

## Lecture 5 - Plots and lines

### Understanding magic

Let us look at the following curious thing:

```
> x=rnorm(100)
> y=rnorm(100,sd=0.1)+x
> k=ks.test(x,y)
> k
```

```
Two-sample Kolmogorov-Smirnov test
```

```
data: x and y
D = 0.05, p-value = 0.9996
alternative hypothesis: two.sided
```

```
> print(k)
```

```
Two-sample Kolmogorov-Smirnov test
```

```
data: x and y
D = 0.38, p-value = 1.071e-06
alternative hypothesis: two.sided
```

```
>
```

If everything in R is vectors or lists, why is the result of the KS test printed like this?

Here is another example:

```
> s=summary(x)
> s
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-2.22700	-0.57190	-0.08799	0.01895	0.72390	2.04100

```
>
```

The magic is the following:

Even though things are always vectors or lists, they can have a "class". To see the class of an object, we can use the `class()` function:

```
> class(s)
[1] "table"
> class(k)
```

```
[1] "hstest"
```

```
>
```

And now the trick:

When we say `print(k)`, the actual function that is called is `print.hstest()` for `k`, and `print.table()` for `s`.

```
> print.hstest(k)
```

```
Two-sample Kolmogorov-Smirnov test
```

```
data: x and y
D = 0.38, p-value = 1.071e-06
alternative hypothesis: two.sided
```

```
> print.table(s)
```

```
   Min.  1st Qu.  Median    Mean 3rd Qu.    Max.
-2.22700 -0.57190 -0.08799  0.01895  0.72390  2.04100
```

```
> print.hstest(s)
```

```
Error in strsplit(x, as.character(split), as.logical(extended)) :
  non-character argument in strsplit()
```

```
> print
```

```
function (x, ...)
UseMethod("print")
<environment: namespace:base>
```

```
>
```

All this is not very important, yet.

It only matters if we want to understand what function is called.

```
> help(print.hstest)
```

```
help() for print.hstest is shown in browser netscape ...
Use      help( print.hstest , htmlhelp=FALSE)
or      options(htmlhelp = FALSE)
to revert.
Warning message:
Using non-linked HTML file: style sheet and hyperlinks may be incorrect in:
help(print.hstest)
```

```
>
```

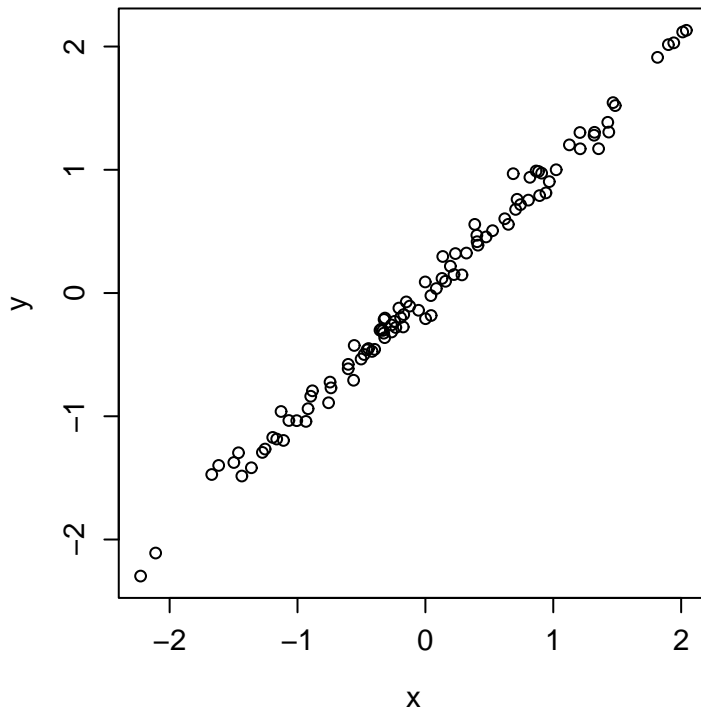
Another function that depends on its input is `plot()`.

```
> plot
```

```
awk: cmd. line:1: warning: escape sequence '\.' treated as plain '.'
```

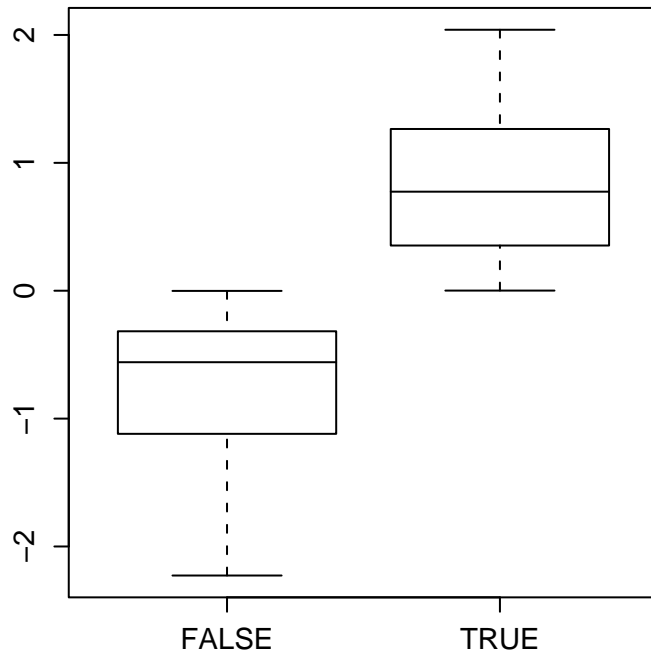
```
function (x, y, ...)  
{  
  if (is.null(attr(x, "class")) && is.function(x)) {  
    nms <- names(list(...))  
    if (missing(y))  
      y <- {  
        if (!"from" %in% nms)  
          0  
        else if (!"to" %in% nms)  
          1  
        else if (!"xlim" %in% nms)  
          NULL  
      }  
    if ("ylab" %in% nms)  
      plot.function(x, y, ...)  
    else plot.function(x, y, ylab = paste(deparse(substitute(x)),  
      "(x)"), ...)  
  }  
  else UseMethod("plot")  
}  
<environment: namespace:base>
```

```
> plot(x,y);v()
```

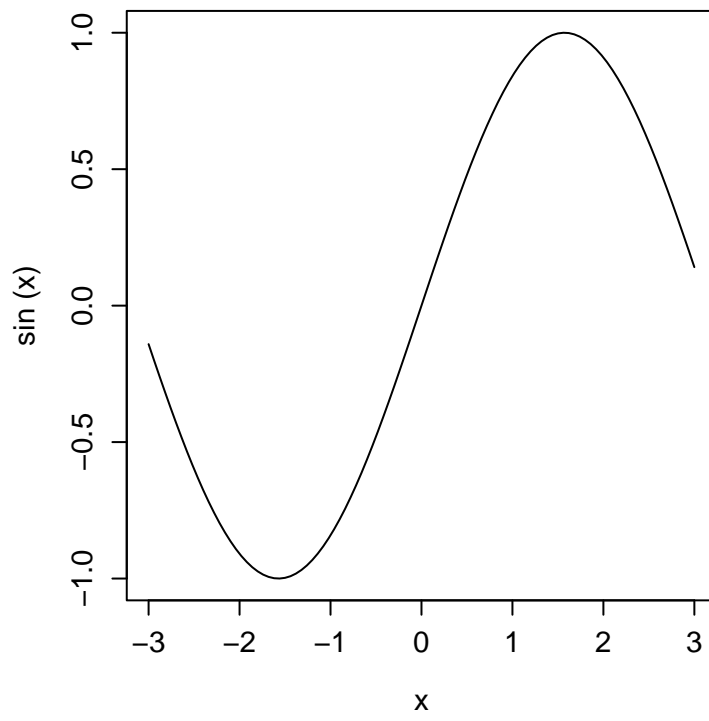


```
> z=as.factor( x>0 )
```

```
> plot(z,x);v()
```



```
> plot(sin,from=-3,to=3);v()
```



>

And now we can understand how to bring up help for these functions:

```
> class(sin)
```

```
[1] "function"
```

```
> help(plot.function)
```

```
help() for plot.function is shown in browser netscape ...
```

```
Use      help( plot.function , htmlhelp=FALSE)
```

```
or      options(htmlhelp = FALSE)
```

```
to revert.
```

```
Warning message:
```

```
Using non-linked HTML file: style sheet and hyperlinks may be incorrect in:
```

```
help(plot.function)
```

```
> class(z)
```

```
awk: cmd. line:1: warning: escape sequence '\.' treated as plain '.'
```

```
[1] "factor"
```

```
> help(plot.factor)
```

```
help() for plot.factor is shown in browser netscape ...
```

```
Use      help( plot.factor , htmlhelp=FALSE)
```

```
or      options(htmlhelp = FALSE)
```

```
to revert.
```

```
Warning message:
```

```
Using non-linked HTML file: style sheet and hyperlinks may be incorrect in:
```

```
help(plot.factor)
```

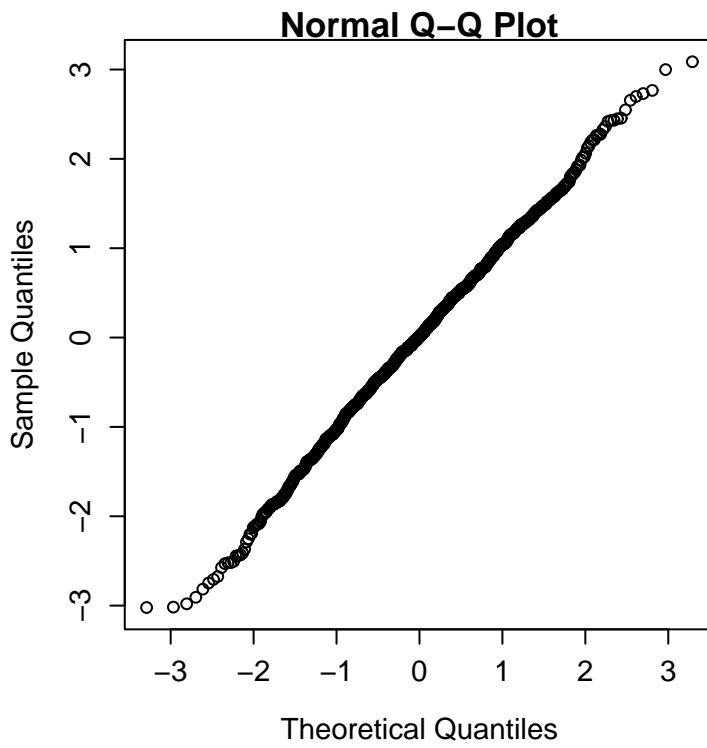
## Some plot commands we already encountered

qqplot

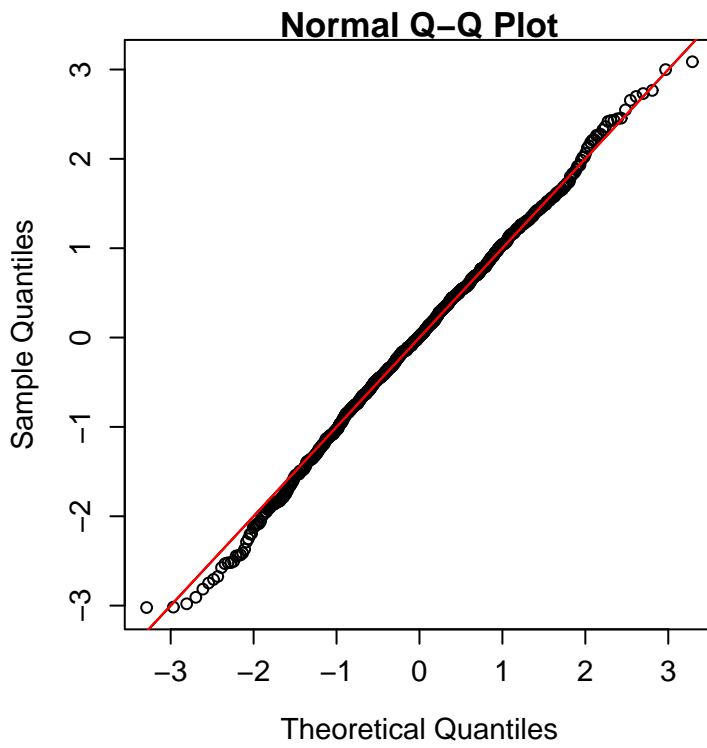
Last time we learned about qqplot and qqnorm. Let us see how to interpret them:

```
> x=rnorm(1000)
```

```
> qqnorm(x);v()
```



```
> abline(0,1,col="red");v()
```



Warning message:  
parameter "color" couldn't be set in high-level plot() function

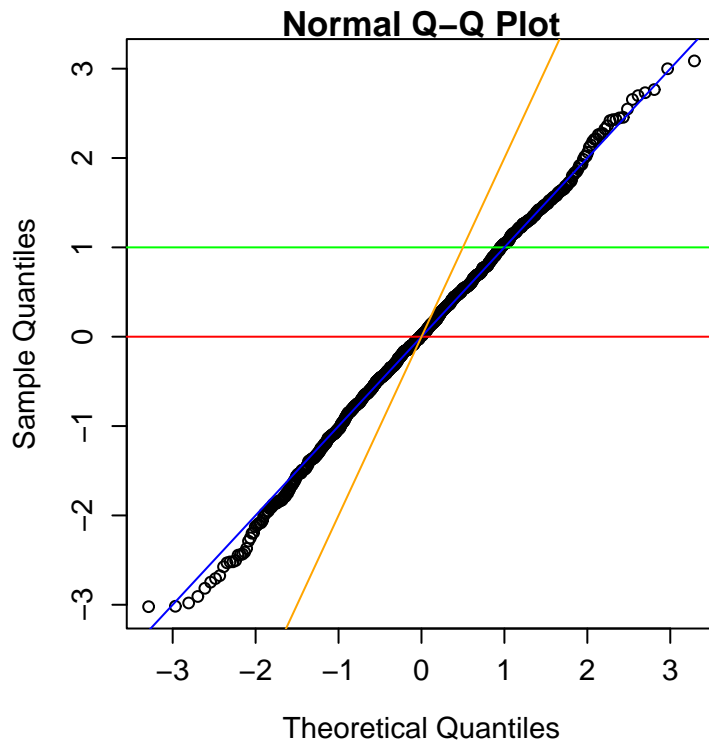
>

abline adds a line to the plot. It has two parameters: slope and intercept.

The col= parameter to plots specifies color.

```
> qqnorm(x);abline(0,0,col="red");abline(1,0,col="green");abline(0,1,col="blue")  
abline(0,2,col="orange");v()
```

>



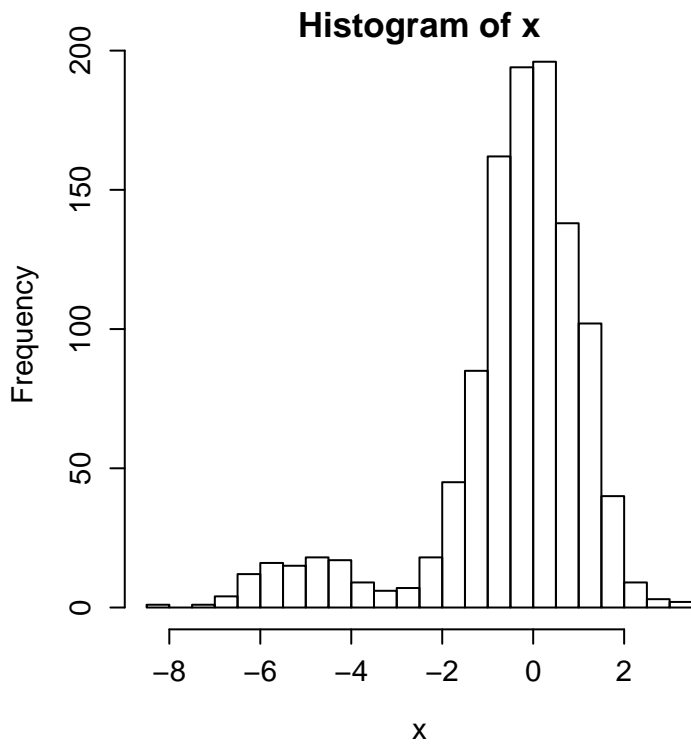
>

>

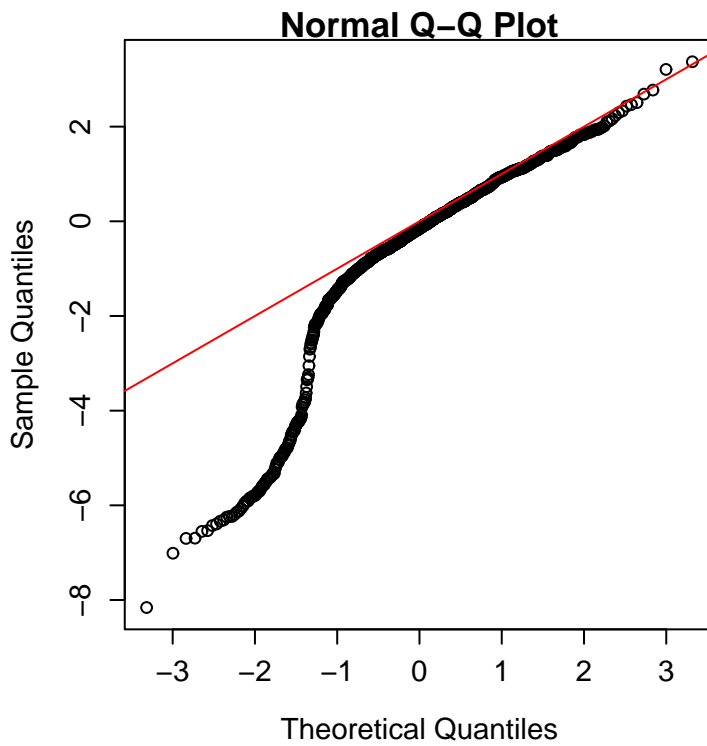
Let us add points on the right to the normal:

```
> x=c( rnorm(1000), rnorm(100,mean=-5) )
```

```
> hist(x,br=30);v()
```



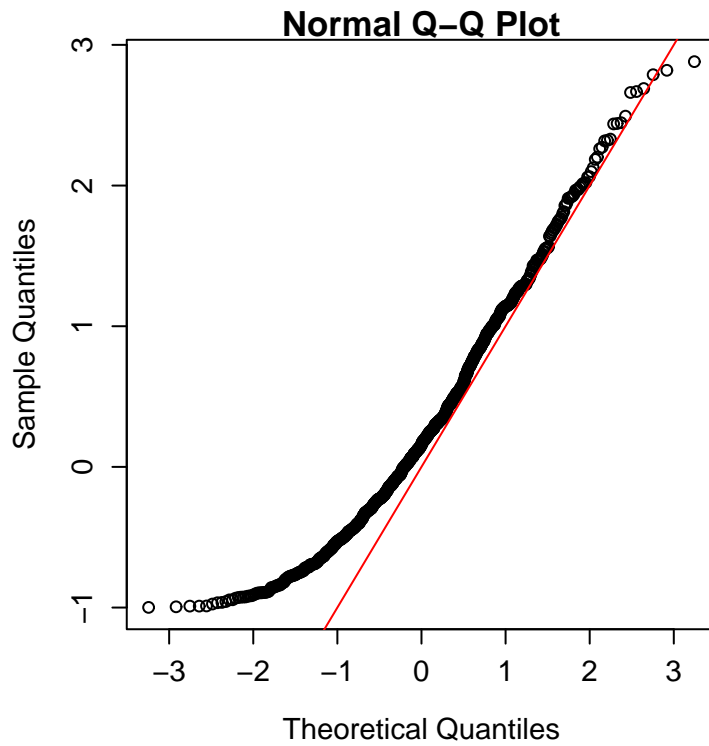
```
> qqnorm(x);abline(0,1,col=2);v()
```



>

If we have more points on the left than we should have, then quantiles occur earlier, and thus the points are low than they should be.

```
> x=rnorm(1000)
> x=x[ x>-1 ]
> qqnorm(x); abline(0,1,col=2);v()
```

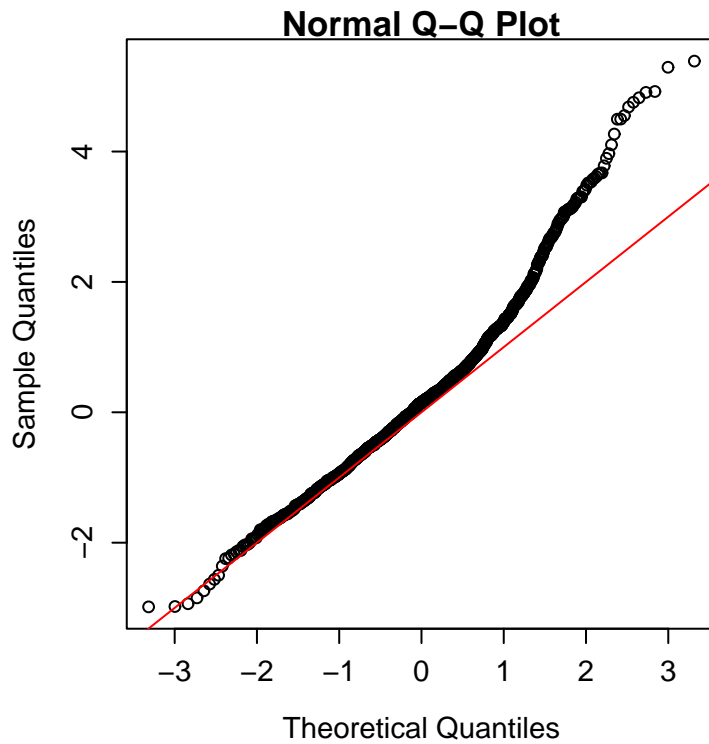


>

If we don't have enough points on the left, quantiles happen later, and the points move up.

Now let us add points on the right:

```
> x=c( rnorm(1000), rnorm(100,mean=3) )  
> qqnorm(x);abline(0,1,col=2);v()
```

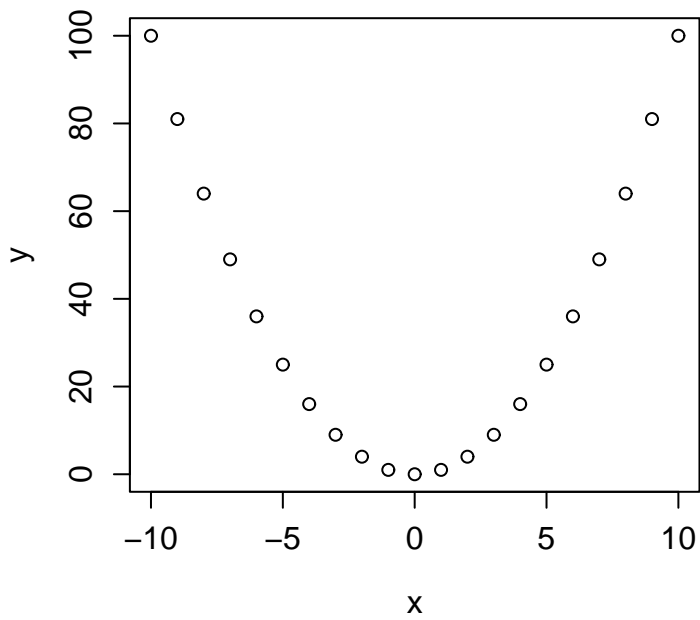


>

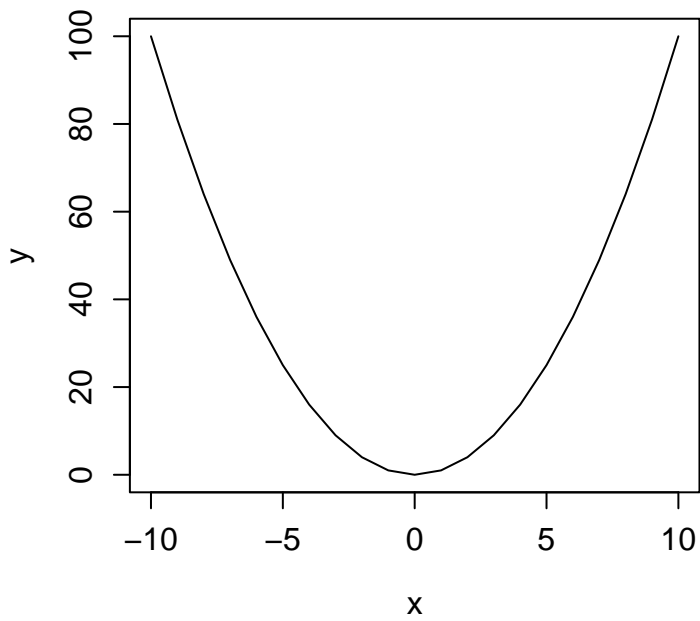
If we have too many points on the right, then the last quantiles occur too late, and the points move up.

plot

```
> x=-10:10 ; y= x ^ 2  
> plot(x,y) ; v()
```



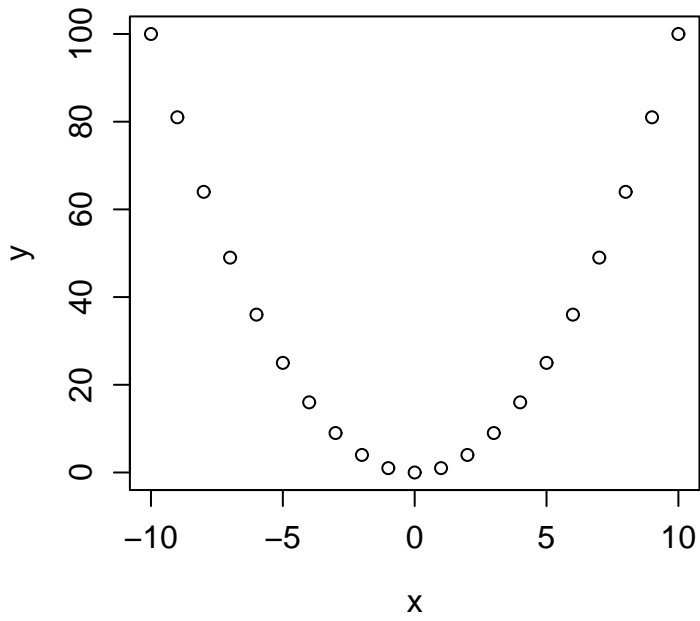
```
> plot(x,y,type="l");v()
```



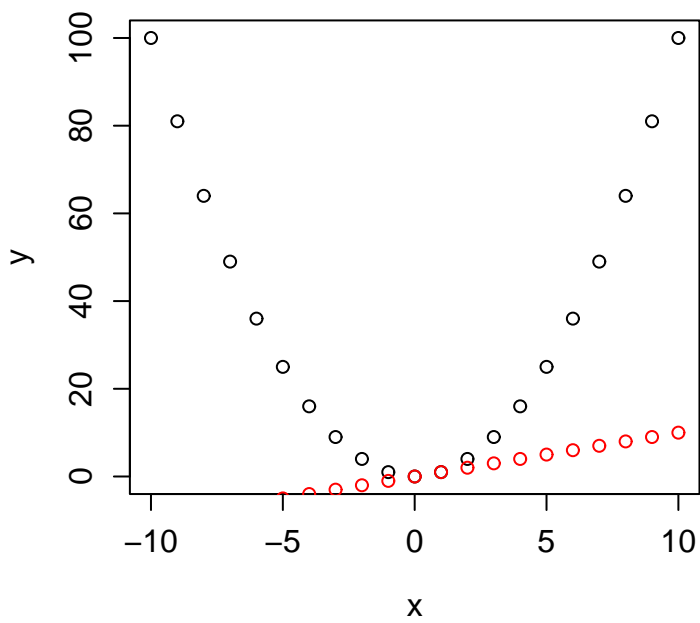
```
>
```

We can also add to an existing plot

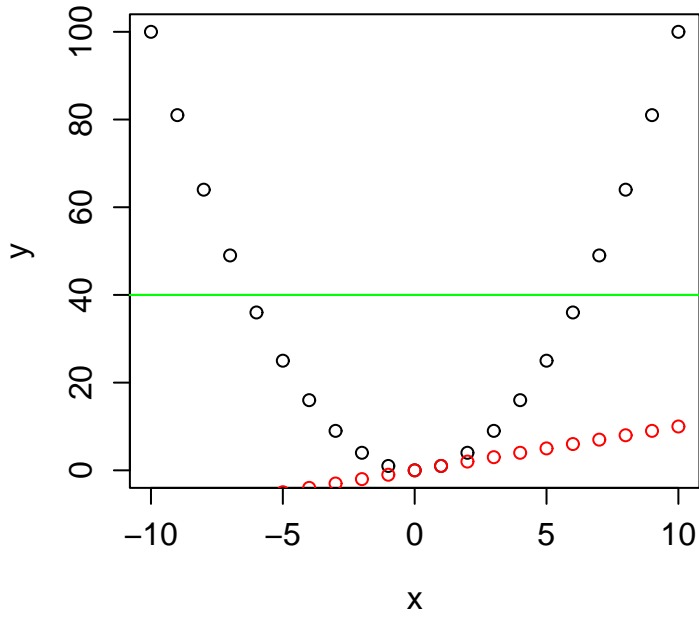
```
> plot(x,y);v()
```



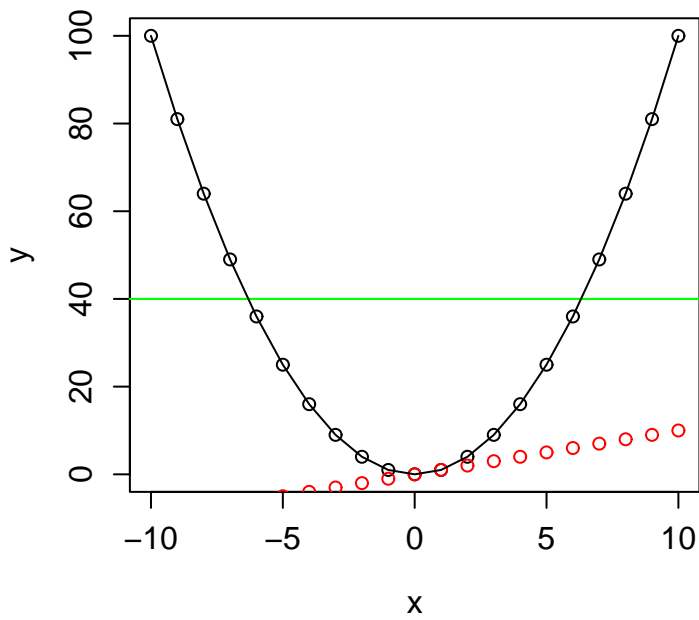
```
> points(x,x,col=2);v()
```



```
> abline(40,0,col="green");v()
```

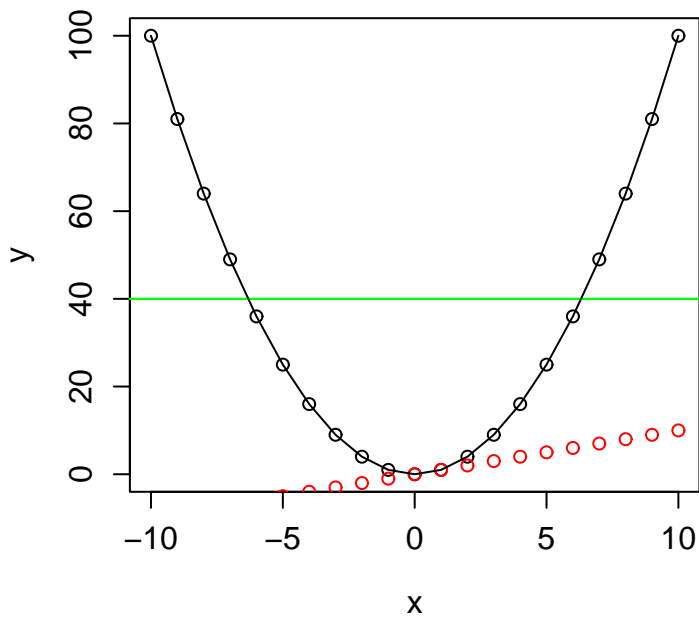


```
> lines(x,y);v()
```



```
> title("A nice plot");v()
```

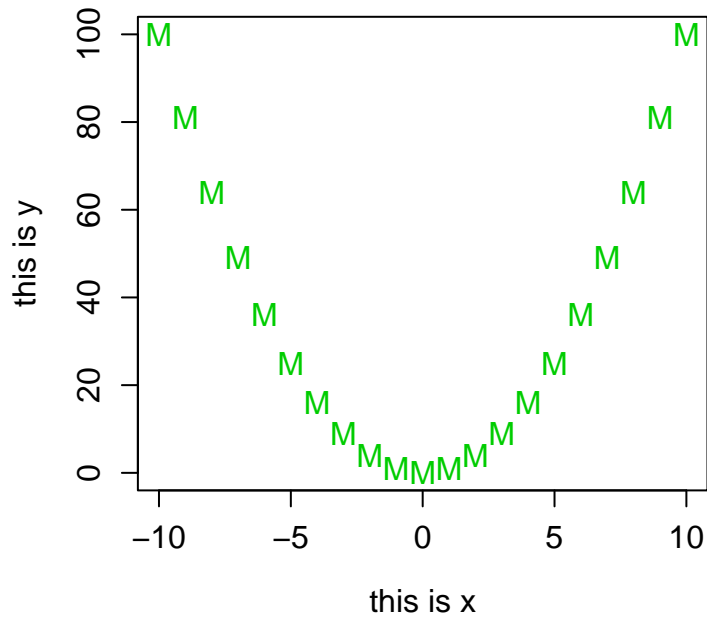
## A nice plot



```
> plot(x,y,pch="M",  
      col=3, xlab="this is x", ylab= "this is y",  
      main="A very nice plot" );v()
```

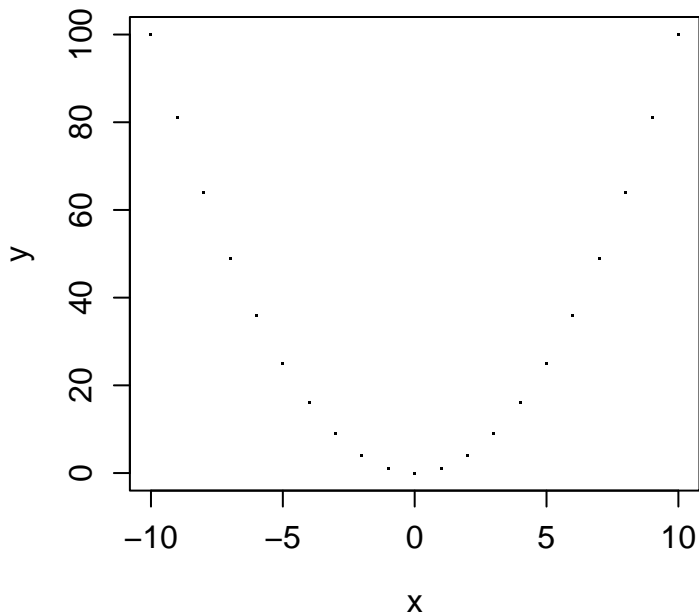
++

## A very nice plot



The most useful pch is pch="."

```
> plot(x,y,pch=".");v()
```



```
>
```

## Saving a plot to a file

We have a couple of options for saving a plot to a file:

One is saving as postscript. For others, look at `help(Devices)`.

First we specify the device, and the file:

```
> postscript(file="graph1.eps",width=5,height=5,horizontal=F,onefile=F)
```

```
>
```

Then we do all the plot commands:

```
> plot(sin,-3,3)
> abline(-1,0,col=2); abline(1,0,col=3)
>
```

Finally we close the device;

```
> dev.off()
```

```
  X11
    2
```

```
>
```

### Putting several plots into one

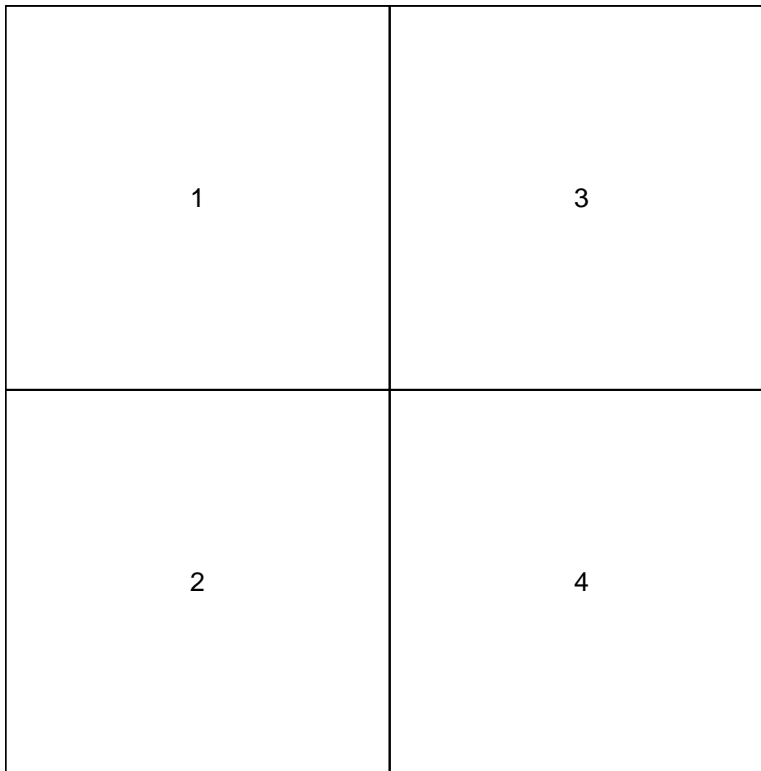
```
> two.by.two=matrix( 1:4, 2, 2)
```

```
> two.by.two
```

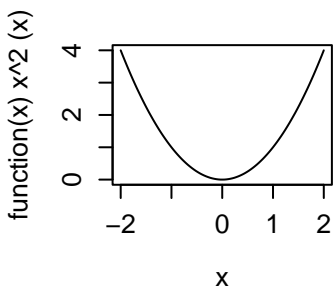
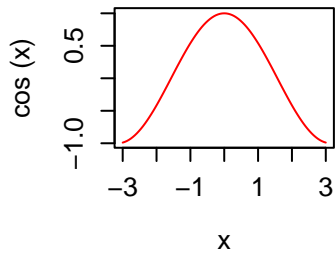
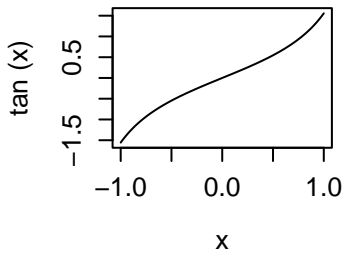
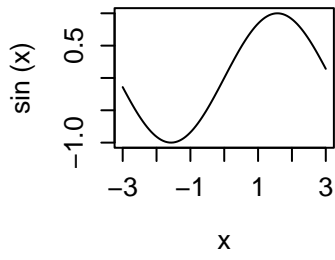
```
      [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
> layout(two.by.two)
```

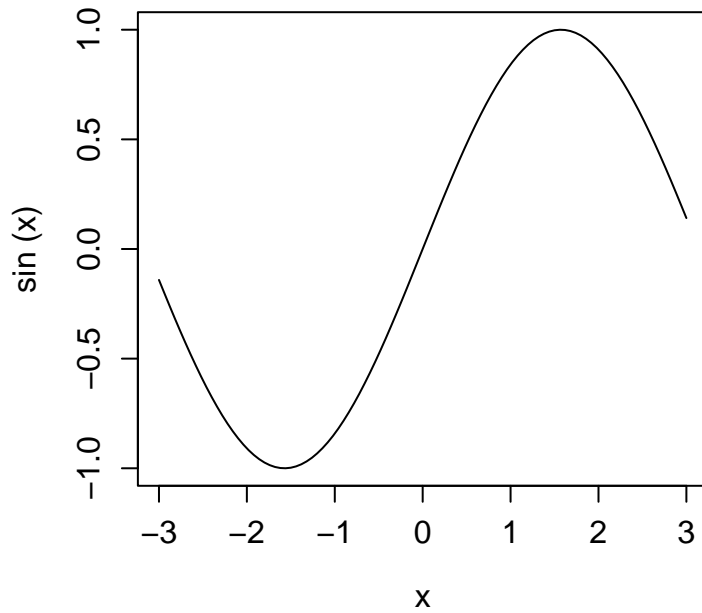
```
> layout.show(4);v()
```



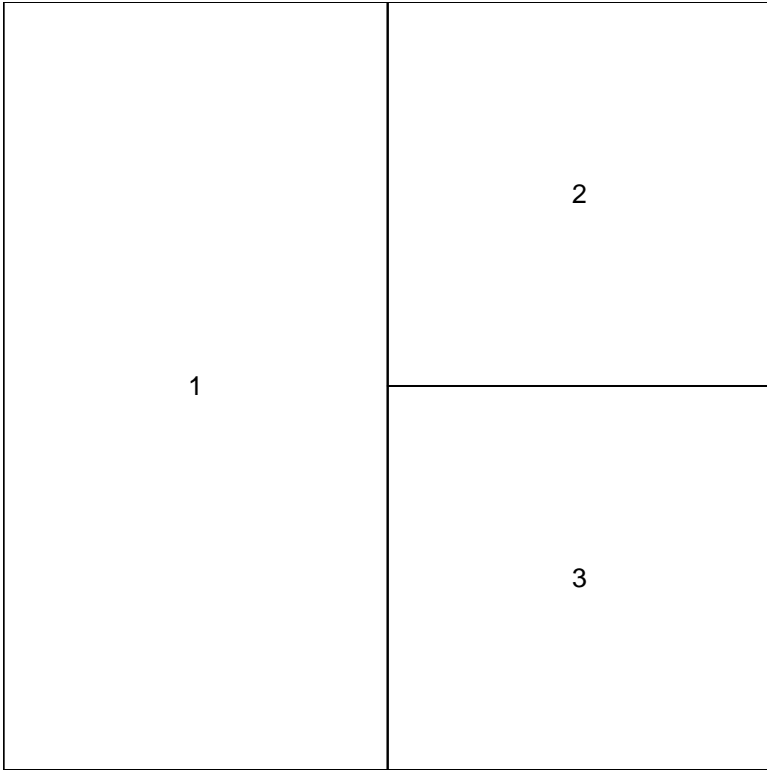
```
> plot(sin,-3,3)
> plot(cos,-3,3,col=2)
> plot(tan,-1,1)
> plot(function(x) x^2, -2, 2)
> v()
```



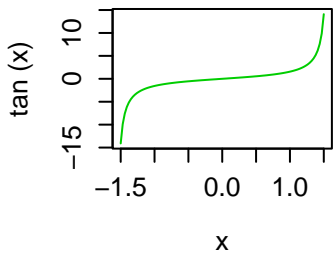
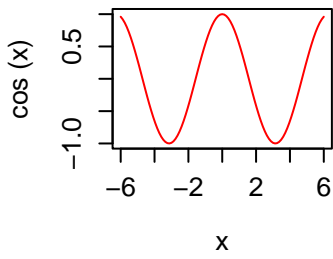
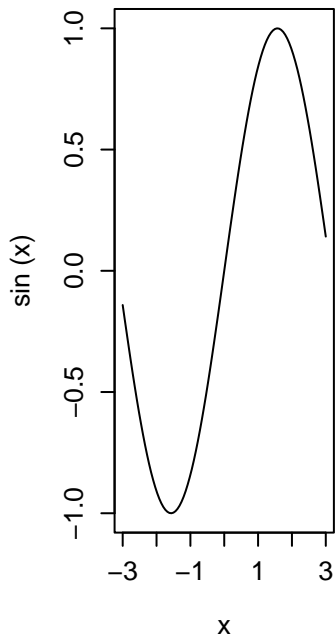
```
> layout(1)
> plot(sin,-3,3);v()
```



```
> l=matrix( c(1,1,2,3),2,2)
> l
      [,1] [,2]
[1,]    1    2
[2,]    1    3
> layout(1); layout.show(3); v()
```



```
> plot(sin,-3,3); plot(cos,-6,6,col=2); plot(tan,-1.5,1.5,col=3)  
> v()
```



>

why are the figures so small?

R reserves space for the title, the axes, the labels, and so on. The space is proportional to the character size, and that size does not change when we split the plot.

> `par("mar")`

```
[1] 5.1 4.1 4.1 2.1
```

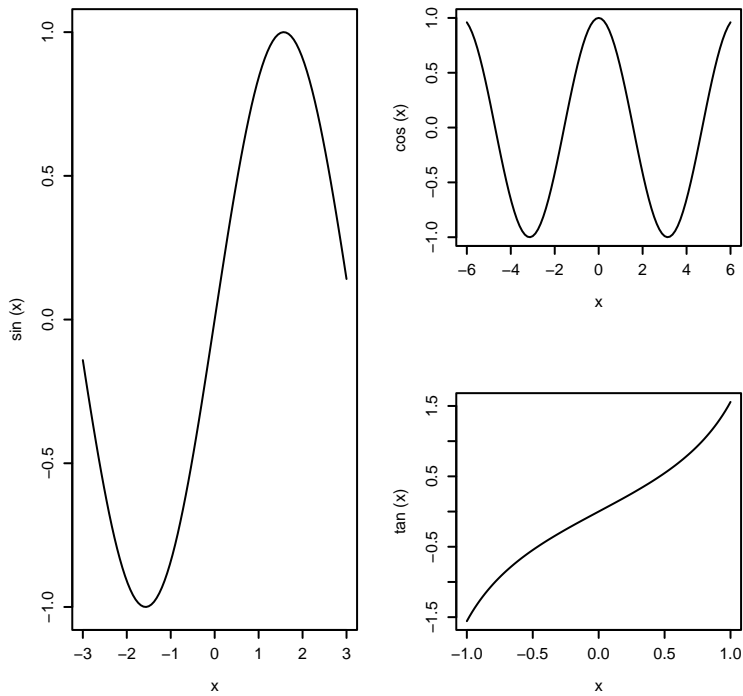
>

above we see how much space is kept for the margin, in lines. top, bottom, left, right.

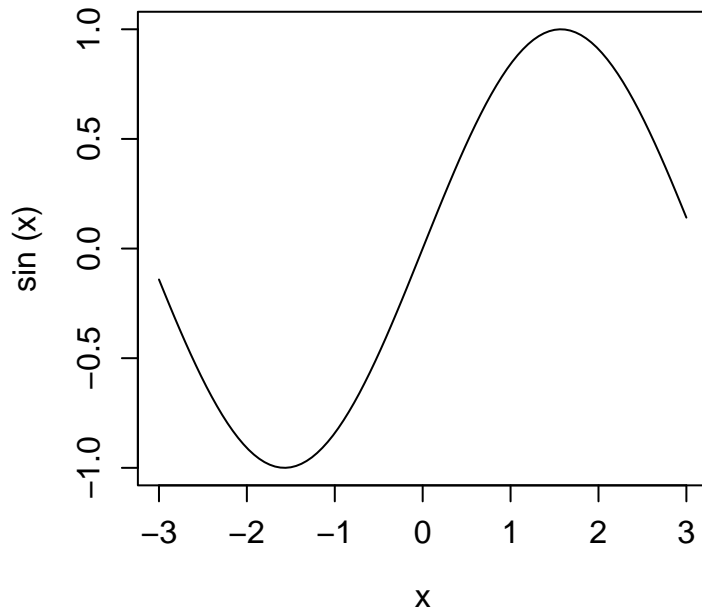
So, if we want to keep figures looking the same, we should scale the text:

> `par(cex=0.5)`

> `plot(sin,-3,3); plot(cos,-6,6); plot(tan,-1,1); v()`



```
> par("cex")
[1] 0.5
> layout(1)
> plot(sin,-3,3); v()
```



```
> par("cex")
[1] 1
```

```
>
```

The function `layout()` changes the character size!

## Linear regression

```
> a=read.table("data/regression.txt",sep="\t",head=T)
> a
```

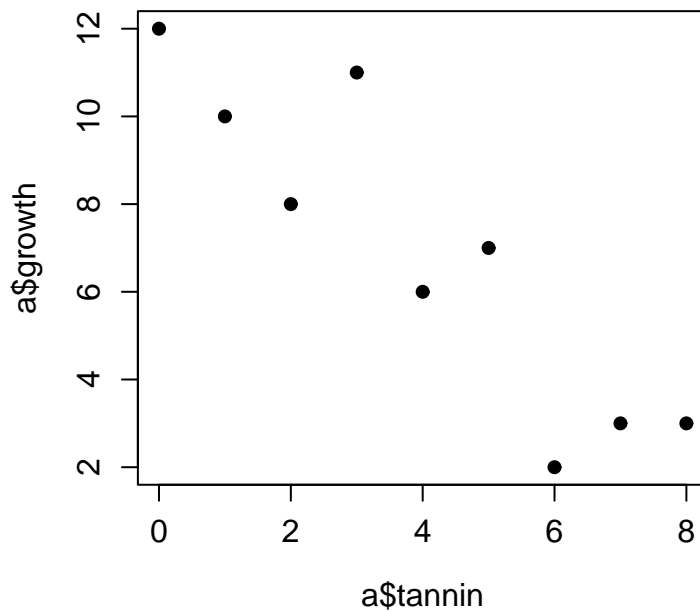
```
  growth tannin
1      12      0
```

```

2     10     1
3     8      2
4     11     3
5     6      4
6     7      5
7     2      6
8     3      7
9     3      8

```

```
> plot(a$tannin,a$growth,pch=16);v()
```



```
>
```

The growth of the caterpillar seems to depend on the tannin content.

we can try to fit a linear regression.

```
> regression=lm( growth ~ tannin, data = a )
> regression
```

Call:

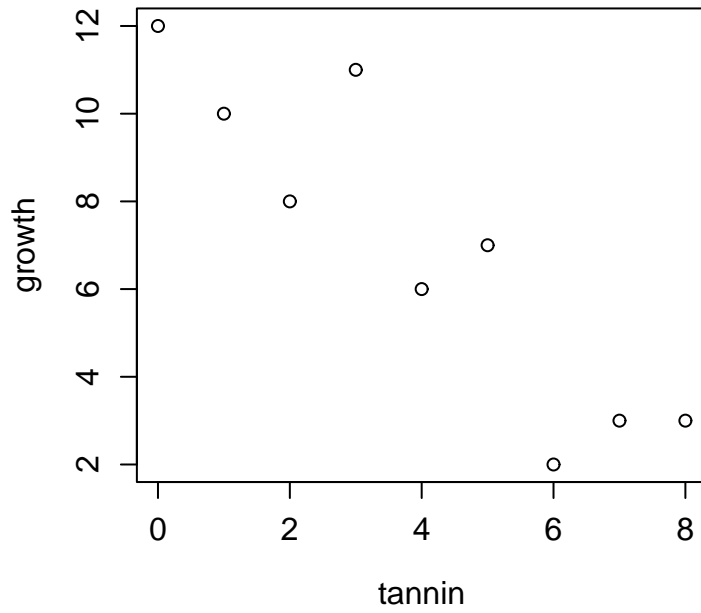
```
lm(formula = growth ~ tannin, data = a)
```

Coefficients:

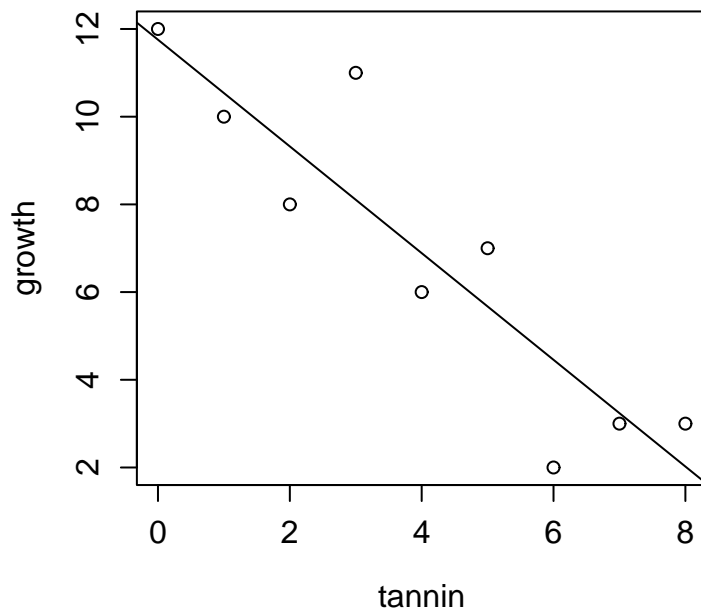
```
(Intercept)      tannin
    11.756         -1.217
```

this means to do a linear model of growth, so that it depends on tannin.

```
> plot( growth ~ tannin, data=a);v()
```



```
> abline( regression );v()
```



>

R found a good line. The method tries to minimize the square distance to the line.

Here is another line:

```
> reg1=lm( growth ~ 1, data=a)
```

```
> reg1
```

Call:

```
lm(formula = growth ~ 1, data = a)
```

Coefficients:

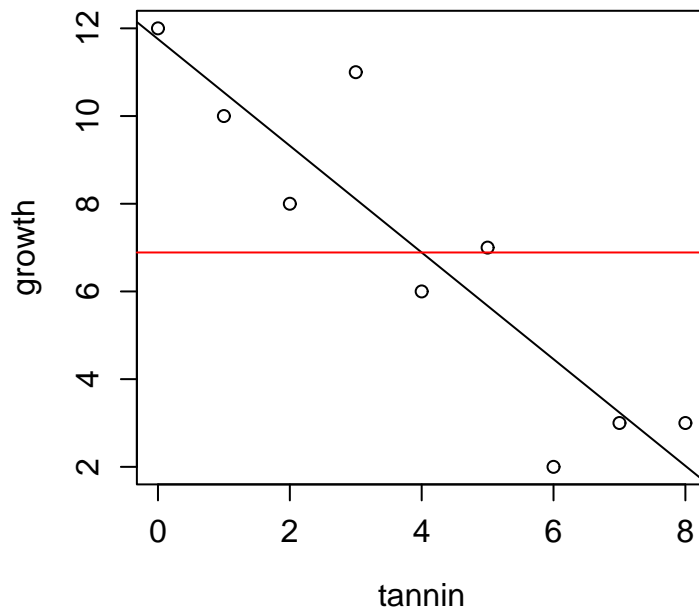
(Intercept)

6.889

In this case, R tries to fit a line that doesn't depend on anything, i.e. it is constant.

The best constant to fit the growth is 6.889

```
> abline(6.889,0,col=2); v()
```



```
> mean(a$growth)
```

```
[1] 6.888889
```

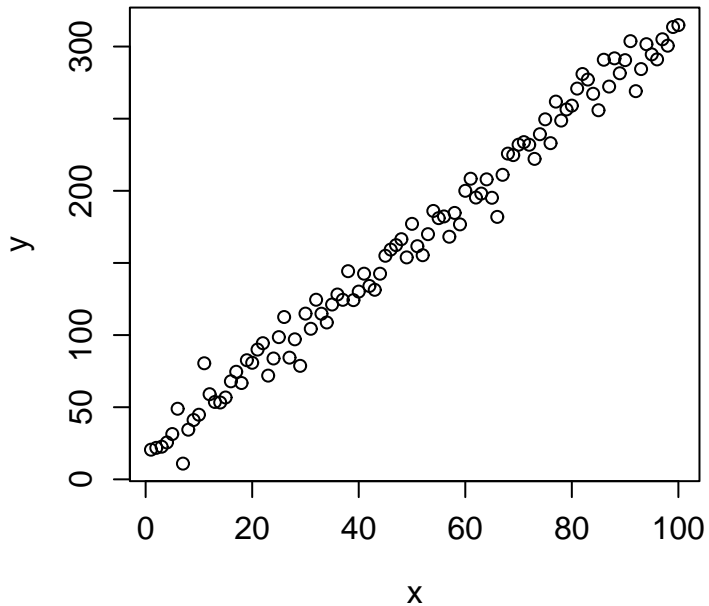
>

Such a line will be a good fit if we have normal errors.

For example:

```
> x = 1:100
```

```
> y= 3*x + 15 + rnorm(100,sd=10)
> plot(x,y); v()
```



```
> reg=lm( y ~ x )
> reg
```

```
Call:
lm(formula = y ~ x)
```

```
Coefficients:
(Intercept)          x
    16.567         2.977
```

```
>
```

We can see that the right values, 3 and 15 were almost found.

```
> summary(reg)
```

```
Call:
lm(formula = y ~ x)
```

```
Residuals:
    Min       1Q   Median       3Q      Max
-31.14652  -5.59417  -0.03833   6.73383  31.15194
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 16.56712    2.12266   7.805 6.6e-12 ***
x            2.97744    0.03649  81.592 < 2e-16 ***
```

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 10.53 on 98 degrees of freedom

Multiple R-Squared: 0.9855, Adjusted R-squared: 0.9853

F-statistic: 6657 on 1 and 98 DF, p-value: < 2.2e-16

>

Summary tells us that the standard error on the intercept is 2.1, and on the slope 0.03

That means that the estimate of the slope is better than that of the intercept.