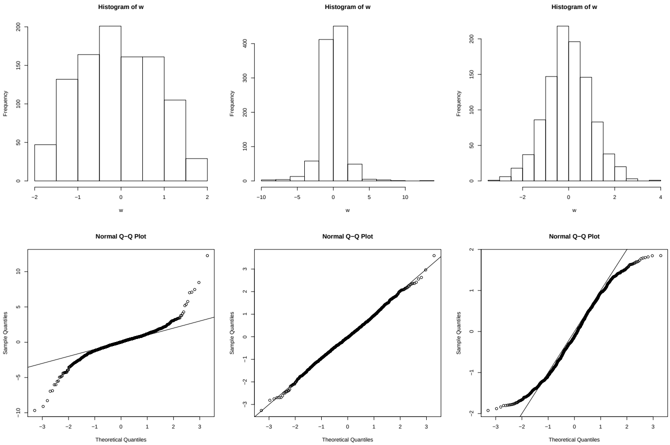
# Exercise

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



# Exercise

Find the corresponding normal QQ plot for each histogram:



# Normal QQ plot

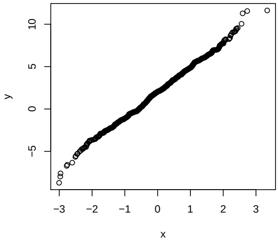
If you want to find out, you can try t-distribution (a long tail distribution)

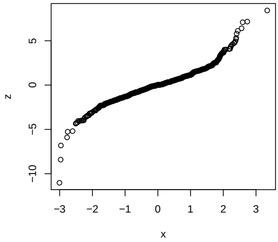
```
> w<-rt(1000,df=3)
> hist(w)
> qqnorm(w)
```

# 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)
```

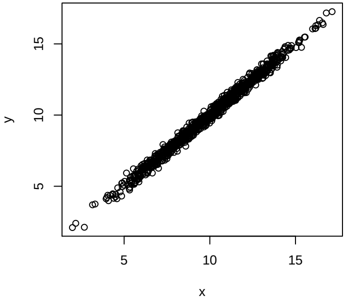


# Plot

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 $\epsilon \sim N(0, 0.25^2)$.

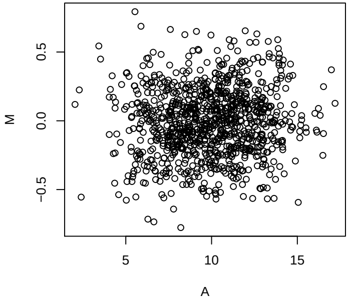
```
> x<-rnorm(1000,mean=10,sd=2.5)
> y<-x+rnorm(1000,mean=0,sd=0.25)
> plot(x,y)
```



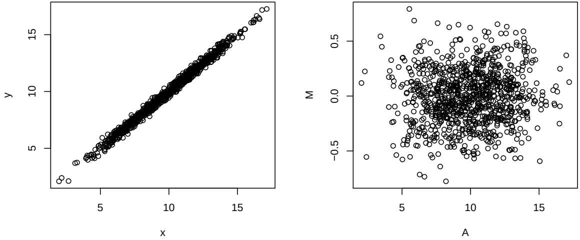
# Plot

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)
```



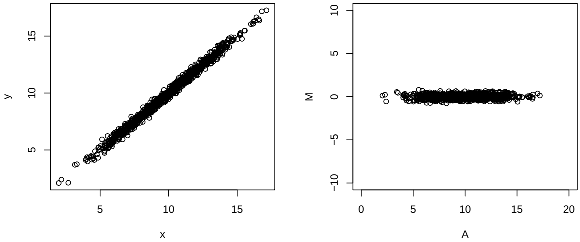
The MA plot looks quite different from the original plot. Why?



# 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:

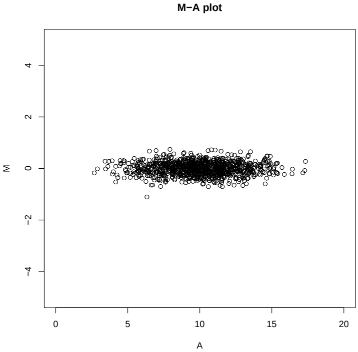
```
> plot(A,M, xlim=c(0,20), ylim=c(-10,10))
```



# Plot

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

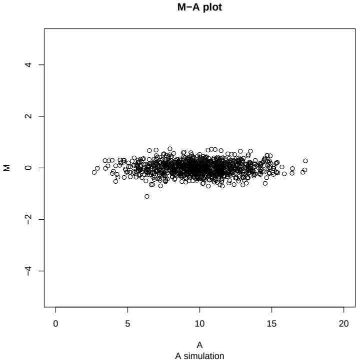
```
> plot(A,M, xlim=c(0,20), ylim=c(-5,5), main="M-A plot")
```



# Plot

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

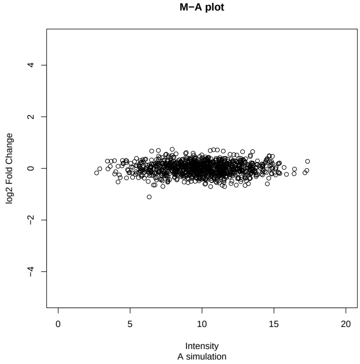
```
> plot(A,M, xlim=c(0,20), ylim=c(-5,5), main="M-A plot",
+ sub="A simulation")
```



# Plot

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

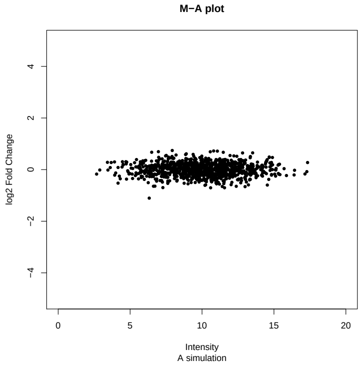
```
> plot(A,M, xlim=c(0,20), ylim=c(-5,5), main="M-A plot",
+ sub="A simulation",
+ xlab="Intensity", ylab="log2 Fold Change")
```



# Plot

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

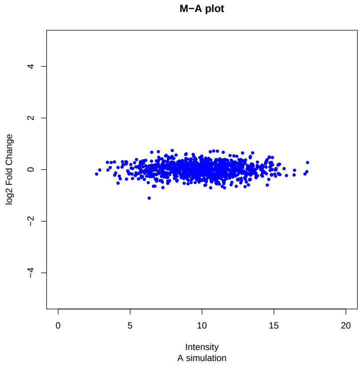
```
> plot(A,M, xlim=c(0,20), ylim=c(-5,5), main="M-A plot",
+ sub="A simulation",
+ xlab="Intensity", ylab="log2 Fold Change"),
+ pch=20)
```



# Plot

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

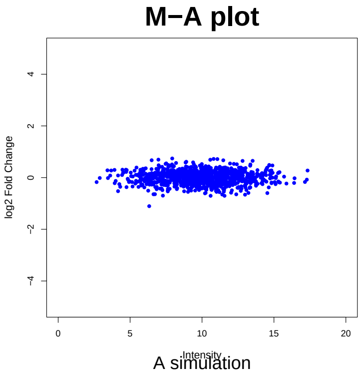
```
> plot(A,M, xlim=c(0,20), ylim=c(-5,5), main="M-A plot",
+ sub="A simulation",
+ xlab="Intensity", ylab="log2 Fold Change"),
+ pch=20, col="blue")
```



# Plot

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

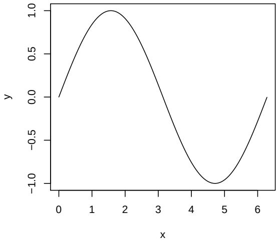
```
> plot(A,M, xlim=c(0,20), ylim=c(-5,5), main="M-A plot",
+ sub="A simulation",
+ xlab="Intensity", ylab="log2 Fold Change"),
+ pch=20, col="blue",
+ cex=1.2, cex.lab=1.2, cex.main=3, cex.sub=2)
```



# 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
```

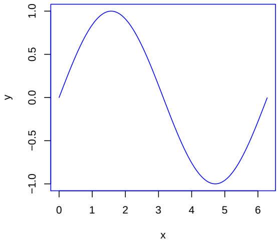
Differences between par() and setting parameters in plot() (and other high-level plotting functions):

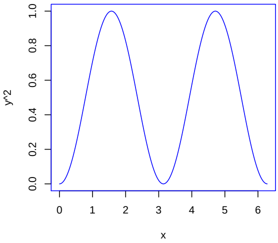
- Setting parameters using par() result in *permanent* changes of the values for the current graphics device.
- Parameter values set in plot() etc. are only effective when executing that particular command.

# par()

For example:

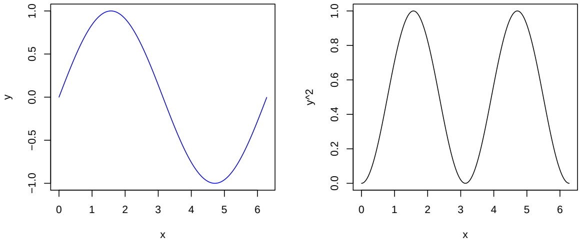
```
> x<-seq(0, 2*pi, by=0.01)
> y<-sin(x)

> 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)

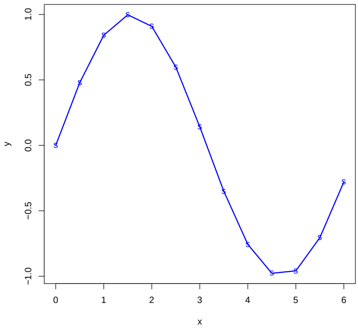
# Low-level plotting functions

Examples are:

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

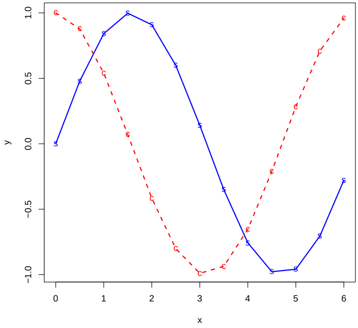
# Low-level plotting functions

```
> 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")
```



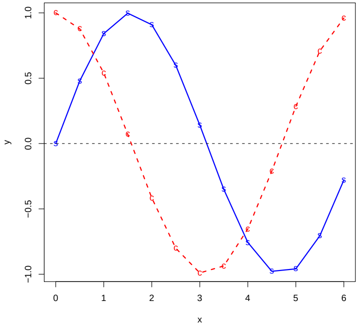
# Low-level plotting functions

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



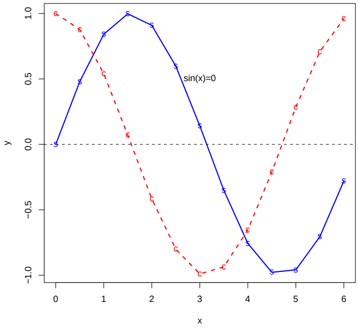
# Low-level plotting functions

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



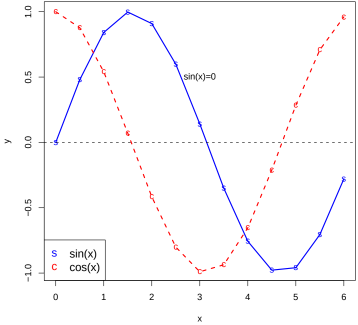
# Low-level plotting functions

```
> 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"))
```

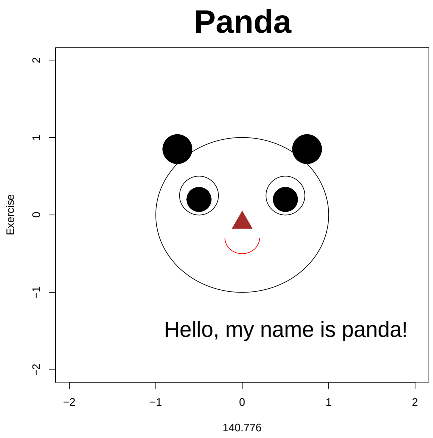


# Interactive plotting functions

Try the locator() function:

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

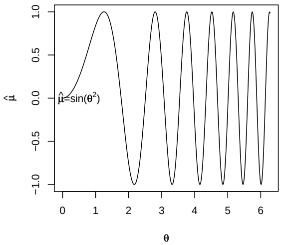
# Exercise



# Mathematical notations

You can show mathematical symbols in texts using the function expression(). Please use help(plotmath) to learn details.

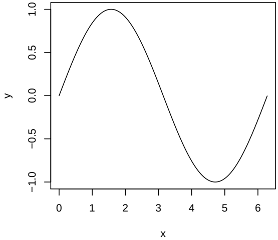
```
> x<-seq(0,2*pi,by=0.01); y<-sin(x^2)
> xtext<-expression(theta)
> ytext<-expression(hat(mu))
> mtext<-expression(paste(hat(mu),"=sin(",theta^2,")"))
> plot(x,y,type="l",xlab=xtext,ylab=ytext)
> text(0.5,0,mtext)
```

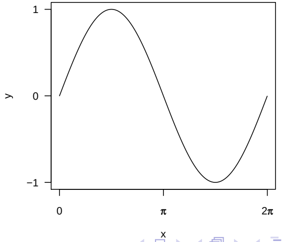


# Axes

Axis() can be used to set the axis line and tick marks:

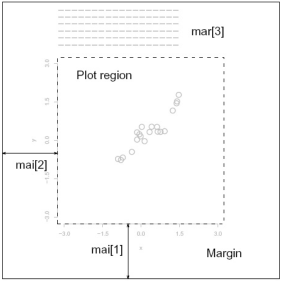
```
> x<-seq(0,2*pi,by=0.01); y<-sin(x)
> plot(x,y,type="l")
> plot(x,y,type="l",xaxt="n",yaxt="n")
> u<-2*pi*(0:2)/2
> axis(1,at=u,labels=c("0",expression(pi),
+ expression(paste("2", pi))))
> axis(2,at=c(-1,0,1),las=1)
```



# Margin

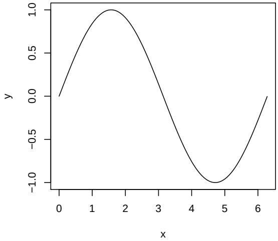
- mai sets margins measured by inches
- mar sets margins using text line as measurement unit

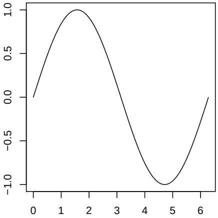



Venables and Smith, An introduction to R

# Margin

```
> plot(x,y,type="l")
> oldpar<-par(mai=c(0.5,0.5,1,1))
> plot(x,y,type="l")
> par(oldpar)
```

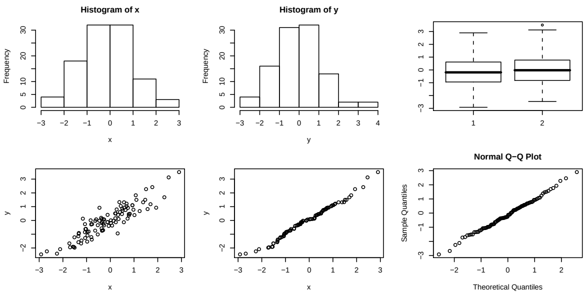


# Multiple figures

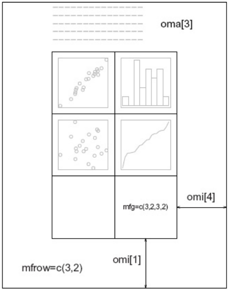
mfcol and mfrow allow you to create multiple figures on a single page. mfcol fills the subplots by column, and mfrow fills by row.

```
> x<-rnorm(100); y<-x+rnorm(100,sd=0.5)
> oldpar<-par(mfrow=c(2,3),oma=c(0,1,1,0),mar=c(4,4,3,2))
> hist(x); hist(y); boxplot(x,y)
> plot(x,y); qqplot(x,y); qqnorm(x)
> par(oldpar)
```



# Margin

- omi sets outer margins measured by inches
- oma sets outer margins using text line as measurement unit

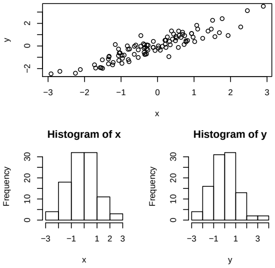



Venables and Smith, An introduction to R

# Multiple figures

layout() is another way to organize multiple figures:

```
> layout(matrix(c(1,1,2,3),2,2,byrow=TRUE))
> oldpar<-par(oma=c(0,1,1,0),mar=c(4,4,3,2))
> plot(x,y)
> hist(x); hist(y)
> par(oldpar)
```



# Save plots

You can save plots to files using pdf(), postscript(), png(), jpeg(), bmp(), tiff(), bitmap(). For example, the plot in the previous slide can be saved to a pdf file using the commands below:

```
> pdf("testplot.pdf",width=4, height=4, pointsize=10)
> layout(matrix(c(1,1,2,3),2,2,byrow=TRUE))
> oldpar<-par(oma=c(0,1,1,0),mar=c(4,4,3,2))
> plot(x,y)
> hist(x); hist(y)
> par(oldpar)
> dev.off()
```

# Graphics devices

R can generate graphics on many types of devices:

- X11(), windows(), quartz() are default for UNIX, Windows, and Mac OS X respectively.
- pdf(), png(), jpeg(), etc.

# Graphics devices

To plot to a specific device, usually you need to go through the following steps:

1. Open a device driver, e.g. pdf()
2. Make a plot
3. Close the device driver, e.g. dev.off()

# Graphics devices

Sometimes you want to work with multiple devices and copy figures from one device to the others. To do this, read help documents for these functions:

- dev.list()
- dev.prev(), dev.next()
- dev.set()
- dev.copy(), dev.print()
- graphics.off()
- ...

# Plotting packages

There are several R packages for plotting:

- **graphics**: contains functions for the "base" graphing systems, including plot, hist, etc. This is the package we've talked about so far.
- **lattice**: contains code for producing Trellis graphics, including xyplot, bwplot, levelplot etc.
- **grid**: implements a different graphing system independent of "base"; the **lattice** package builds on top of **grid**; we seldom call functions from the **grid** package directly.
- **grDevices**: contains all the code implementing the various graphics devices, including X11, PDF, PostScript, PNG, etc.

# Lattice functions

This package contains a lot of functions:

- xyplot: scatterplots
- bwplot: box-and-whisker plots
- histogram
- stripplot: like a boxplot but with acutal points
- . . .

google to learn, example, bwplot function in r
ftp://cran.r-project.org/pub/R/web/packages/tigerstats/vignettes/bwplot.html

# Lattice functions

Lattice functions usually take a formula as the first argument: $z \sim x \mid y$. It means plotting z vs. x conditional on y.

```
> library(lattice)
> x<-rnorm(100)
> y<-rbinom(100,5,0.5)
> z<-x+2*y+0.5*x*y+rnorm(100,sd=0.5)
> xyplot(z~x | y)
```