Distributions

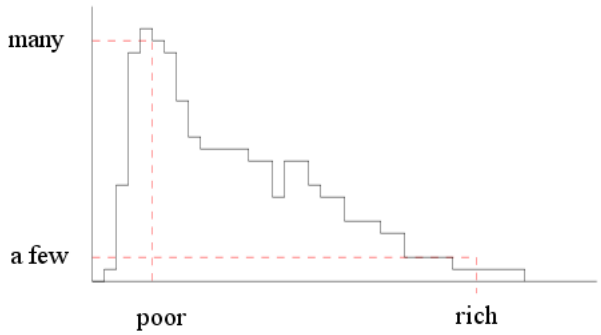
Contents

1 The concept of distributions

Statistics deals with random data. The randomness present in a data is quantified through the concept of (probability) distribution. By the distribution of a random variable we understand any statement that specifies two things:

1. the set of possible values
2. a rule which, given any subset of values, tells us the the probability of the random variable taking a value from that subset.

Suppose that we make a list of all the people in a country and make a histogram of their incomes, i.e., plot the number of people at each income level against the income levels. I should get a histogram like this



Income histogram of a typical country

The shape of this histogram gives us an idea about the income distribution: a few people are very rich, quite a lot are poor, the bulk being of the middle income range.
It is a common practice in statistics to regard the distribution as the *ultimate truth* behind the data. Most statistical procedures try to infer about the underlying distribution based on the data.

2 How to express

There are various ways to express a distribution mathematically. A popular way is by the **probability density function (pdf)** which is a nonnegative function $f(x)$ with

$$\int_{-\infty}^{\infty} f(x)dx = 1.$$

Often we consider families of pdf's such that all the members in the same family have some common shape, and differ only in terms of some quantitative details controlled by a small number of **parameters**. Familiarity with the shapes of each family may be gained by using R as follows.

```
curve(dnorm(x,mean=0,sd=1),-5,5,ylim=c(0,1),ylab='')
title(main=expression(paste("N(",mu,", ",sigma^2,") densities")))
curve(dnorm(x,mean=0,sd=0.5),-5,5,add=T,col='red')
curve(dnorm(x,mean=0,sd=1.5),-5,5,add=T,col='blue')
curve(dnorm(x,mean=1,sd=1),-5,5,add=T,col='green')
```

Let's add a legend to the plot.

```
legend("topright",col=c('black','red','blue','green'),lwd=1,
legend=c(expression(paste(mu,'=0 ',sigma,'=1')),
expression(paste(mu,'=0 ',sigma,'=0.5')),
```

```
expression(paste(mu,'=0 ',sigma,'=1.5')),
expression(paste(mu,'=1 ',sigma,'=1'))))
```

Exercise 1: Make a similar plot for the Exponential(λ) distribution, which has p.d.f.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases}$$

The R function to compute this pdf is `dexp(x,rate)`, where `rate` is what we have called $\lambda(> 0)$. ■

3 How to estimate

A typical statistical analysis does not start with a distribution, rather with a data set. We often make the assumption that the data set comes from some unknown distribution, and try to estimate the distribution from the data. Some estimation procedures do not assume any knowledge about the family of the distribution. These are called the **nonparametric** techniques. The **parametric** techniques on the other hand assume a known family and try to estimate the parameters to choose the most suitable member of the family.

3.1 Nonparametric

A simple graphical method to estimate a distribution is to draw the histogram. Let's first load the SDSS quasar data set.

```
# Construct large and small samples of SDSS quasar redshifts and r-i colors
qso = read.table(
"http://astrostatistics.psu.edu/datasets/SDSS_QSO.dat",
head=T,fill=T)
qso = na.omit(qso)
dim(qso)
names(qso)
summary(qso)
```

Next we extract the variables of interest.

```
z.all = qso[,4]
r.i.all = qso[,9]-qso[,11]
SDSS.all = data.frame(z.all,r.i.all)
```

```
oldpar = par(mfrow=c(1,3))
hist(z.all, main='', xlab='Redshift') #Default binning
hist(z.all, breaks=50, main='', xlab='Redshift') #10 bins
hist(z.all, breaks='scott', main='', xlab='Redshift')
par(oldpar)
```

The `breaks='scott'` option means the classes are determined by some algorithm called Scott's algorithm (we hardly need to worry about what that is at this stage).
A histogram is often used to approximate a pdf. Since the area under a pdf is 1, it is sometimes desirable to scale the histogram appropriately to make the area 1. This may be done using the `prob=T` option:

```
hist(z.all, main='', xlab='Redshift',prob=T)
```

This option is useful when we want to overlay a pdf curve on the histogram, as we shall do later.
Another approach resulting in a plot that is slightly difficult to interpret is to plot the percentiles.

```
plot(quantile(z.all, (1:100)/100), pch=20, cex=0.5,
     xlab='Percentile', ylab='Redshift')
```

A more frequently used method is to draw the empirical cumulative distribution function (ecdf). For a sample $x_i, ..., x_n$ the ecdf is defined as a function from real line to [0,1] as

$$\hat{F}_n(x) = \frac{1}{n}(\text{Number of } x_i\text{'s} \leq x.$$

To understand the structure of the ecdf let us focus on only a few points, say the first 10:

```
plot(ecdf(z.all[1:10]))
```

Here is a more annotated example.

```
# Read magnitudes for Milky Way and M 31 globular clusters
GC1 = read.table(
"http://astrostatistics.psu.edu/MSMA//datasets/GlobClus_MWG.dat"
```

```
,header=T)
GC2 = read.table(
"http://astrostatistics.psu.edu/MSMA/datasets/GlobClus_M31.dat",
header=T)
K1 = GC1[,2]
K2 = GC2[,2]

plot(ecdf(K1), cex.points=0.5, xlab="K (mag)", ylab="E.C.D.F.",
  main="Milky Way & M 31 globular cluster KLF")
plot(ecdf(K2-24.90), col.hor=2, col.vert=2, col.points=2,
     pch=3, cex.points=0.5, add=T)
text(-7.5, 0.8, lab="MWG")
text(-10.5,0.9, lab="M 31", col=2)
```

The ecdf function, however, does not provide any confidence band for the estimated cdf. For that we could use the ecdf.ksCI function from the sfsmisc package.

```
install.packages('sfsmisc')
library('sfsmisc')
ecdf.ksCI(K1)
```

A more sophisticated technique is called **kernel density estimation**. Here we imagine a small Gaussian[1] density centred around each data point, and average all these densities. While there are R packages to compute kernel density estimates easily, it is instructive to spell out the steps graphically. We shall work with the first 5 points from the data set.

```
z.5 = z.all[1:5]
```

Next we shall consider a Gaussian density centred at each of these points. We shall take a pretty small σ, say 0.05.

```
plot(0,0,xlim=range(z.5),ylim=c(0,10),xlab='z',ylab='density',ty='n')
for(val in z.5) curve(dnorm(x,mean=val,sd=0.05),xlim=range(z.5),add=T)
rug(z.5) #Adds tiny vertical bars on the horizontal axis
         #to mark each point
```

Now we are going to average the densities.

[1] It is possible to use other symmetric densities as well.

```
zgrid = seq(min(z.5),max(z.5),len=100)
kd = sapply(zgrid,
      function(x) mean(dnorm(x,mean=z.5, sd=0.05)))
lines(zgrid,kd,lwd=3,xlim=range(z.5))
```

A more sophisticated version of the same thing is done by the `density` function of R. However, we prefer to use the external package `sm` since it also finds confidence bands. We shall work with the following subset of the data.

```
z.200 = z.all[1:200]
```

```
install.packages('sm')
library(sm)
?sm.density #looking up help
dens = sm.density(z.200, h=bw.nrd(z.200)) #Choosing h automatically
lines(dens$eval.points, dens$upper, col=2)
lines(dens$eval.points, dens$lower, col=2)
```

3.2 Parametric

Here we have to first choose a family of distributions (typically based on the underlying theory or the histogram). If the chosen family is a standard one then the `fitdistr` function from the `MASS` package can hopefully pick out the best fitting member of the family.

```
# Read Milky Way Galaxy and M 31 globular cluster K magnitudes
GC_MWG = read.table(
'http://astrostatistics.psu.edu/MSMA/datasets/GlobClus_MWG.dat',
header=T)
KGC_MWG = GC_MWG[,2]
kseq = seq(min(KGC_MWG)-1, max(KGC_MWG)+1, 0.25)
# Fit normal distributions
library(MASS)
par(mfrow=c(1,2))
hist(KGC_MWG, breaks=kseq,
     main='', xlab='K mag',ylab='N',col=gray(0.5),prob=T)
normfit_MWG = fitdistr(KGC_MWG,'normal')
normfit_MWG

lines(kseq,
```

```
dnorm(kseq,
      mean=normfit_MWG$estimate[[1]],
      sd=normfit_MWG$estimate[[2]]))
```

Exercise 2: Repeat the same process for the M 31 globular cluster K magnitudes. The data set is at

`http://astrostatistics.psu.edu/MSMA/datasets/GlobClus_M31.dat`

■

4 How to compare data with a distribution

4.1 Comparing a single number with a distribution

A dog that is 12 years old, is old enough. But a human being of the same age is considered young. Our conclusion here depends on the number 12 as compared against the lifespan distribution of the species in question.
More mathematically, if we know the distribution of a random variable X, and we are asked to judge if a certain given value c is too large for it, we can check the probability $P(X > c)$. If it is small, then we may say that c looks like too high a value for X to take. This is the idea behind the **(upper-tailed) p-value** for comparing a number against a given distribution. Of course, we can similarly consider the **(lower-tailed) p-value** $P(X < c)$ to judge if c is too small a value for X.
We have to sometimes judge if a given value c is "too extreme" for X. It is not easy to come up with a **(two-tailed) p-value** in general. However, this question often arises in situations where the distribution of X is symmetric around 0. Then we use $P(X > |c|)$ as the (two-tailed) p-value.
Before we learn how to compute these using R, let us notice one common feature of any type of p-value:

> a small p-value implies that the given number does not go well with the given distribution.

Let us see how to compute $P(X > c)$ if X has a $N(1, 2^2)$ distribution and $c = 4$.

```
1-pnorm(4,mean=1,sd=2)
```

The `pnorm` function computes the Gaussian cumulative distributions. Compare its name with `dnorm`, the Gaussian pdf. Every standard distribution in R has 4 functions associated with it:

- a `p-` function: the distribution function,

- a `d-` function: the pdf/pmf function,
- a `q-` function: the inverse distribution function, also called the quantile function,
- an `r-` function: the random number generator function.

You may like to look up the helps of `runif`, `dlnorm`, `pexp` and `qgamma`.
If the distribution is not available mathematically, but we have only a sample from the distribution, then we can proceed as follows.

```
x = rnorm(100,mean=1,sd=2) #We simulate a data set first
mean(x>4)
```

This technique sits at the heart of all randomized and bootstrap tests, as we shall see soon.
It is the standard practice of all statistical softwares to report results of tests in terms of p-values.

4.2 Comparing a sample with a distribution

Many statistical procedures are geared to worked for sample from a specific (family of) distribution. Blindly applying these techniques without checking the distributional assumptions may lead to nonsensical results. Here we discuss some ways to check if a given sample can be safely assumed to come from a given distribution. A first rough check may always be done by eye-balling the shape of the data histogram.
A more easily interpretable method is the **quantile quantile plot**, where the quantiles of the distribution are plotted against the quantiles of the sample.

4.2.1 Testing Gaussianity

This is required so often, that the quantile-quantile plot for Gaussian distribution has a special name **normal plot**. It is implemented by the R function `qqnorm`

```
# Read magnitudes for Milky Way and M 31 globular clusters
GC1 = read.table(
"http://astrostatistics.psu.edu/MSMA//datasets/GlobClus_MWG.dat",
header=T)
names(GC1)
summary(GC1)
qqnorm(GC1[,2])
```

Graphical methods are easy to interpret, but introduces an element of subjectivity that is often undesirable. So a class of objective tests have been devised for testing a sample for normality.

```
asteroids = read.table(
"http://astrostatistics.psu.edu/MSMA/datasets/asteroid_dens.dat",
header=T)
dens = asteroids[,2]
summary(dens)
```

Now we shall apply a standard test called Shapiro-Wilk's test for normality. This test, like any other statistical test, first summarizes the data set into a single number generically called the **test statistic**. The test statistic for the Shapiro-Wilk's test is denoted by the symbol W. The designers of the test have worked out the distribution of W under the assumption that the data set actually hails from a normal distribution. The observed value of W is then compared against this distribution using the p-value technique given earlier.

```
shapiro.test(dens)
```

The Shapiro-Wilk's test is by no means the only test for normality. The package `nortest` contains a slew of other tests.

```
install.packages('nortest')
library('nortest')
```

- **Anderson-Darlington test:** Here the test statistic is

$$A = -n - \frac{1}{n}\sum_{i=1}^{n}[2i-1][\ln(p_{(i)}) + \ln(1-p_{(n-i+1)})],$$

  where $p_{(i)} = \Phi([x_{(i)} - \overline{x}]/s)$. The R command is

```
ad.test(dens)
```

- **Cramer-von Mises test:** This test uses the test statistic

$$\frac{1}{12n} + \sum_{i=1}^{n}(p_{(i)} - \frac{2i-1}{2n}),$$

  where $p_{(i)} = \Phi([x_{(i)} - \overline{x}]/s)$. The R command is

```
cvm.test(dens)
```

Here are some others tests from the same package.

```
lillie.test(dens)
pearson.test(dens)
```

The package `outliers` contains some more.

```
install.packages('outliers')
library('outliers')
dixon.test(dens)
chisq.out.test(dens)
grubbs.test(dens)
grubbs.test(dens,type=20)
```

4.2.2 Testing for general distribution

The Kolmogorov-Smirnov test is useful for testing a sample comes from a given *continuous* distribution.

```
ks.test(dens,"pexp",rate=3) #Just a silly example!
```

This test has problem with ties (i.e, repeated data values).

4.3 Comparing the distribution of one sample with another

Sometimes we have two samples, and want to compare the distributions they come from.

```
# Read magnitudes for Milky Way and M 31 globular clusters
GC1 = read.table(
"http://astrostatistics.psu.edu/MSMA//datasets/GlobClus_MWG.dat",
header=T)
GC2 = read.table(
"http://astrostatistics.psu.edu/MSMA/datasets/GlobClus_M31.dat",header=T)
K1 = GC1[,2]
K2 = GC2[,2]
```

Visual comparison of the two histograms is of course one simple (and crude) way to do this.

```
par(mfrow=c(1,2))
hist(K1)
hist(K2)
par(mfrow=c(1,1))
```

Notice that they shapes are quite similar in both the cases. However, the horizontal scales are shifted.
We can also plot a 2-sample quantile-quantile plot, which plots the quantiles of one sample against those of the other.

```
qqplot(K1, K2, xlab="MWG", ylab="M31",
       main="Milky Way & M31 QQ Plot")
```

The linear pattern again indicates of the similarity of the two distributions. Indeed, the linear pattern means that one may be obtained from the other by some shifting and/or scaling. Let's try shifting first. We need to have an idea as to how much we should shift by. Here are three ways:

```
# Three estimates of the distance modulus to M 31
DMmn = mean(K2) - mean(K1) ; DMmn
sigDMmn = sqrt(var(K1)/length(K1) + var(K2)/length(K2)) ; sigDMmn
DMmed = median(K2) - median(K1) ; DMmed
sigDMmed = sqrt(mad(K1)^2/length(K1) + mad(K2)^2/length(K2)) ; sigDMmed
wilcox.test(K2, K1, conf.int=T)
```

```
qqplot(K1, K2-24.90, xlab="MWG", ylab="M31-24.90", main="Milky Way & M31 QQ Plot")
```

Does it require scaling also? To find out we can compare the linear pattern with the $y = x$ line:

```
abline(0,1) #intercept=0, slope=1
```

The slope is different from 1, so the two distributions still differ in scale.
We can also compare the distributions underlying two independent samples using the 2-sample Kolmogorov-Smirnov test or the Mood Test.

```
# Two-sample tests for Milky Way and M31 globular clusters
ks.test(K1, K2-24.90) # with distance modulus offset removed
mood.test(K1, K2-24.90)
```

5 How to tweak data to modify distribution

Many statistical procedures require a sample from a Gaussian distribution. But suppose that the histogram of the sample turns out to be different from the Gaussian symmetric bell shaped curve. If the histogram has a single peak but is skewed to one side, then we can transform the data (in a one-to-one fashion) such that the transformed data has a bell-shaped histogram. Such a family of transform is the **Box-Cox transforms** defined by

$$f_\lambda(x) = \begin{cases} \frac{x^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log(x) & \text{otherwise.} \end{cases}$$

Here λ is the "skew-correction" control. Its appropriate value needs to be chosen based on the amount and direction of skewness present in the sample. Typically, values between -2 and 2 are used.
The `car` package of R computes Box-Cox transform. The following R code seeks to choose a good value for λ based on the histogram. The exercise that follows urges you to come up with a more objective way.

```
qso = read.table(
"http://astrostatistics.psu.edu/datasets/SDSS_QSO.dat",
head=T,fill=T)
qso = na.omit(qso)
z.all = qso[,4]
install.packages('car')
library(car)
lambda = seq(-1,1,0.5)
tmp=sapply(lambda,function(x) bcPower(z.all,lambda=x))
par(mfrow=c(2,3))
sapply(1:ncol(tmp), function(i) hist(tmp[,i],main=lambda[i]))
par(mfrow=c(1,1))
```

Exercise 3: Instead of drawing histograms why not apply the Shapiro-Wilk's test for each value of λ, and choose the value yielding the highest p-value? Implement this idea by `apply`-ing `shapiro.test` to the columns of `tmp`. ■

6 Simulation: generating data from a distribution

A distribution is an idealized mathematical description of the behaviour of a sample. Thus, in a typical statistical analysis, sample comes first, and a distribution is postulated by the scientist to provide an analytically tractable way to represent the behaviour of the sample. But there are situations where we start with a distribution and want to have a sample from it:

- We may want to evaluate some statistical method, which requires data with some specific distributional assumption. Then we may simulate data with those assumptions and check if the method really works or not. Also, we may deliberately generate data violating the assumptions to see how robust the method is.
- Suppose that we have some data with error bars. We want to use some statistical method that is meant for precise data. Then we can incorporate the error bars by running the procedure a large number of times each time freshly jittering the data within the error bars.
- A distribution may be to complicated to analyze mathematically. Then we can try to simulate a large sample from it, and base our conclusion on the sample.

We shall illustrate these scenarios now.

6.1 Simulation to evaluate performance

We shall check if Shapiro-Wilk's method is a good test for normality or not. Like any statistical test, it can fail in two ways, either by failing to identify a truly normal sample, or by wrongly identifying a non-normal sample as normal. Statistical literature calls these, respectively, the **type I** and **type II** errors. Let's try to get an idea about the type I error:

```
tmp = sapply(1:100,function(x) shapiro.test(rnorm(100))$p.val < 0.05)
mean(tmp)
```

The **power** of a test is defined as $1-$ type II error. In our example, it measures how well the Shapiro-Wilk's test is able to detect a non-Gaussian distribution. Let's generate samples from Exponential(1) distribution, and see if Shapiro-Wilk's test detects that the data are from some non-normal distribution:

```
mean(sapply(1:100,function(x) shapiro.test(rexp(100))$p.val < 0.05))
```

6.2 Simulation to incorporate error bars

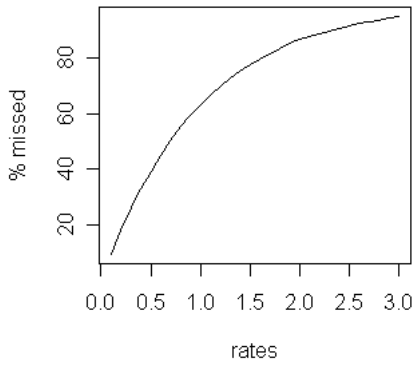
Jittering data is a simple way to incorporate the effects of error bars. We shall illustrate this with a regression example in the next lecture.
For now, here is a *skeleton* of the approach. Don't type this verbatim!

```
simerr = function(cen,lo,hi=lo)
  runif(length(cen),min=cen-lo,max=cen+hi)

tmp = sapply(1:100,
  function(dummy) {
    x=simerr(......)
    y=simerr(......)
    ......
  })
```

6.3 Simulation to simplify mathematics

A huge radio active source emits particles at random time points. A particle detector is detecting them. However, the detector is imperfect and gets "jammed" for 1 sec every time a particle hits it. No further particles can be detected within this period. After the 1 sec is over, the detector again starts functioning normally. We want to know the fraction of particles that the detector is missing. Since this fraction may depend on the rate at which the source emits particles we want to get a plot like the following:



Want a plot like this

Random emission of particles from a huge radio active source is well-studied process. A popular model is to assume that the time gaps between successive emissions are independent and have Exponential distributions.

```
gaps = rexp(999,rate=1)
```

The question now is to determine which of the particles will fail to be detected. A particle is missed if it comes within 1 sec of its predecessor. So the number of missed particles is precisely the number of gaps that are below 1.

```
miss = sum(gaps < 1)
miss
```

Now we want to repeat this experiment a number of times, say 50 times.

```
missList = c()
for(i in 1:50) {
   gaps = rexp(999,rate=1)
   miss = sum(gaps < 1)
   missList = c(missList,miss)
}
mean(missList)
var(missList)
```

All these are done for rate = 1. For other rates we need to repeat the entire process afresh. So we shall use yet another `for` loop.

```
rates = seq(0.1,3,0.1)
mnList = c()
vrList = c()
for(lambda in rates) {
   missList = c()
   for(i in 1:50) {
      gaps = rexp(1000,rate=lambda)
      miss = sum(gaps < 1)
      missList = c(missList,miss)
   }

   mn = mean(missList)
   vr = var(missList)
   mnList = c(mnList,mn)
   vrList = c(vrList, vr)
}
```

Now we can finally make the plot.

```
plot(rates,mnList/10,ty="l",ylab="% missed")
```

We shall throw in two error bars for a good measure.

```
up = mnList + 2*sqrt(vrList)
lo = mnList - 2*sqrt(vrList)

lines(rates,up/10,col="red")
lines(rates,lo/10,col="blue")
```

Exercise 4: (This exercise is difficult) We can estimate the actual rate from the hitting times of the particles if the counter were perfect. The formula is to take the reciprocal of the average of the gaps. But if we apply the same formula for the imperfect counter, then we may get a wrong estimate of the true rate. We want to make a correction graph that may be used to correct the under estimate.
Explain why the following R program achieves this. You will need to look up the online help of `cumsum` and `diff`.

```
rates = seq(0.1,3,0.1)
avgUnderEst = c()
for(lambda in rates) {
   underEst = c()
   for(i in 1:50) {
      gaps = rexp(1000,rate=lambda)
      hits = cumsum(gaps)
      obsHits = c(hits[1], hits[gaps>1])
      obsGaps = diff(obsHits)
      underEst = c(underEst,1/mean(obsGaps))
   }
   avgUnderEst = c(avgUnderEst,mean(underEst))
}

plot(avgUnderEst,rates,ty="l")
```

Can you interpret the plot? ■

6.4 Bootstrap

Use of repeated sampling is very common in every walk of science to get an idea about sampling error (confidence intervals, error bars, standard errors etc).

Imagine the following situation. You have a data set. You perform some analysis to get some numbers as result. You want to have an idea about the sampling error of your result. The ideal way is by repeating the entire process (starting with *fresh* data collection each time). It is easy to imagine situations where you simply cannot afford to do this. What then?
One simple way out is as follows. Take the data that you have, estimate the distribution from it using any of the way discussed earlier, then simulate fresh data from this *estimated* distribution. This technique is called **bootstrapping**. Owing to the simplicity of the idea and its wide applicability, it is one of the most powerful tools in the repertoire of a statistician. Depending on how the distribution function is estimated, there are two types of bootstrap: parametric and nonparametric. Nonparametric bootstrap using empirical cdf to estimate the distribution is the most common form. We shall discuss a simple example next. More complex applications will follow later.
Astronomical data sets are often riddled with outliers (values that are far from the rest). To get rid of such outliers one sometimes ignores a few of the extreme values. One such method is the **trimmed mean**.

```
x = c(1,2,3,4,5,6,100,7,8,9)
mean(x)
mean(x,trim=0.1) #ignore the extreme 10% points from BOTH ends
mean(x[x!=1 & x!=100])
```

We shall compute 10%-trimmed mean of `Vmag` from the Hipparcos data set.

```
hip = read.table(
'http://astrostatistics.psu.edu/MSMA/datasets/HIP.dat',
head=T,fill=T)
hip = na.omit(hip)
attach(hip)
mean(Vmag,trim=0.1)
```

We want to estimate the standard error of this estimate. For ordinary mean we have a simple formula, but unfortunately such a simple formula does not exist for the trimmed mean. So we shall use bootstrap here as follows. We shall generate 100 resamples each of same size as the original sample.

```
trmean = sapply(1:100,
  function(dummy) {
    resamp = sample(Vmag,length(Vmag),replace=T)
    mean(resamp,trim=0.1)
  })
```

```
sd(trmean)
```

A smarter way to achieve the same thing is by using the `boot` package.

```
Vmag = rnorm(10)
library(boot)
result = boot(Vmag,function(d,w) mean(d[w],trim=0.1), 100)
result
plot(result)
```

7 Hints for selected exercises

1.

```
curve(dexp(x,rate=1),0,5,ylim=c(0,3),ylab='')
title(main=expression(paste("Expo(",lambda,") densities")))
curve(dexp(x,rate=2),0,5,add=T,col='red')
curve(dexp(x,rate=3),0,5,add=T,col='blue')

legend("topright",col=c('black','red','blue'),lwd=1,
legend=c(expression(paste(lambda,'=1 ')),
expression(paste(lambda,'=2')),
expression(paste(lambda,'=3'))))
```

2.

```
GC_M31 = read.table(
'http://astrostatistics.psu.edu/MSMA/datasets/GlobClus_M31.dat',
header=T)

KGC_M31 = GC_M31[,2]

kseq = seq(min(KGC_M31)-1, max(KGC_M31)+1, 0.25)

hist(KGC_M31, breaks=kseq, prob=T,
     main='',xlab='K mag',ylab='N',col=gray(0.5))
normfit_M31 = fitdistr(KGC_M31,'normal') ; normfit_M31
```

```
lines(kseq,dnorm(kseq,
                 mean=normfit_M31$estimate[[1]],
                 sd=normfit_M31$estimate[[2]]))
```

3.

```
pval = apply(tmp,2,function(x) shapiro.test(x)$p)
lambda[which.max(pval)]
```

3. The plot is not the graph of a function as it doubles back over itself. This is because if we find too low a count, then this could mean two things: either the number of hits is really too low, or the number is too high causing the detector to remain jammed most of the time. Remember that even an undetected particle can prolong the period of jamming.