Working with random numbers

In a statistical analysis, randomness is used as a tool in several ways. Sir Ronald Fisher came up with the idea that you should randomly assign individuals to treatments in an experiment, in order to be able to distinguish treatment effects properly. The concept of randomness is also essential for statistical inference. Using a single dataset, statistical inference combines the data with a certain hypothesis, in order to assess how plausible the data are if you would collect the same type of data a large number of times.

This afternoon, we will produce and exploit controlled randomness.
A helper file with R commands is available on blackboard. It is called "random2007.txt". Carefully scrutinizing the R code is your major task in the exercise. Save a copy of the helper file for yourself, and add ample comments to make sure you can modify the code later, when you need to, for other assignments and for your own projects.

♣ Statistical Distributions

Various stochastic processes can generate randomness, and this randomness is perceived or observed in the form of random numbers. Often, different stochastic processes share similarities, and these similarities make it that the processes that generate random numbers are described by families of probability distribution functions.

Wikipedia (www.wikipedia.org) currently contains quite good descriptions of many probability distributions, in my opinion. You can also consult an appropriate book in a library.

Look up the following distributions. For every distribution in the list, suggest a data variable that you could collect, which can be described by such a distribution. For example, *sex ratios of offspring of a number of female painted turtles, can be described by binomial distributions.*

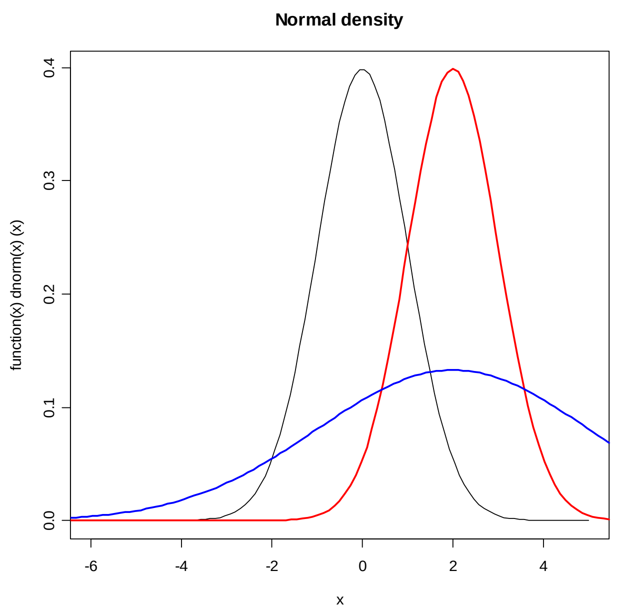
- Poisson
- gamma
- exponential
- chi-square
- normal
- Pareto (...sounds exotic)
- quasi-binomial (...a teaser)

All probability distributions have probability mass functions, cumulative distribution functions, quantiles, and other properties. We will plot and calculate some of these properties for various distributions.
First, read up a little on two functions we will use for plotting:

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

We will now make a graph of three normal distributions. The normal distribution has two parameters, one for the mean, and one for the standard deviation of the random numbers. Find out in which parameters the three following distributions differ.

```
plot(function(x)dnorm(x), -6, 5, main = "Normal density")
curve(dnorm(x,2), add=TRUE, col="red",lwd=2)
curve(dnorm(x,2,3), add=TRUE, col="blue",lwd=2)
```



We can also plot cumulative distribution functions of these distributions:

```
plot(function(x)pnorm(x), -6, 10, main = "Cumulative normal density")
curve(pnorm(x,2), add=TRUE, col="red",lwd=2)
curve(pnorm(x,2,3), add=TRUE, col="blue",lwd=2)
```

Random variables are usually written as X, their probability mass function as $f(X)$, and the cumulative distribution function as $F(X)$.

The quantile (or *fractile*) x_q of the distribution, with $0 < q < 1$, is defined by

$$q = P(X < x) = F(x) = \int_{-\infty}^{x} f(\xi)d\xi$$

i.e., q is the probability of observing $X < x_q$. The quantile $x_{1/2}$ is called the *median* of the distribution; $x_{1/4}$ and $x_{3/4}$ are the lower and upper *quartiles*. In analogy, also *quintiles* and *percentiles*, etc., are in use. Some examples:

```
qnorm(0.25,mean=2,sd=1)
qnorm(0.5,mean=2,sd=1)
qnorm(0.75,mean=2,sd=1)
```

Quantiles can also be defined for one minus the cumulative distribution function, for example:

```
qnorm(0.25,mean=2,sd=1, lower.tail=FALSE)
```

♣ Statistical Inference

Not only in the description of randomness, but also in hypothesis testing, random variables play the major role. Consider the following *t*-test on artificial data, used to test whether the mean of two samples differs. The artificial data consist of measurements on two groups. For the first group, the measurements are (1, 2, 3, 4, 5, 6, 7, 8, 9, 10), for the second group (7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20).

```
outputtest<-t.test(x=c(1:10),y=c(7:20),var.equal=TRUE)
```

The *outputtest* object contains the following information:

```
        Two Sample t-test

data:  1:10 and c(7:20)
t = -5.1473, df = 22, p-value = 3.691e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -11.223245  -4.776755
sample estimates:
mean of x mean of y
      5.5      13.5
```

The means of the variables in the two groups are significantly different. Note that the test assumes random numbers with equal variances in the two groups. Using the cumulative distribution function of the student t distribution, we can calculate the tail probability of the two-sided test 'by hand'. For the artificial data, we then need the value of the t statistic, and the degrees of freedom to do that:

```
2*pt(outputtest$statistic,df=outputtest$parameter)
```

or one can extract the corresponding variable from *outputtest*:

```
outputtest$p.value
```

In the R package, all probability distributions defined have the possibility to draw random numbers from them (or actually pseudo-random numbers). To draw 100 random numbers from a Poisson distribution with parameter *7.6*, all you need to type is:

```
data<-rpois(100,7.6)
```

Plot the result:

```
hist(data)
```

and convince yourself that these data have the property of a Poisson distribution that mean and variance are equal:

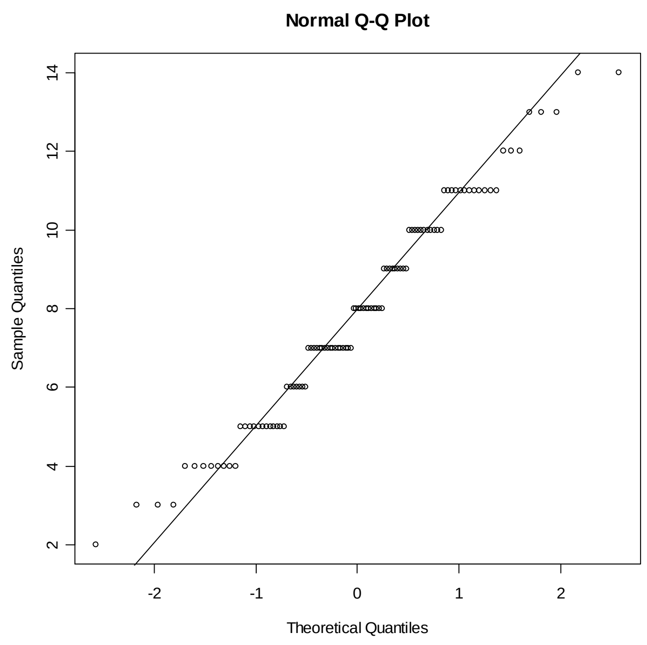
```
mean(data)
var(data)
```

In the final stage of an analysis, when you are busy with model checking, so called probability plots are often used. I guess that everybody has heard at least once about normal probability plots. Normal probability plots give a visual way to determine if a data distribution is approximately normal. These plots are produced by doing the following.

1. The data values are arranged from smallest to largest.
2. The percentile of each data value is determined.
3. From these percentiles, normal calculations are done to determine their corresponding quantiles.
4. Each quantile is plotted against its corresponding data value.

If the data distribution is normal, then the points should form an approximate straight line in the probability plot. Departures from this straight line indicate departures from normality.

```
qqnorm(data)
qqline(data)
```



However, you can make the same type of plots for other distributions as well. These always have 'quantiles' of a probability distribution on one axis, and ordered data points on the vertical. If the distribution fits, you should observe a more or less straight line.

We draw 100 random numbers from a binomial distribution where the sample size per draw is 20, and the probability of success per item 0.34 (so each grab of marbles contains 20 marbles, the proportion of beautiful ones in the bag is 34%. You can grab 100 times).

```
x<-rbinom(100,20,0.34)
```

We compare sample x to a Poisson distribution with parameter value *9*.

```
plot(qpois(ppoints(x),9),sort(x))
```

or to a binomial distribution

```
plot(qbinom(ppoints(x),20,0.34),sort(x))
```

or to a gamma distribution

```
plot(qgamma(ppoints(x),9),sort(x))
```

Which one of the three distributions fits the simulated data *x* worst?

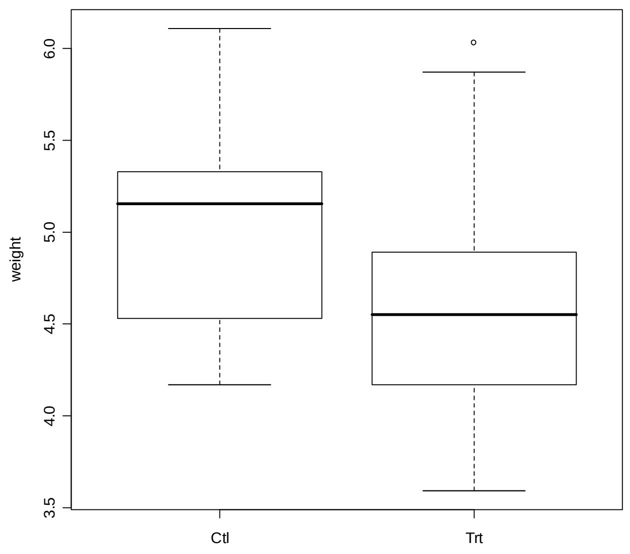
♣ Simulate Data

In order to prepare our brain already a bit for the exercises of the following days, we will now simulate a relatively simple dataset.
This task starts by running an example of help(lm), from page nine of Dobson's (1990) book on generalized linear models. The data is on weight of plants, receiving either a Control ("Ctl") or a Treatment ("Trt") with a potential influence on weight.

```
## Annette Dobson (1990) "An Introduction to Generalized Linear Models".
## Page 9: Plant Weight Data.
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
group <- gl(2,10,20, labels=c("Ctl","Trt"))
weight <- c(ctl, trt)
```

The data look like this:

```
boxplot(weight~group ,ylab="weight")
```



We can fit a linear model to the data

```
summary(lm.D9 <- lm(weight ~ group))
```

and test whether the weights of the two groups are really different

```
drop1(lm.D9,test="F")
```

```
Single term deletions
```

```
Model:
weight ~ group
       Df     Sum of Sq    RSS        AIC         F value     Pr(F)
<none>                     8.7293     -12.5811
group  1      0.6882       9.4175     -13.0633    1.4191      0.249
```

and take a look at the estimates of the model coefficients ,which we will use below.
Is the group effect significant?
Can you write down what the model for the weight data looks like in polynomial form?

```
 lm.D9$coef
```

We are now going to simulate a new dataset, with 100 observations, according to the estimated model. Go through the following lines carefully. You will understand them quicker if you inspect the variables which are created, and if you modify the code with small changes and inspect the effects of those changes.

```
mean<-rep(lm.D9$coef[1],100)
gr<-gl(2,50, labels=c("Ctl","Trt"))
groupeffect<-ifelse(gr=="Trt",lm.D9$coef[2],0)
error<-rnorm(100,sd=0.696)
```

This is the resulting weight data set:

```
w<-mean+groupeffect+error
```

Which you can inspect by means of graphical representations, or by fitting the linear model again:

```
boxplot(w~gr)
summary(lm(w ~ gr))
drop1(lm(w ~ gr),test="F")
```

If you want to run such a simulation repeatedly, and you only want to inspect a simple numerical outcome of the simulation, then it becomes advantageous to combine the previous lines into a function so that you don't need to retype a lot and therefore you can't introduce many new typing errors either.
Here we make a function with arguments "N", which is the sample size, and "sdev" which is the standard deviation of the error variable. So if you feed this particular function a value for "N" and another one for "sdev", then it will give the tail probability of the F-test of the group effect in a balanced dataset.

```
samplepvalD9<-function(N,sdev){
m<-rep(lm.D9$coef[1],N)
grps<-gl(2,N/2, labels=c("Ctl","Trt"))
groupeff<-ifelse(grps=="Trt",lm.D9$coef[2],0)
err<-rnorm(N,sd=sdev)
wght<-m+groupeff+err
drop1(lm(wght ~ grps),test="F")$"Pr(F)"[-1]
}
```

An example:

```
samplepvalD9(10,1)
```

When you program a function yourself, be aware of the following complications:
- In the body of the function , which is inside the "{" and "}" brackets, local variables occur, which have no meaning outside these brackets.
For example:

```
m
```

does not return anything. Usually you try to maximize the number of local variables in a function, and minimize the so-called global ones. For the samplevalD9() function, the model coefficients of *lm.D9* act as global variables.

```
lm.D9$coef[1]
lm.D9$coef[2]
```

- Inside the function, it can be necessary to extract a value from the output of another function. That is what the "drop1(lm(wght ~ grps),test="F")$"Pr(F)"[-1]" line does. When you are programming your own functions, you can use the str() function to inspect the values of an object which you might need. For example, to extract all values inside the drop1() command, you type

```
str(drop1(lm(w ~ gr),test="F"))
```

It will usually require some fiddling to precisely get hold of the value you need.

♣ Power Analysis

We can also use simulations to carry out a power analysis: So if the alternative hypothesis were really correct, what would be the probability to obtain a significant test? Using the example and the parameter estimates from Dobson, a power analysis of the group effect goes as follows:

```
nreplicates<- 500
```

We will use 500 simulated datasets to estimate the power of our test of the group effect.

First set up a vector to hold the p-value of each simulation:

```
pval <- numeric(nreplicates)
```

We will test the power of an experiment with 50 observations, where the standard deviation of the error is 0.7, and where the estimates of the treatment effects are as in the given data.
That requires a loop, which is used to fill up al entries in the *pval* vector.

```
for (i in 1:nreplicates) {
  pval[i] <- samplepvalD9(50,0.7)
}
```

Then this is the power of such a test:

```
sum(pval<0.05)/nreplicates
```

Now it's your turn to write a program.
Write a function which does a power analysis using sample size and the standard deviation of the error as arguments, and which uses the estimates of the treatment effects as in the Dobson data. The output of the function has to be the power.
Use this function to investigate how the power depends on the error variance, given a certain sample size.
Let the standard deviation of the error vary from 0.1 to 1 in small steps, and make a graph of the power as a function of the error variance.
What can you conclude?

♠ Many packages can draw pseudo-random numbers

Excel can drawn random numbers from a limited number of distributions

Poptools has facilities for so-called Monte-Carlo simulation

Mathematica and **Maple** have many functions to work with distribution functions and random numbers.

For **C++** , there are libraries for random number generation, such as **newran** or the **boost/random** library.

For applications of random numbers in cryptography, pseudo-random numbers are just not good enough. If someone knew the state of your computer at a given moment, it would be possible to reconstruct any sequence of random numbers it draws. For securing data, random numbers have to be really random, see: www.random.org on this interesting issue.

Tom Van Dooren 09/2007
t.j.m.van.dooren@biology.leidenuniv.nl