

Lecture 5 - Plots and lines

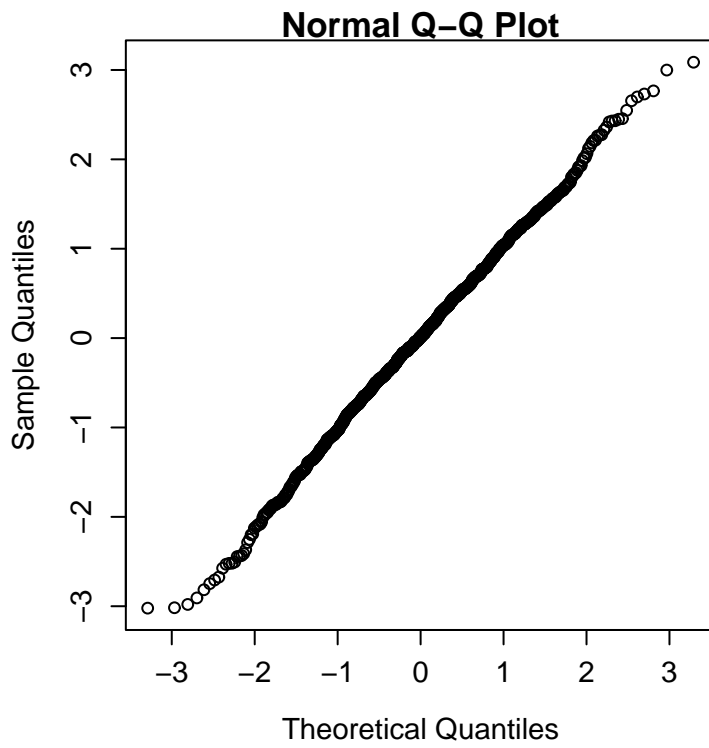
Some plot commands we already encountered

qqplot

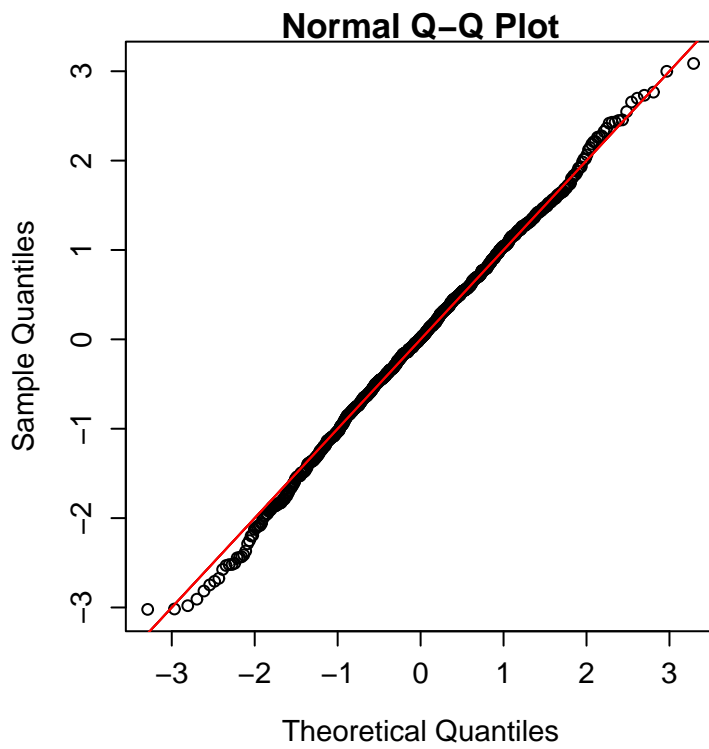
Last time we learned about qqplot and qqnorm. Let us see how to interpret them:

```
> 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

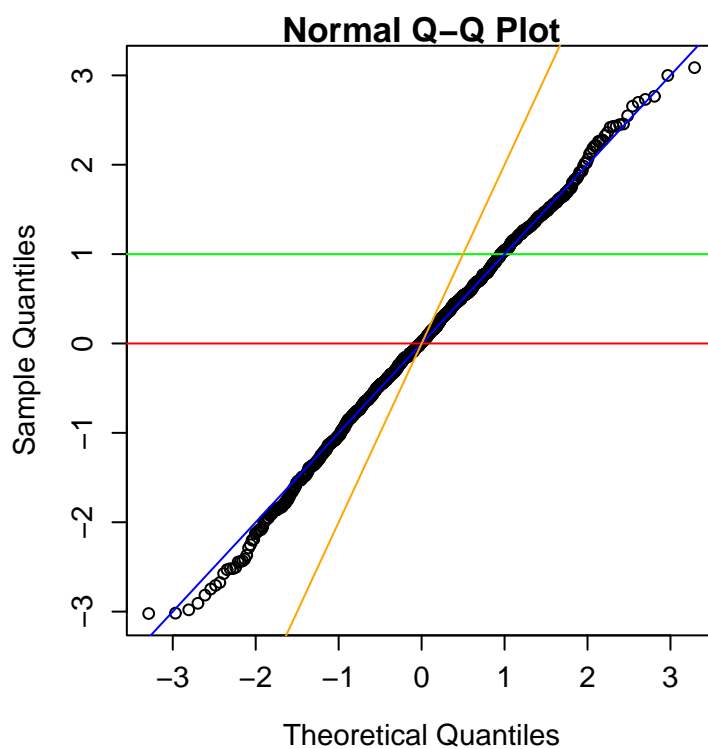
>

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()
```

>



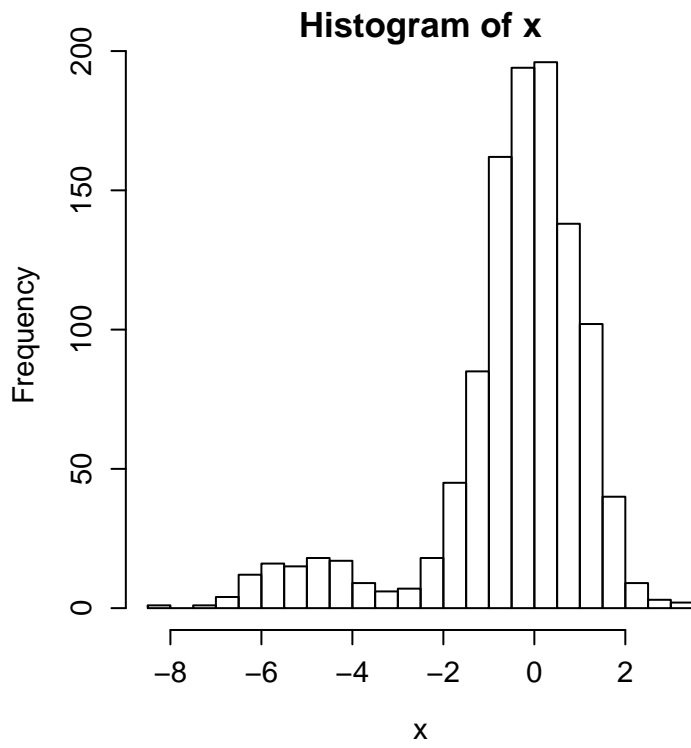
>

>

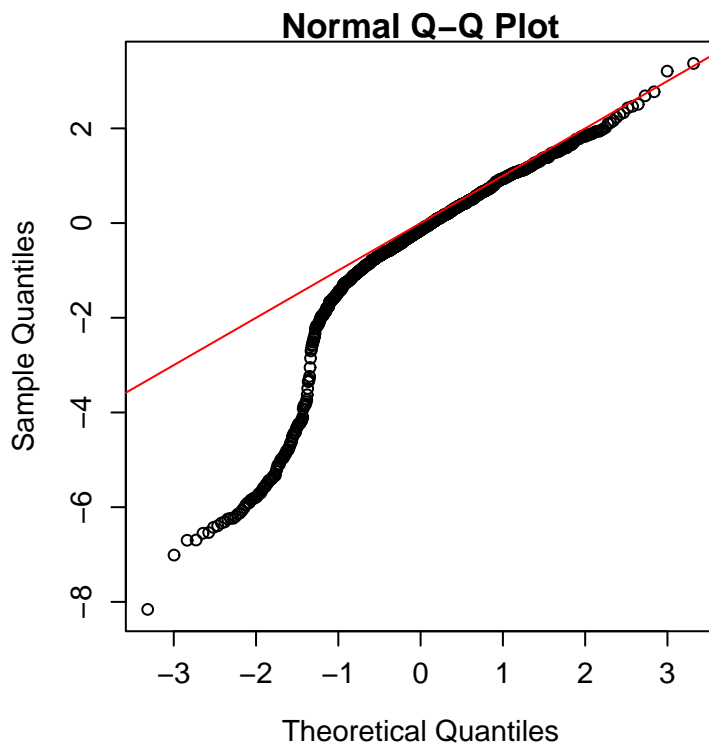
Let us add points on the right to the normal:

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

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



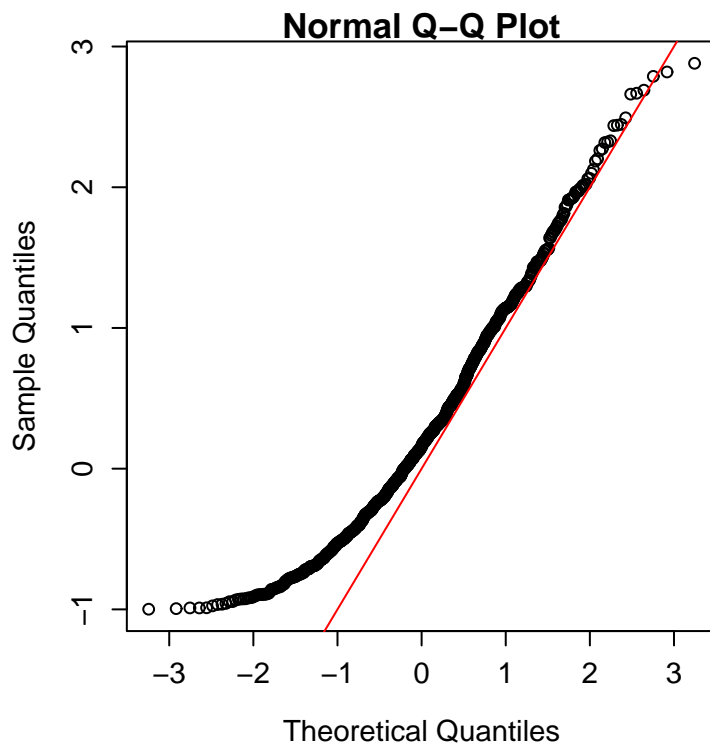
```
> qqnorm(x);abline(0,1,col=2);v()
```



>

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()
```

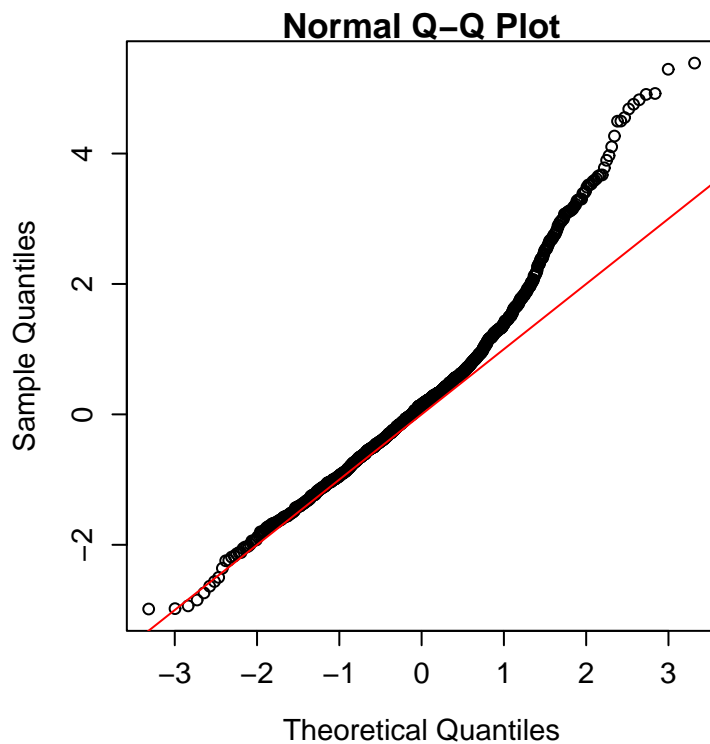


>

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()
```

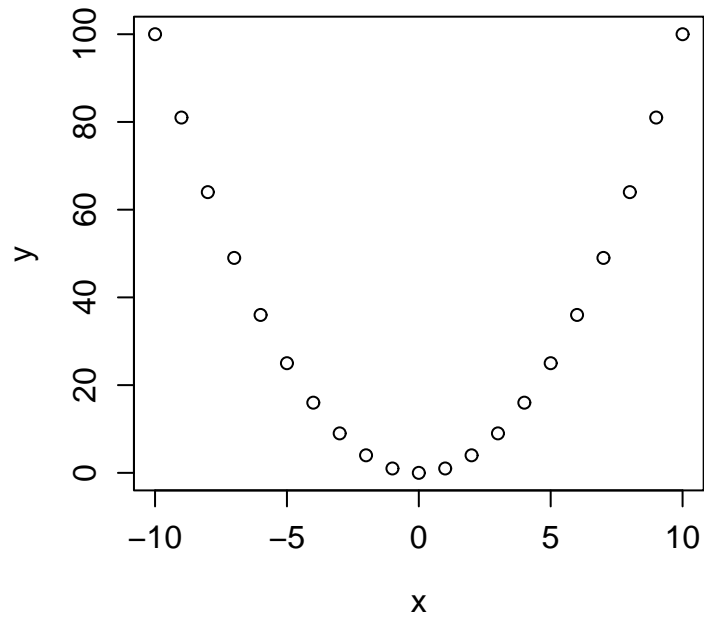


>

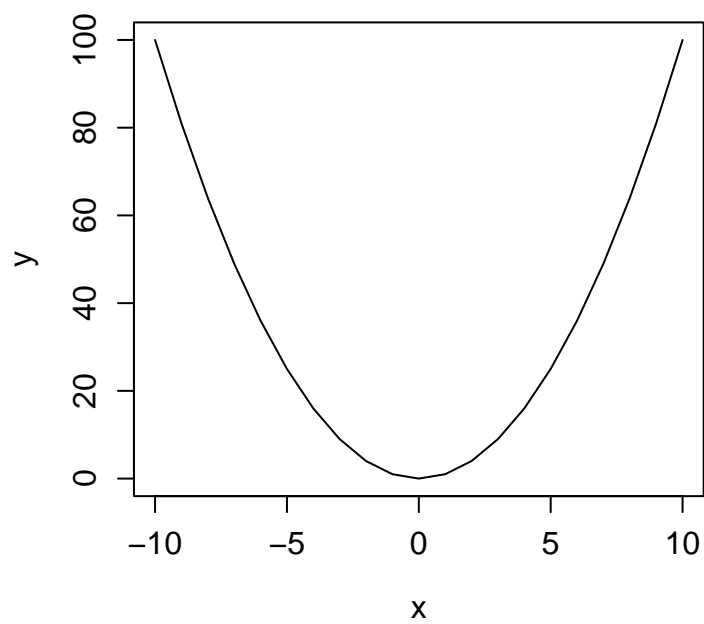
If we have too many points on the right, then the last quantiles occur too late, and the points move up.

plot

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



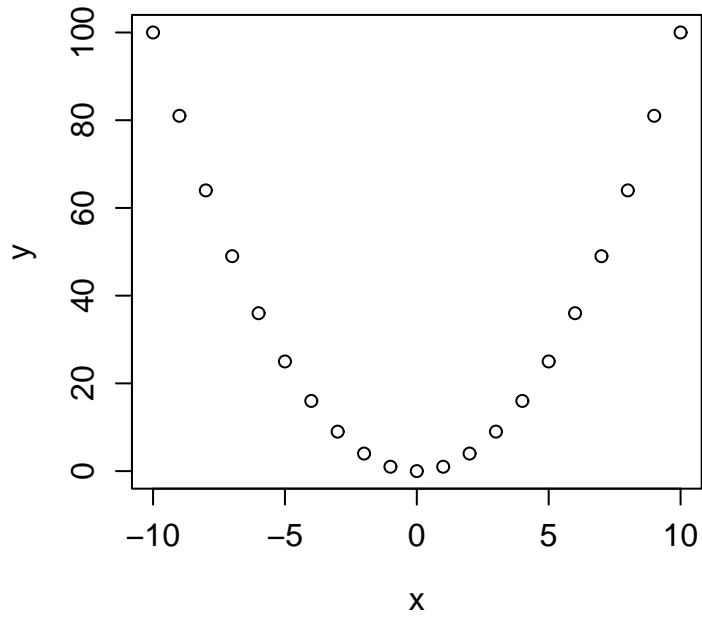
```
> plot(x,y,type="l");v()
```



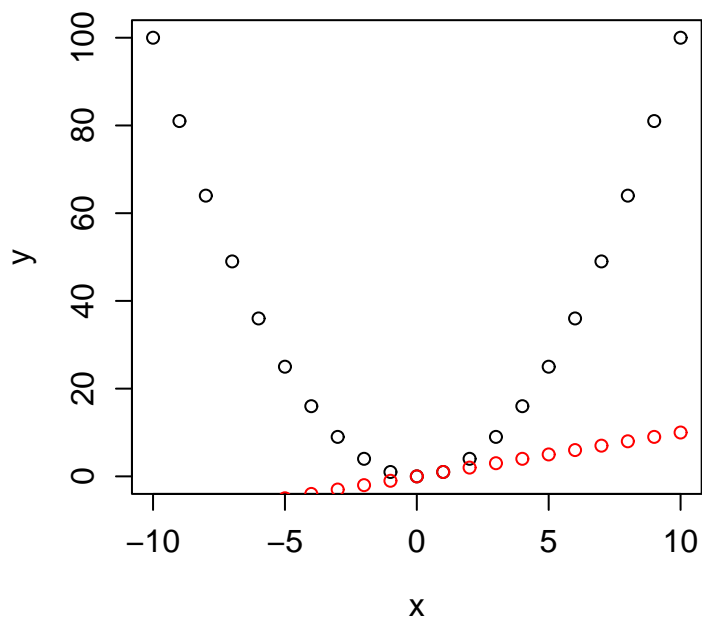
```
>
```

We can also add to an existing plot

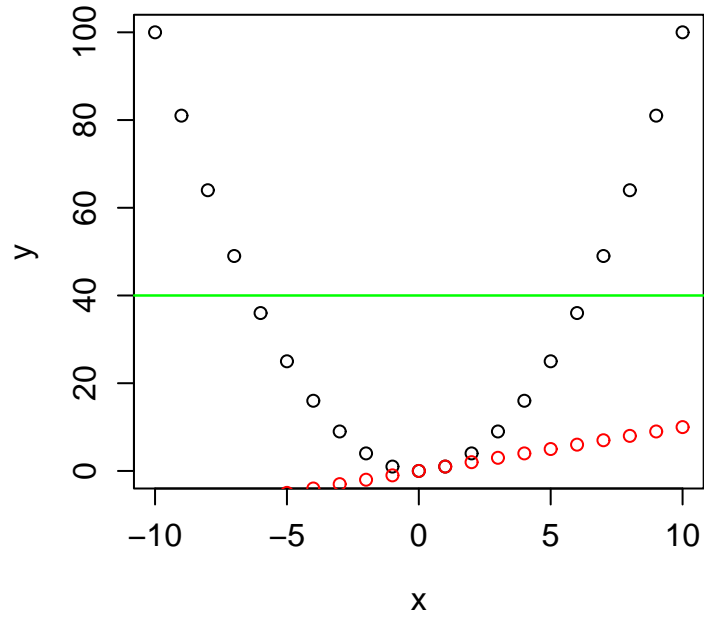
```
> plot(x,y);v()
```



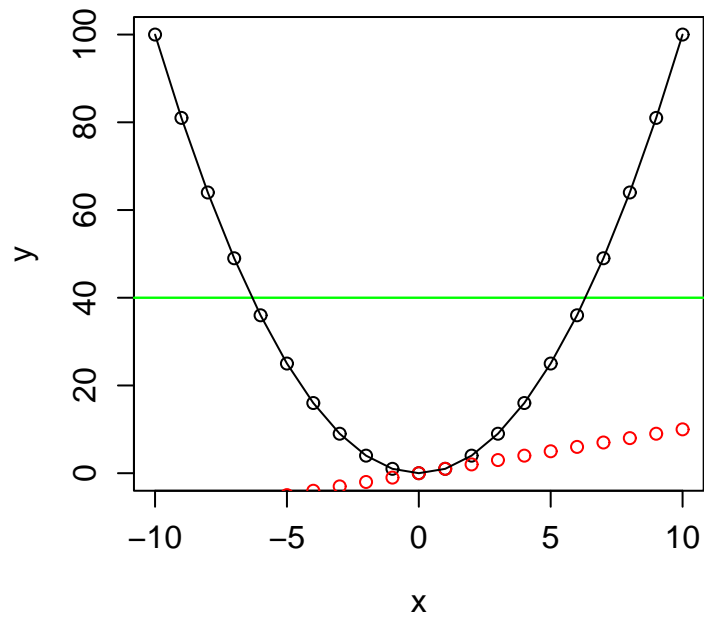
```
> points(x,x,col=2);v()
```



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

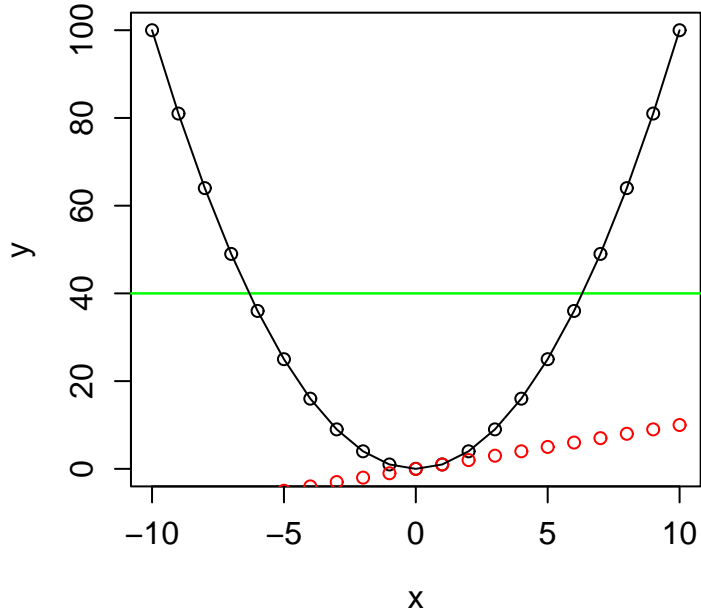


```
> lines(x,y);v()
```



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

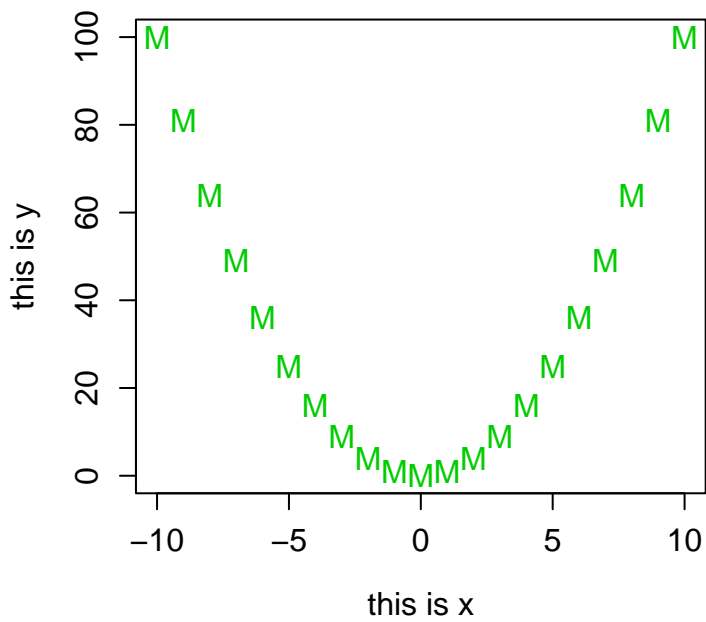
A nice plot



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

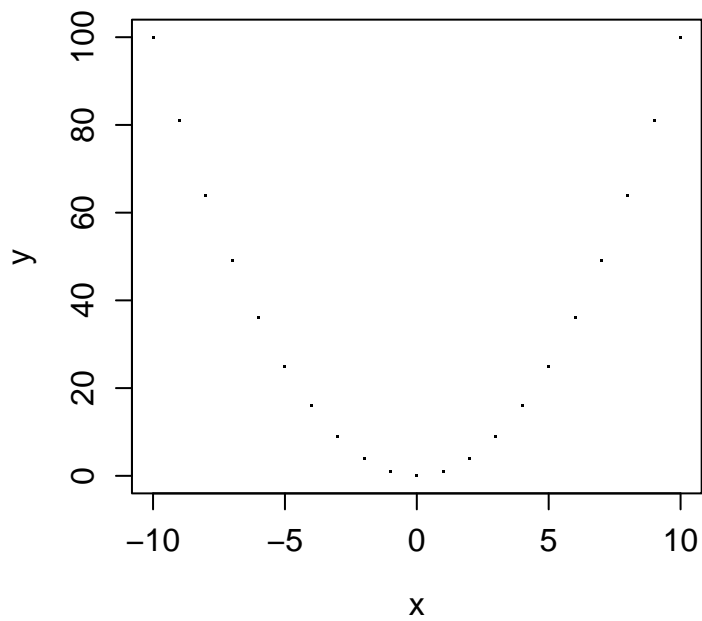
++

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

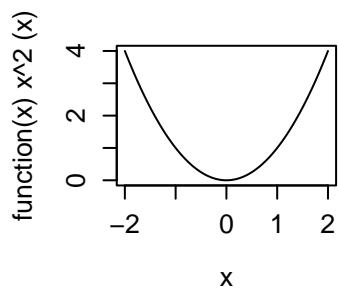
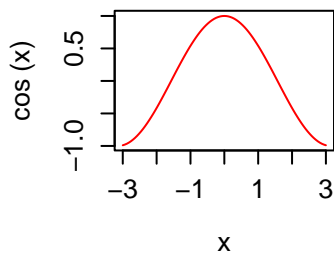
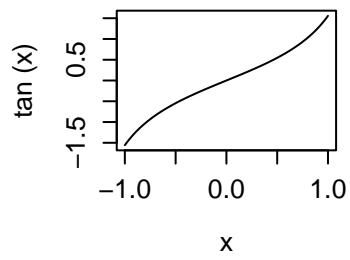
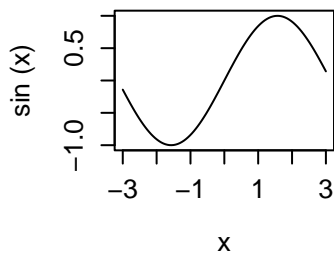
```
> two.by.two=matrix( 1:4, 2, 2)
> two.by.two
```

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

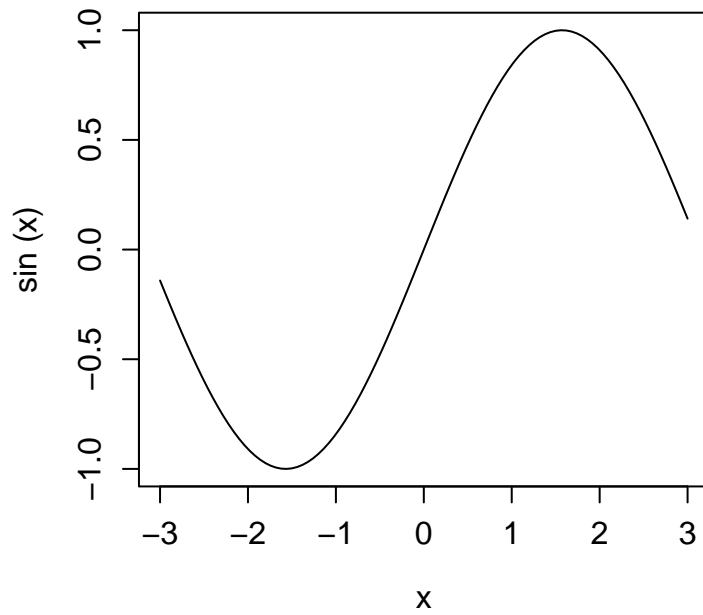
```
> layout(two.by.two)
> layout.show(4);v()
```

1	3
2	4

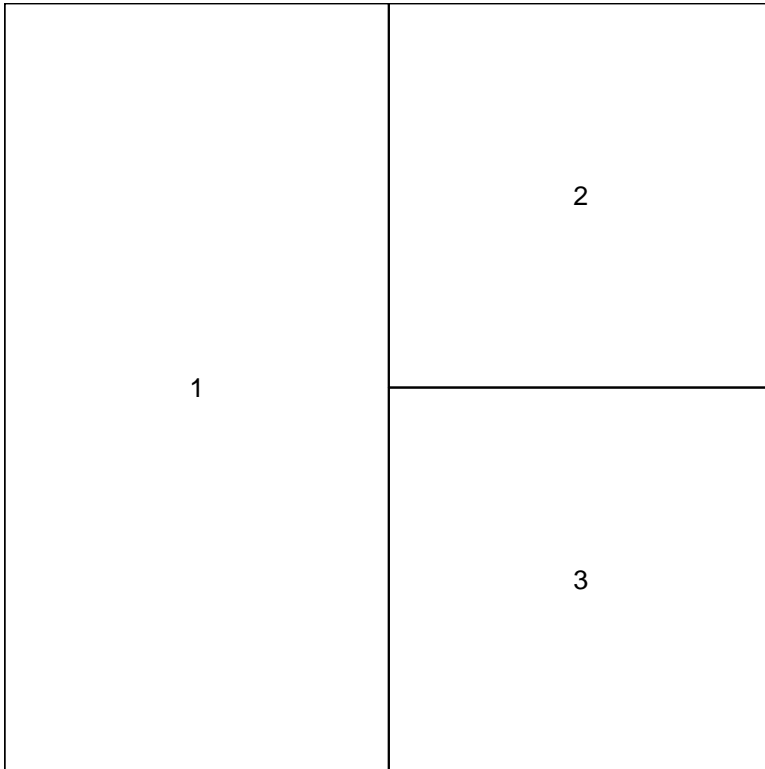
```
> plot(sin,-3,3)
> plot(cos,-3,3,col=2)
> plot(tan,-1,1)
> plot(function(x) x^2, -2, 2)
> v()
```



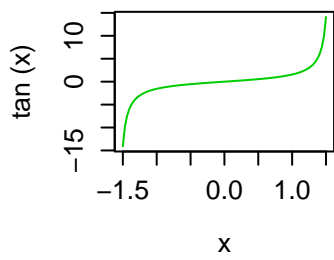
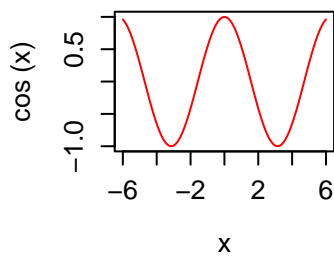
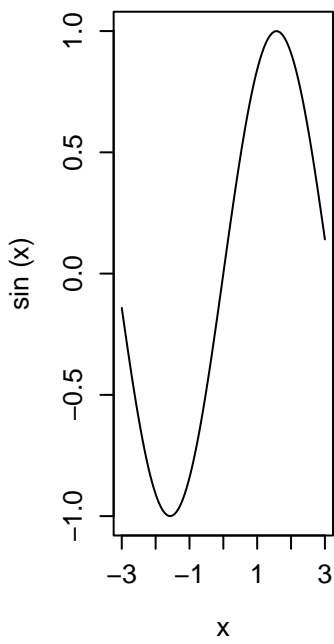
```
> layout(1)
> plot(sin,-3,3);v()
```



```
> 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
```

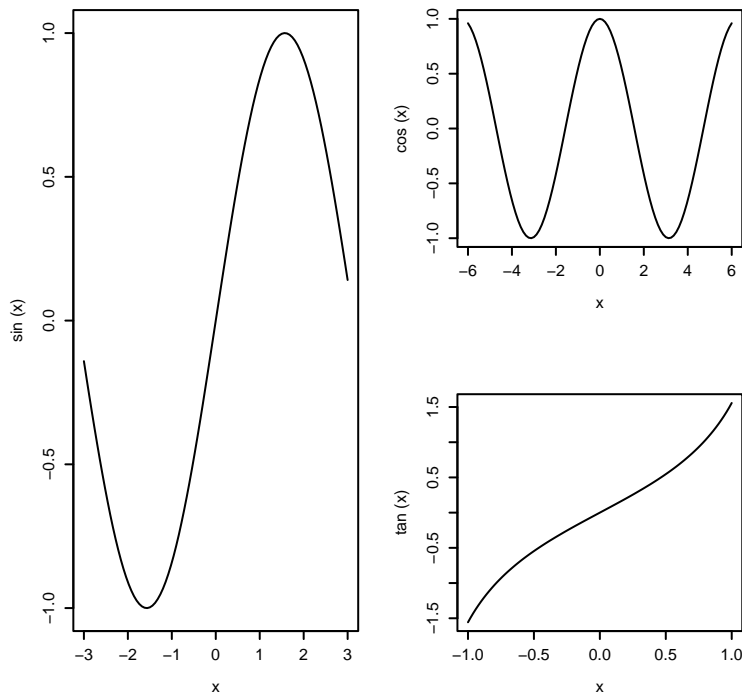
>

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

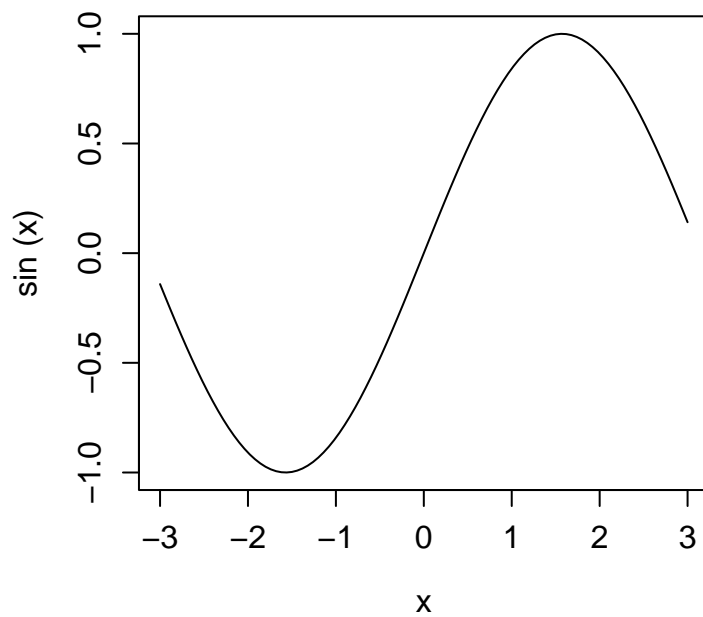
So, if we want to keep figures looking the same, we should scale the text:

```
> 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
>
```

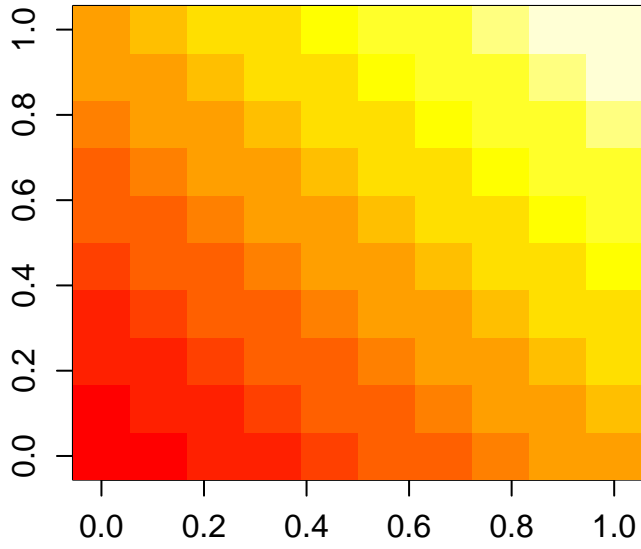
The function `layout()` changes the character size!

2D plots - the image function

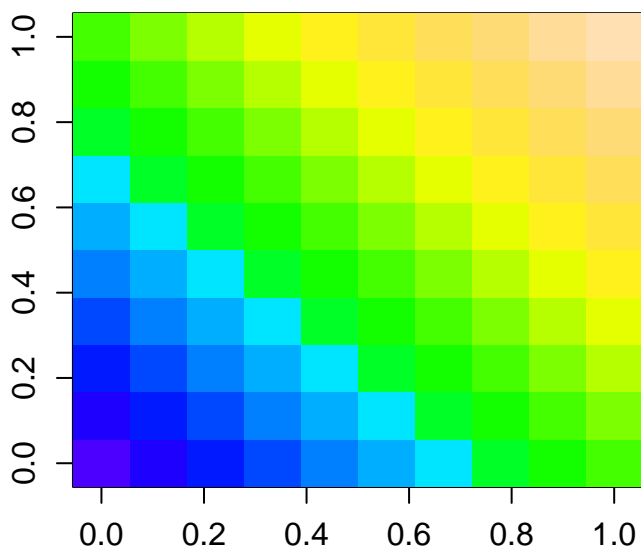
We already encountered the `image` function when we plotted the evolving population earlier. Let us see how this works:

```
> a=matrix(0,10,10)
> for(i in 1:10)
  for(j in 1:10)
    a[i,j] = i+j
++ >
```

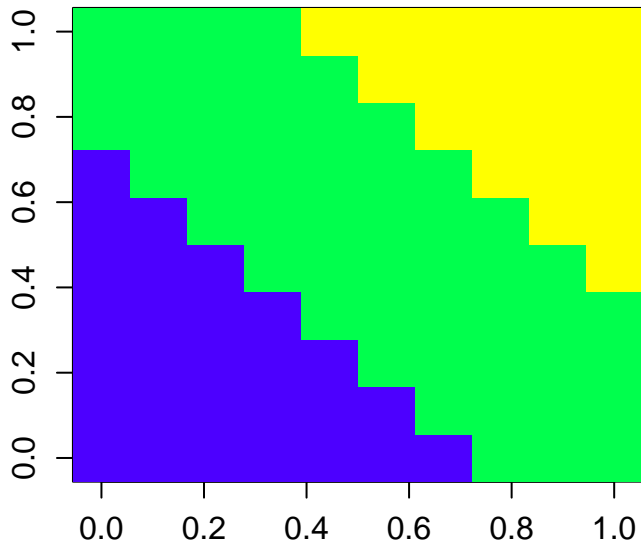
```
> image(a);v()
```



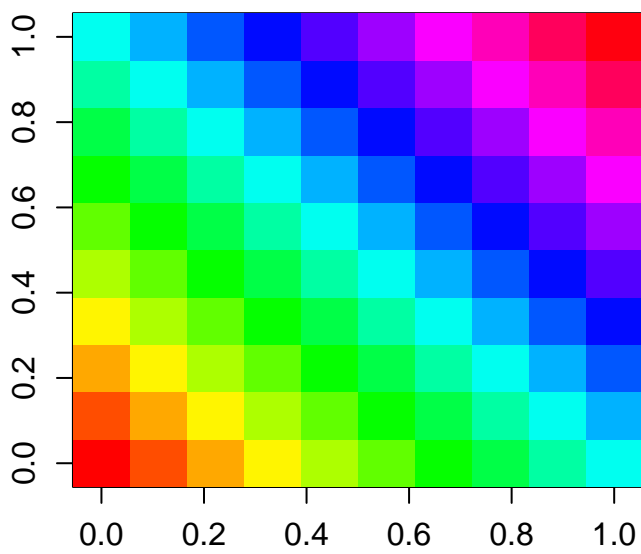
```
> image(a,col=topo.colors(100));v()
```



```
> image(a,col=topo.colors(3));v()
```



```
> image(a,col=rainbow(100));v()
```

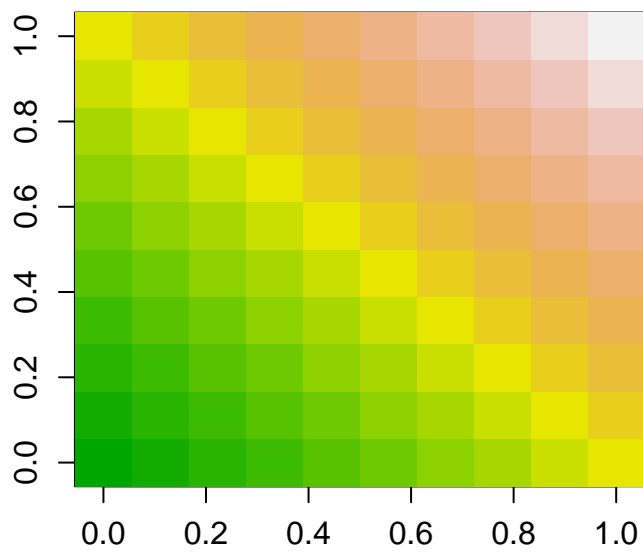


>

Side comment:

How to create the above matrix without a loop:

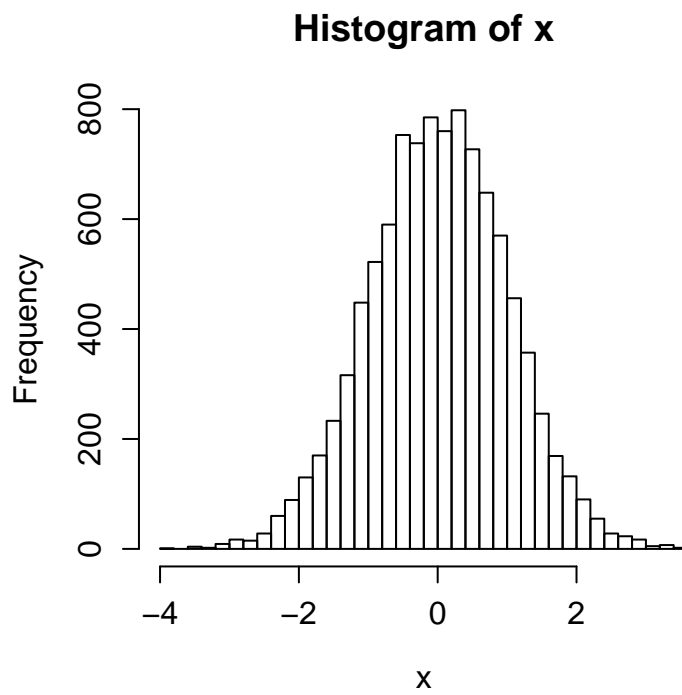
```
> a=outer(1:10,1:10,'+')  
> image(a,col=terrain.colors(100));v()
```



>

Some more about the normal distribution

```
> x=rnorm(10000)
> hist(x,n=30);v()
```



>

How far are 5% of the points?

```
> x=sort(x)
> length(x)/100*5
```

```
[1] 500
```

```
> x[500]
```

```
[1] -1.627749
```

```
> x[length(x)-500]
```

```
[1] 1.629363
```

>

We see that 5% of the points are 1.6 standard deviations below the mean, and 5% 1.6 above the mean.

```
> x[250]
```

```
[1] -1.945368
```

```
> x[length(x)-250]
```

```
[1] 1.971970
```

>

If we look at a distance of 1.96 std deviations away from the mean, or more, we'll see 5% of the points.

There is a function that does this:

```
> qnorm(0.025)
```

```
[1] -1.959964
```

```
> qnorm(0.05)
```

```
[1] -1.644854
```

```
> qnorm(0.005)
```

```
[1] -2.575829
```

>

It is convenient to remember that if we are ~ 2 std. deviations away, we are at the 5% level, and at 3 std. deviations we are at less than 1% level.

Even for this there is a function:

```
> pnorm(-3)*2
[1] 0.002699796
> pnorm(-2)*2
[1] 0.04550026
> pnorm(-1)*2
[1] 0.3173105
>
```

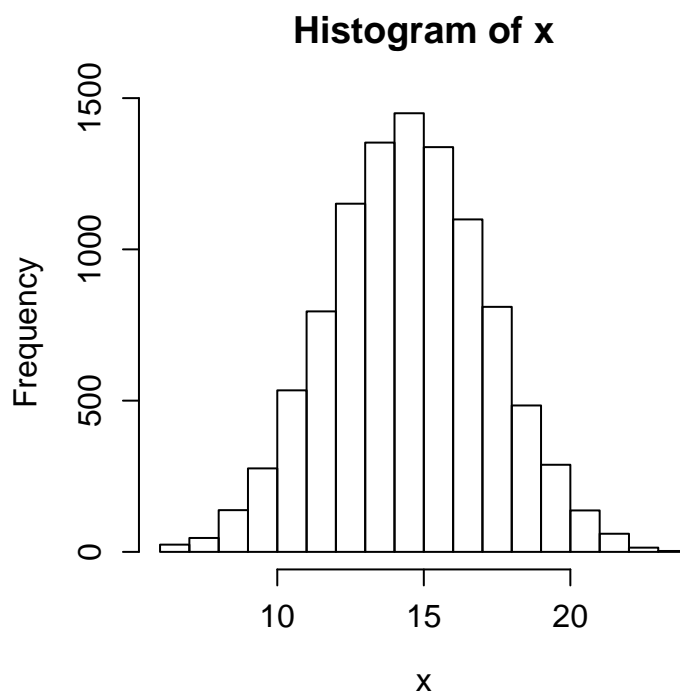
We saw 3 functions associated with the normal distribution:

`rnorm`, `qnorm`, `pnorm`.

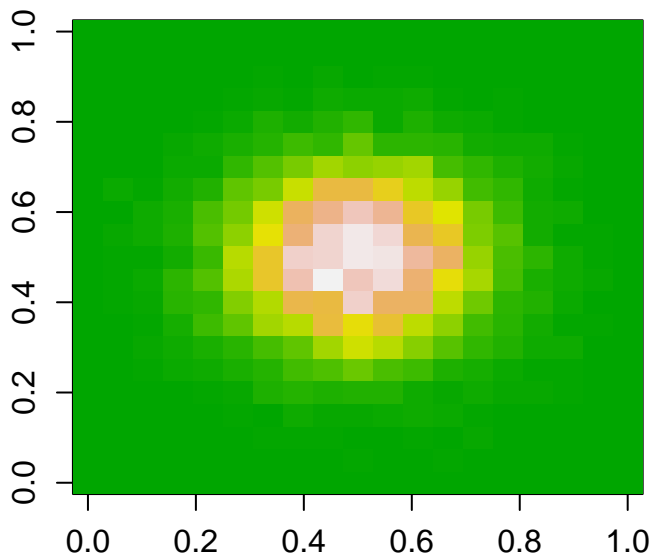
There is one additional, `dnorm`, which is the density.

These 4 functions are present for many distributions. For example, the binomial

```
> rbinom(10,30,0.5)
[1] 16 19 14 13 15 19 13 14 15 13
> x=rbinom(10000,30,0.5)
> hist(x);v()
```



```
> y=rbinom(10000,30,0.5)
> image(table(x,y),col=terrain.colors(100));v()
```



```
>
>
```

What is the chance to get 11 or less out of 30?

```
> sum(x <= 11)
[1] 1018
> sum(x <= 11)/length(x)
[1] 0.1018
> pbinom(11,30,0.5)
[1] 0.1002442
```

And 17 or more?

```
> sum(x >= 17)/length(x)
[1] 0.2895
> ?pbinom
```

Binomial package:stats R Documentation

The Binomial Distribution

Description:

Density, distribution function, quantile function and random generation for the binomial distribution with parameters 'size' and 'prob'.

Usage:

```
dbinom(x, size, prob, log = FALSE)
pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
rbinom(n, size, prob)
```

Arguments:

`x`, `q`: vector of quantiles.

`p`: vector of probabilities.

`n`: number of observations. If `'length(n) > 1'`, the length is taken to be the number required.

`size`: number of trials.

`prob`: probability of success on each trial.

`log`, `log.p`: logical; if TRUE, probabilities `p` are given as $\log(p)$.

`lower.tail`: logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.

Details:

The binomial distribution with `'size' = n` and `'prob' = p` has density

$$p(x) = \text{choose}(n, x) p^x (1-p)^{(n-x)}$$

for $x = 0, \dots, n$.

If an element of `'x'` is not integer, the result of `'dbinom'` is zero, with a warning. $p(x)$ is computed using Loader's algorithm, see the reference below.

The quantile is defined as the smallest value x such that $F(x) \geq p$, where F is the distribution function.

Value:

`'dbinom'` gives the density, `'pbinom'` gives the distribution function, `'qbinom'` gives the quantile function and `'rbinom'` generates random deviates.

If `'size'` is not an integer, `'NaN'` is returned.

References:

Catherine Loader (2000). `_Fast and Accurate Computation of Binomial Probabilities_`; manuscript available from <URL: <http://cm.bell-labs.com/cm/ms/departments/sia/catherine/dbinom>>

See Also:

'`dnbinom`' for the negative binomial, and '`dpois`' for the Poisson distribution.

Examples:

```
# Compute P(45 < X < 55) for X Binomial(100,0.5)
sum(dbinom(46:54, 100, 0.5))

## Using "log = TRUE" for an extended range :
n <- 2000
k <- seq(0, n, by = 20)
plot(k, dbinom(k, n, pi/10, log=TRUE), type='l', ylab="log density",
      main = "dbinom(*, log=TRUE) is better than log(dbinom(*))")
lines(k, log(dbinom(k, n, pi/10)), col='red', lwd=2)
## extreme points are omitted since dbinom gives 0.
mtext("dbinom(k, log=TRUE)", adj=0)
mtext("extended range", adj=0, line = -1, font=4)
mtext("log(dbinom(k))", col="red", adj=1)
```

```
> 1-pbinom(17,30,0.5,lower=T)
```

```
[1] 0.1807973
```

```
> sum(x >= 17)/length(x)
```

```
[1] 0.2895
```

```
>
```

Why is that?

```
>
```

At what point can we reject the hypothesis that the coin is fair?

```
> qbinom(0.05,30,0.5)
```

```
[1] 11
```

```
> pbinom(11,30,0.5)
```

```
[1] 0.1002442
```

```
> pbinom(10,30,0.5)
```

```
[1] 0.04936857
```

```
> qbinom(0.95,30,0.5)
```

```
[1] 19
```

```
> pbinom(18,30,0.5)
```

```
[1] 0.8997558
```

```
> pbinom(19,30,0.5)
```

```
[1] 0.9506314
```

```

>
> sum(x < 11 | x > 19)/length(x)
[1] 0.0986
> qbinom(c(0.025,0.975),30,0.5)
[1] 10 20
> sum(x < 10 | x > 20)/length(x)
[1] 0.0422
>

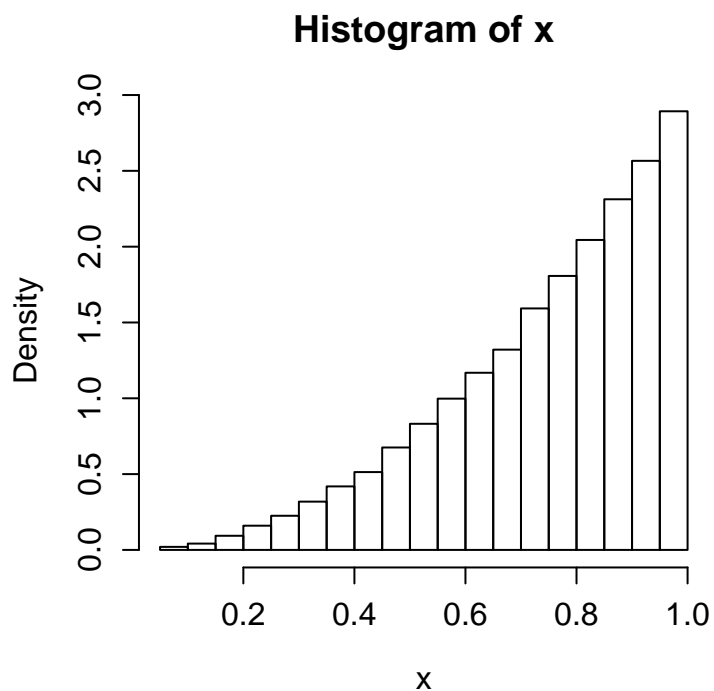
```

Another distribution is the uniform distribution

```

> runif(10)
[1] 0.421884943 0.003038261 0.944455681 0.793884157 0.273780152 0.203727521
[7] 0.468094018 0.169645566 0.674382183 0.177781886
> runif(10,min=-5,max=5)
[1] 1.8787583 2.4242072 0.1591313 -4.6074219 2.6036353 1.1042102
[7] -1.1624493 -3.5101939 4.2309112 -2.8054177
> x=runif(100000)
> y=runif(100000)
> i=y < x^2
> x=x[i]
> hist(x,freq=F);v()

```

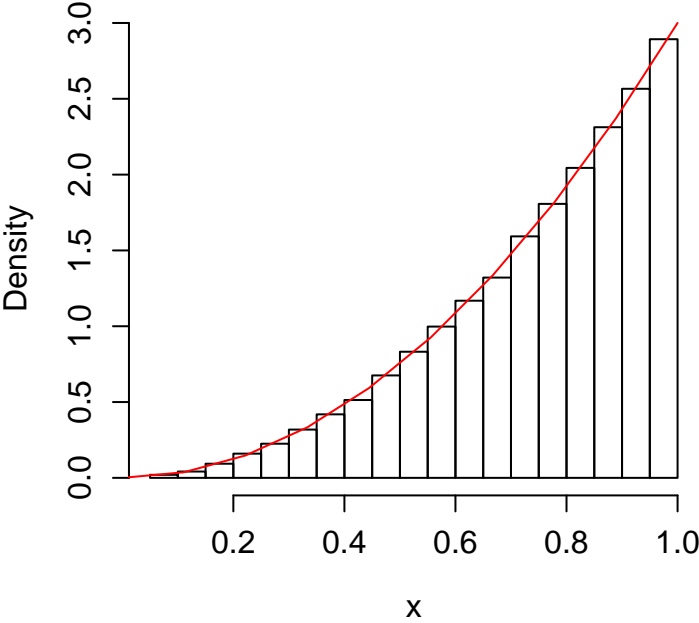


```

> z=seq(0,1,length=10)
> lines(z,3*z^2,col=2);v()

```

Histogram of x



>