3. Classical Univariate Statistics

3.1. Standard tests

Standard one– and two– sample Normal theory and non-parametric classical univariate tests are readily available in **R** and S-plus.

Many of these are *generic* functions and what is returned depends on the context, i.e. whether it is a one-sample or two-sample test depends on whether you give the function `t.test()` the names of one or two samples.

Example: (Data shoes in MASS library but note how to enter the data direct into vectors `A` and `B`)

```
> data(shoes)
> shoes
$A
 [1] 13.2 8.2 10.9 14.3 10.7 6.6 9.5 10.8 8.8 13.3
$B
 [1] 14.0 8.8 11.2 14.2 11.8 6.4 9.8 11.3 9.3 13.6
```

Or enter the data directly:

```
> A<- c(13.2, 8.2, 10.9, 14.3, 10.7, 6.6, 9.5, 10.8, 8.8, 13.3)
> B<- c(14.0, 8.8, 11.2, 14.2, 11.8, 6.4, 9.8, 11.3, 9.3, 13.6)
```

```
> t.test(A,B)

         Welch Two Sample t-test

data:  A and B
t = -0.3689, df = 17.987, p-value = 0.7165
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.745046  1.925046
sample estimates:
mean of x mean of y
    10.63     11.04
```

```
> t.test(A-B)

         One Sample t-test

data:  A - B
t = -3.3489, df = 9, p-value = 0.008539
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -0.6869539 -0.1330461
sample estimates:
mean of x
    -0.41

> t.test(A,B,paired)
Error in all(arg == choices) : Object "paired" not found
> t.test(A,B,paired=TRUE)

         Paired t-test

data:  A and B
t = -3.3489, df = 9, p-value = 0.008539
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.6869539 -0.1330461
sample estimates:
mean of the differences
                  -0.41
```

The full list of tests available is

```
binom.test     chisq.test    cor.test        fisher.test

friedman.test  kruskal.test  mantelhaen.test mcnemar.test

prop.test      t.test        var.test        wilcox.test

chisq.gof      ks.gof
```

And details can be found in `help()`.

Note that the results of each of these functions is ***an object*** and individual elements of these objects can be accessed separately by using a $ sign with `name-of-ofbject$name-of-element`:

```
> t.test(A-B)$p.value
[1] 0.00853878

> t.test(A-B)$conf.int
[1] -0.6869539 -0.1330461
attr(,"conf.level")
[1] 0.95
```

Again, a list of elements of each of these tests is given in the help system.

Some of these tests depend upon assumptions on the underlying distribution of the data and others do not. For example the t-test presumes data are normally distributed but the non-parametric Wilcoxon does not. Both of them can test whether the measures of location are the same for two samples or whether the measure has a specific value (e.g. 0) for one sample, but the t-test works in terms of the ***mean*** as the measure of location and the Wilcoxon uses the ***median*** as the measure.

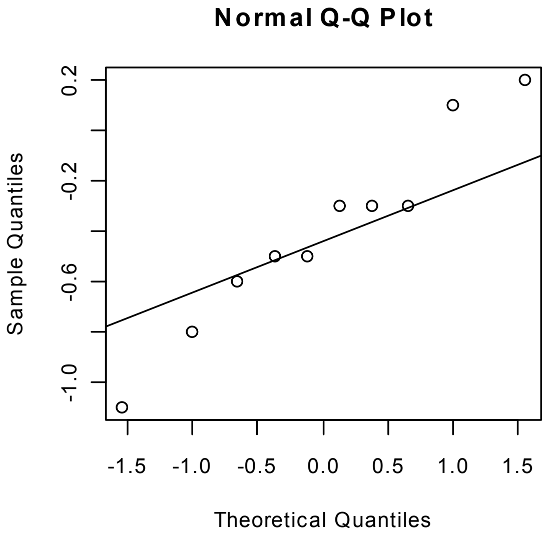
Not only is the median more resistant to outliers but the probability argument used to obtain the p-value is based on combinatorial arguments rather than one assumptions about probability distributions of the data.

Example (shoe data again, paired test):

```
> t.test(A-B)

         One Sample t-test

data:  A - B
t = -3.3489, df = 9, p-value = 0.008539
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -0.6869539 -0.1330461
sample estimates:
mean of x
    -0.41

> wilcox.test(A-B)

         Wilcoxon  signed  rank  test  with  continuity
correction

data:  A - B
V = 3, p-value = 0.01431
alternative hypothesis: true mu is not equal to 0

Warning message:
Cannot compute exact p-value with ties in: wilcox.test(A
- B)
```

Note that the Wilcoxon returns a larger p-value than the t-test, this is largely because the t-test is assuming more about the data and so you 'get more out of the analysis' (more in ⇒ more out). This is fine provided the assumptions made for the t-test are sensible. It is of course possible to check some of the assumptions (e.g. using a normal probability plot for checking normality) but with small samples it is difficult to detect non-normality.

```
> qqnorm(A-B)

> qqline(A-B)
```



This plot suggests that there are at least doubts about normality for these data.

One solution is to use only ‘non-parametric’ methods, but even these make some assumptions.

An alternative is to use permutation methods or simulation techniques, some of which come under a general heading of Monte Carlo Methods. **The Bootstrap** is a particular form of simulation technique.

3.2 The Bootstrap

3.2.1. Introduction

Suppose we have estimated the median by the sample median from a set of data and want to know how variable this estimate is. If we knew what the true density was then we could *simulate* more samples from the same density, taking samples of the same size as the one we have, calculate the median of each and then see how variable our answers were in each of these separate simulated samples.

However, we don't know what the true distribution is. So, either we have to estimate it (e.g. fit a normal distribution) or we have to find some other estimate. It might be possible to use a kernel density estimate but then simulating from this might be complicated. The is a much simpler estimate of the distribution and that is ***the sample itself.***

Specifically, if we have a sample $x_1,\ldots,x_n$ from a distribution F(.) and we calculate the sample distribution function, $F_n(.)$ based on our sample, we can then use $F_n(.)$ directly to generate more samples from our 'best estimate' of the unknown F(.). In fact this is just the same as taking a random sample of size n, ***with replacement***, from our actual data set $x_1,\ldots,x_n$. This may look very strange but it is a very powerful technique and with the use of the **R** function `sample(….., replace=TRUE)` it can be done very easily.

3.2.2 Simple Simulation

First, we give an illustration of the basic idea of estimation by simulation.

Suppose we take a sample of size 20 from a Normal distribution N(5,2.7^2), i.e. mean 5 and standard deviation, calculate the sample mean and then want to calculate a 95% confidence interval for the true mean. The standard way of doing this is to use classical distributional theory and say the 95% confidence interval is given by

$$\overline{x} \pm t_{19}(0.975)s/\sqrt{n}$$

where s is the sample standard deviation and $t_{19}(0.975)$ is the two-sided 95% point of a t-distribution on 19 d.f. (which is 2.093, but can be calculated directly in **R** as `> qt(0.975,19);[1] 2.093024`

```
> x<- rnorm(20,mean=5,sd=2.7)
> mean(x)
[1] 4.75921
> var(x)
[1] 7.10922
> confupper<-mean(x)+qt(0.975,19)*sqrt(var(x)/length(x))
> conflower<-mean(x)-qt(0.975,19)*sqrt(var(x)/length(x))
> confinterval<- c(conflower,confupper)
> confinterval
[1] 3.511338 6.007082
```

This gives the 95% confidence interval based on our sample as (3.511,6.007)

However if we could do a practical experiment to see how variable our estimate of the means is. If we 'simulate' our particular sample by taking lots of similar samples from N(5,2.7^2) and calculate the mean of each of them, then we can see experimentally what the range of values they have. We could then estimate a confidence interval by taking a range which includes 95% of our simulated values. We would not have used the sample standard deviation nor the t-distribution, nor any of the mathematical theory involving the t-distribution.

```
> simulate <-numeric(100)
> for (i in 1:100) simulate[i]<-mean(rnorm(20,mean=5,sd=2.7))
> z<-sort(simulate)
> z
  [1] 3.683628 3.876367 3.901581 3.977876 4.059417 4.084968 4.103193 4.127214
  [9] 4.145423 4.151252 4.151807 4.158069 4.197285 4.220176 4.250503 4.393266
 [17] 4.467750 4.500474 4.521293 4.566604 4.574698 4.613527 4.631142 4.652405
 [25] 4.684332 4.696737 4.699455 4.707846 4.720326 4.732183 4.737479 4.754224
 [33] 4.761708 4.828722 4.832087 4.848576 4.873959 4.897021 4.903855 4.919861
 [41] 4.926004 4.936150 4.955231 4.962549 4.982843 4.985068 5.014348 5.025469
 [49] 5.060322 5.068560 5.070019 5.078719 5.081987 5.086046 5.097905 5.134257
 [57] 5.148494 5.156743 5.162649 5.177213 5.184232 5.190236 5.216297 5.245337
 [65] 5.249008 5.276213 5.296429 5.308889 5.310640 5.316523 5.352162 5.357711
 [73] 5.372045 5.376082 5.423826 5.440574 5.448857 5.477199 5.484830 5.511197
 [81] 5.517907 5.585537 5.598260 5.657312 5.659240 5.662326 5.692230 5.709956
 [89] 5.714891 5.859206 5.926859 5.989662 6.009138 6.072194 6.139593 6.217246
 [97] 6.337749 6.383891 6.385220 6.449417
> z[3]
[1] 3.901581
> z[98]
[1] 6.383891
>
```

Then we could say that an approximate 95% confidence interval is given by (3.90, 6.39), — more precisely this is a 96% interval since 96% of our values lie inside it and 4% outside.

Computational notes:

1: note the declaration of the vector `simulate[.]` of length 100 using `numeric(100).`

2: note the construction of a simple loop with `for (i in 1:100)`, This can be notoriously slow in packages such as **R** and S-plus and advanced programmers would try to replace loops etc by matrix calculations (but I don't intend doing this here).

3: note that we do not need to store all the values in each simulated sample, we just need the mean of them.

4: note the use of `sort(.)`

Difficulty: In this example we ‘knew’ that our sample came from N(5,2.7^2) and we used this to simulate further samples. Of course we could never really know this and so the best we could do is to simulate from our best guess at the distribution, i.e. N($\overline{x}$,s^2), i.e. N(4.76, 2.67^2) since 4.76 and 2.66 were the mean and standard deviation of our original sample:

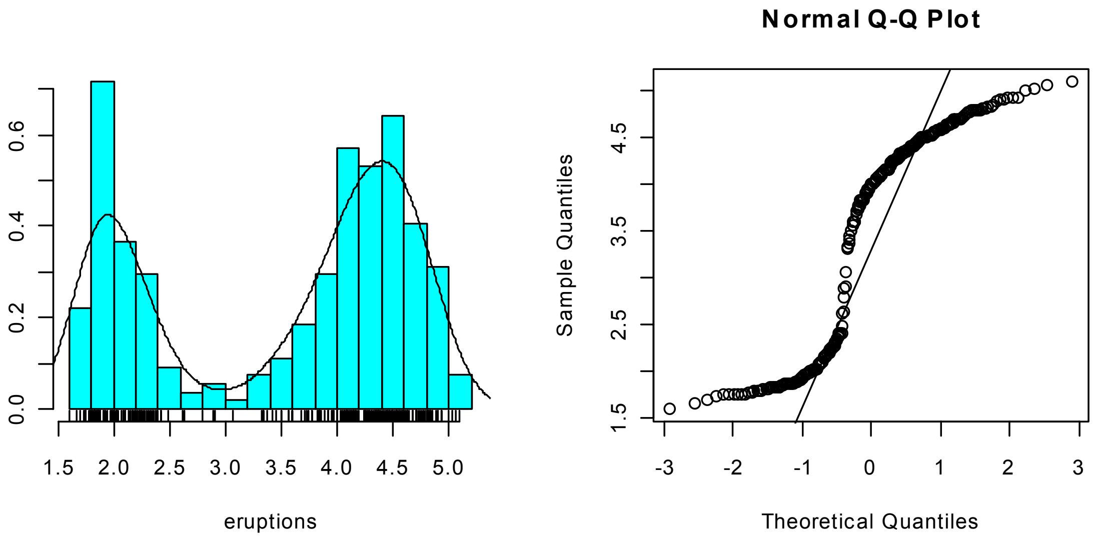
```
> simulate <- numeric(100)
> for(i in 1:100)simulate[i]<-mean(rnorm(20,mean=4.76,sd=2.67))
> z<-sort(simulate)
> z
  [1] 3.396327 3.502508 3.841749 3.870058 3.923347 3.969280 4.050996 4.071518
  [9] 4.110478 4.115113 4.163969 4.182597 4.185243 4.195840 4.277520 4.286212
 [17] 4.289088 4.295636 4.365233 4.374939 4.386417 4.404138 4.417440 4.425339
 [25] 4.460611 4.471175 4.471656 4.501675 4.510588 4.517224 4.528234 4.535420
 [33] 4.551477 4.552511 4.557144 4.557668 4.569633 4.573537 4.578601 4.582115
 [41] 4.583091 4.583715 4.598089 4.599501 4.609323 4.613675 4.666297 4.671486
 [49] 4.671580 4.679536 4.681736 4.721742 4.723026 4.734635 4.738312 4.738548
 [57] 4.796185 4.800435 4.800466 4.874767 4.887359 4.891548 4.893593 4.899882
 [65] 4.908909 4.925140 4.935247 4.964652 4.970336 4.970521 4.992720 5.080825
 [73] 5.081477 5.091885 5.106060 5.110453 5.129278 5.154696 5.159185 5.184879
 [81] 5.197338 5.206724 5.229970 5.256769 5.271497 5.304417 5.348681 5.356202
 [89] 5.364906 5.376544 5.411127 5.441553 5.545112 5.605137 5.680706 5.789276
 [97] 5.855591 5.902711 5.946993 6.147668
> z[3]
[1] 3.841749
> z[98]
[1] 5.902711
```

This gives an estimated confidence interval of (3.84, 5.90) — not very different from our previous estimates, in fact slightly closer to the ‘true answer’ of (3.511,6.007) but this is just an accident.

Difficulty: Although we estimated the mean and variance from our sample, we still ***assumed*** that our data came from a Normal distribution. Of course, we can test this and in the simple case we had above it might seem reasonable, but in other cases it we might know that a Normal distribution was not sensible and we might have no idea of a sensible distribution to use in simulation.

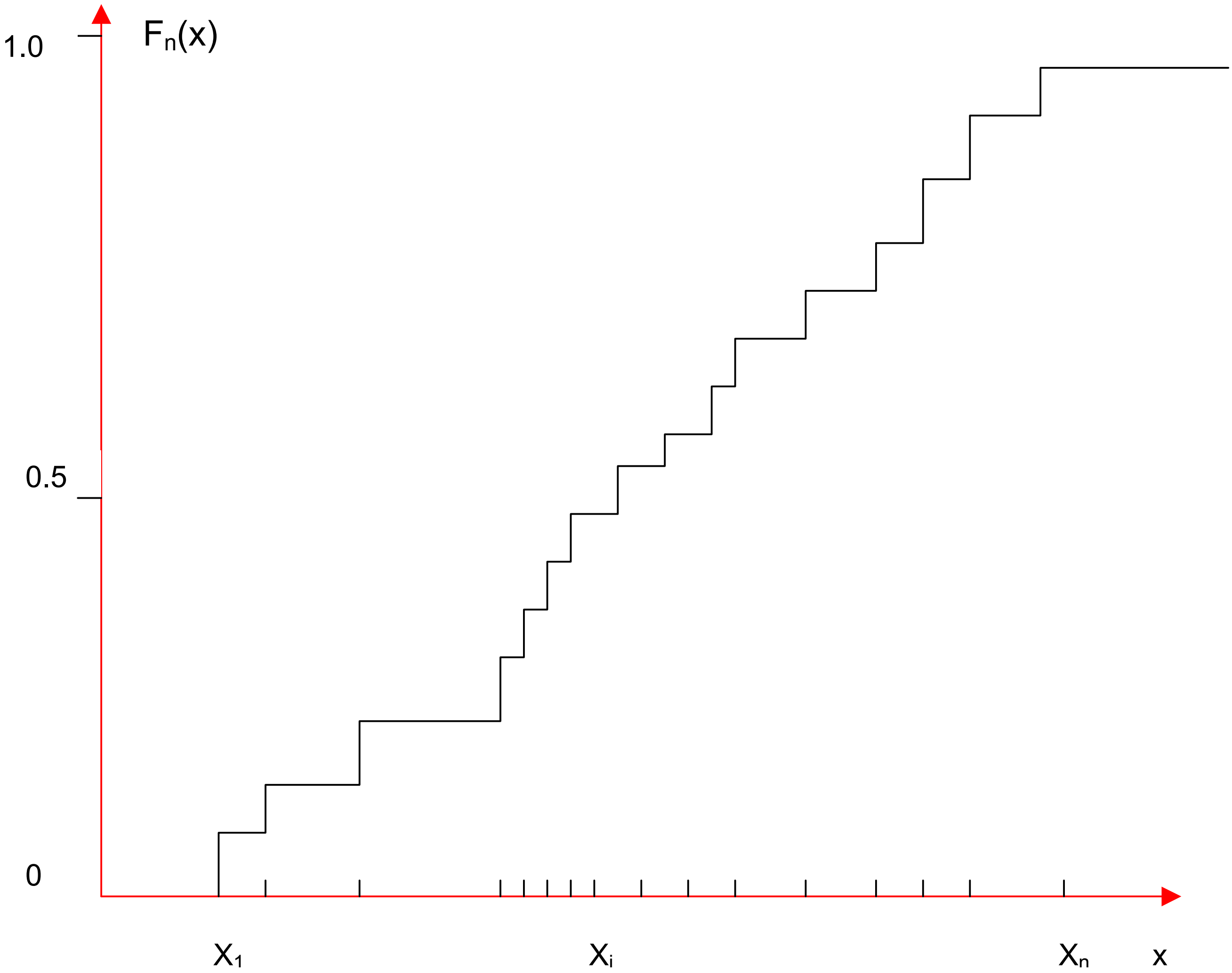
Consider again the problem of estimating the median of the eruption durations of Old Faithful. We have already seen that the distribution is bimodal and so cannot possibly be Normal of any sort but here is how we would check:

```
> data(faithful)
> attach(faithful)
> par(mfrow=c(2,2))
> library(MASS)
> truehist(eruptions,nbins=15)
> rug(eruptions)
> lines(density(eruptions,adjust=0.7))
> qqnorm(eruptions)
> qqline(eruptions)
```



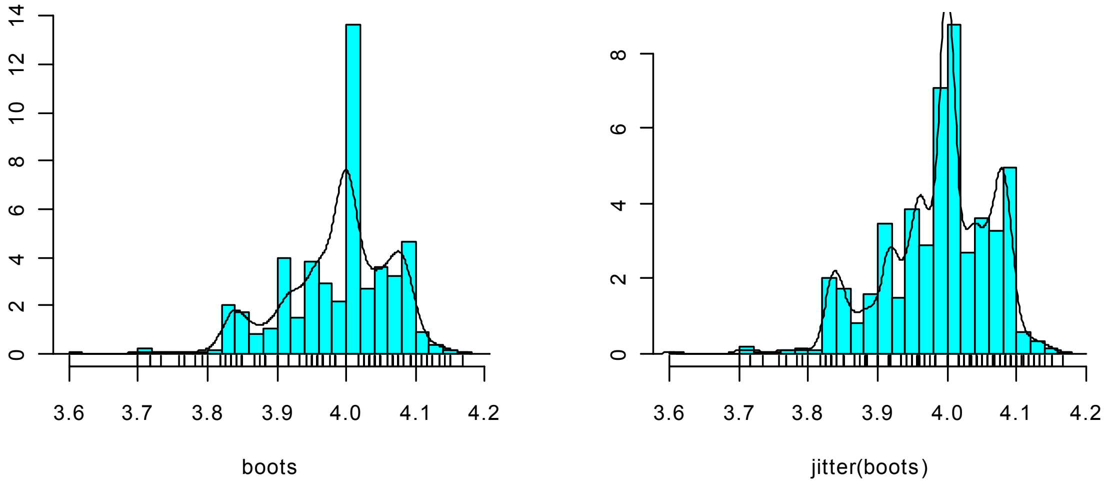
3.2.3 Bootstrap Simulation

We cannot possible pretend that the distribution of the eruption durations is normal so if we want to simulate samples that are like the actual sample we need another distribution. Now the simplest estimate we have of the 'true' distribution of eruption durations is given by the sample itself, i.e. by the sample distribution function $F_n(x)$



If we sample from $F_n(x)$ this is equivalent to taking a sample ***with replacement*** from our original observations.

```
> data(faithful)
> attach(faithful)
> set.seed(137)
> help(numeric)
> boots<- numeric(1000)
> for (i in 1:1000) boots[i]<-
             median(sample(eruptions,replace=TRUE))

> mean(boots-median(eruptions))
[1] -0.013551
> sqrt(var(boots))
[1] 0.07662417
> truehist(boots)
> lines(density(boots))
> rug(boots)
> truehist(jitter(boots))
> lines(density(boots,adjust=0.7))
> rug(jitter(boots))
```



This shows that the bias of the *bootstrap estimate* is –0.0.136 (i.e. quite small) and the estimated standard error is 0.077, quite close to the kernel density estimate of 0.078, which was based in part on a Normal distribution.

Computational notes

1: `Set.seed(137)` chooses the **'seed'** of the random number generator as 137. This means that I can get precisely the same bootstrap sample again if I set the seed to be 137. If I were to set the seed to another number then I would get a different sample and so a different estimate at the end. For example:

```
> set.seed(731)
> for (i in 1:1000) boots[i]<-
                median(sample(eruptions,replace=T))
> mean(boots-median(eruptions))
[1] -0.0180855
> sqrt(var(boots))
[1] 0.08026924
```

— slightly different but not enough to be of practical importance.

2: Note the use of `jitter` in drawing the histogram, there were clearly problems of observations being exactly on the class boundary (not surprising since we know the median is 4) and the use of the jitter makes the histogram a better *density estimate* — if we just wanted to display the actual data then the use of jitter would not be statistically justified (and we should use stem anyway).

3.2.4 Other Types of Bootstrap

The bootstrap sampling used above took the sample distribution function $F_n(x)$ as an estimate of the 'true' distribution function F(x). This is a very 'rough' estimate and it makes intuitive sense to use a smoother one, i.e. to use a ***Smooth Bootstrap.*** We can do this by adding a small amount to each sampled value, rather like using jitter(.).

The procedure used in the second set of simulations illustrating the simple simulation technique (i.e. when we presumed the underlying distribution was Normal but estimated the mean and variance from our sample) is sometimes known as a ***Parametric Bootstrap***.

The general ideas of bootstrapping are very powerful and very widely used for statistical analyses that do not depend very much on assumptions that are difficult to verify. They are one of the techniques that were initially called ***computer intensive*** but now these are becoming so routine and ordinary that this term is becoming old-fashioned.

3.3 Randomization and Permutation Tests

When W.S. Gosset (who was also known as ‘Student’) first derived the t-distribution he did not actually work it out as a result of assuming that the original data were Normally distributed but from a different argument.

Consider the problem of comparing the means of two samples A and B. Each of the observations is labelled either A or B. If the null hypothesis that there is no difference between the two is samples is true then these labels are entirely random. This means that the true distribution of a test statistic (such as the t-statistic) could be assessed by considering random re-labelling of each observation.

We could do this by experiment (or a type of simulation) by doing the following:

Step 1: calculate our two-sample test statistic, t_{obs} say.

Step 2: randomly label each observation as either A or B (keeping the sample sizes the same) and then calculate the same test statistic for comparing all the A observations with the B observations, getting a value t_1 say.

Steps 3, 4,…, 1001: repeat step 2 for a total of a 1000 times getting a thousand values t_1, t_2, …, t_{1000} .

Final step: compare our observed value t_{obs} with the simulated values. If there is no difference between the original samples A and B then the labels are arbitrary and so our t_{obs} will not look unusual amongst the simulated t_1, t_2, …, t_{1000}. However, if our value t_{obs} is amongst the most extreme 5% then we would have evidence (at the 5% level) that there really was a difference between the samples. Specifically, if we order the values so that $t_{(1)}<t_{(2)}<\ldots<t_{(1000)}$ then we would reject the hypothesis that the two samples were the same if either $t_{obs}<t_{(25)}$ or if $t_{obs}>t_{(975)}$.

What Student showed was that if you consider the theoretical distribution of the randomly re-labelled t-values then this was very well approximated by the 'student t-distribution'. [The mathematics involved much the same approximations and limits as are involved in proving the Central Limit Theorem]. It was only later that it was shown that you could get the same result by assuming that the observations were Normally distributed.

Anyway, it is now easy to perform these tests empirically and so avoid either the assumption of Normality or the inaccuracy of the approximations (whichever approach you use). Many packages (not just **R** and S-plus) now offer this facility for many tests, e.g. in SPSS you will find it under Options and Monte Carlo.

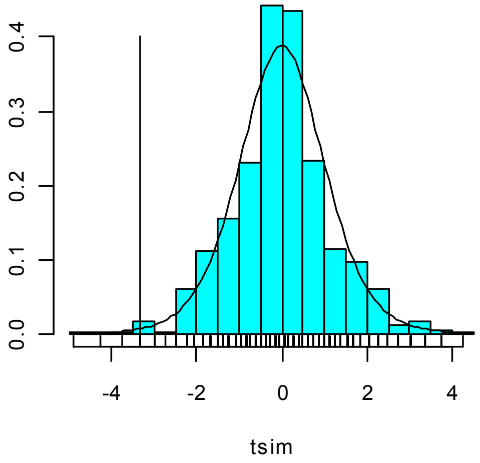
Sometimes, the sample is so small that you can consider all possible relabellings, in which case you just need to calculate the statistics for each labelling once and then the resulting test is know as a ***permutation test*.** Otherwise, it is known as a ***randomization test***.

Example: Paired t-test

Consider the shoes example and consider it is a paired t-test. There are 10 pairs and the paired t-test is just the same as a one-sample test on the differences that the mean is zero. If we randomly relabel each pair as either A-B or B-A then the numerical values of the differences in values stays the same, it is just the sign that is changed (with probability 0.5). In fact there are only 2^{10}=1024 different possibilities so it is practical to consider a permutation test but here we will do it by simulation.

To do this in **R** we will first define a *function* to calculate the t-statistic which is $t_{sim} = \frac{\bar{x}}{\sqrt{var(x)/n}}$. Then to change the signs of the differences randomly we will use the R function `sign()` which is either +1 or –1 according to whether the argument is positive or negative, together with a random Uniform(0,1) number which is generated by the function `runif()`, subtracting 0.5 from it. Note that `sign(runif(10)-0.5)` will produce a vector of length 10 consisting of +1 or –1 with probability 0.5.

```
> data(shoes)
> attach(shoes)
> ttest<- function(x) mean(x)/sqrt(var(x)/length(x))
> d<- A-B
> d
 [1] -0.8 -0.6 -0.3  0.1 -1.1  0.2 -0.3 -0.5 -0.5 -0.3
> ttest(d)
[1] -3.348877
> t.test(A,B,paired=TRUE)

         Paired t-test

data:  A and B
t = -3.3489, df = 9, p-value = 0.008539
alternative hypothesis: true difference in means is not
equal to 0
95 percent confidence interval:
 -0.6869539 -0.1330461
sample estimates:
mean of the differences
                  -0.41

> tsim<- numeric(1000)
> ttest(d)
[1] -3.348877
> for(i in 1:1000)tsim[i]<-ttest(d*sign(runif(10)-0.5))
> truehist(tsim)
> rug(tsim)
> z<- seq(-4,4,0.1)
> lines(z,dt(z,9))
> tobs<-ttest(d)
> markx<- c(tobs,tobs)
> marky<- c(0,0.4)
> lines(markx,marky)
> tsorted<-sort(tsim)
> tsorted[1:10]
 [1]  -4.920934  -4.258442  -3.753745  <  tobs  <-3.348877  -
3.348877 -3.348877 -3.348877
 [8] -3.348877 -3.011905 -3.011905
```



The picture above gives a histogram of the simulated values (and note from the rug plot how few distinct values there are — partly this is because there are only 7 (not 10) distinct values of abs(d).

The vertical line towards the left of the plot marks our observed value of –3.35 and we can see that only **three** of our 1000 simulated values are less than this. We can thus reject the hypothesis that the means are equal at level 2×3/1000=.006, (note that the t-approximations gives a level of 0.0085, quite close as can be seen from the density of t_9 superimposed on the histogram).

3.4 Summary

This section has given a brief summary of classical univariate tests. The key ideas introduced are that

- many of these tests rely on assumptions which may be difficult to verify or may in fact be wrong (e.g. for bimodal data)
- tests involving sample means and variances are more at risk than those depending on medians and permutation arguments
- we can **simulate** similar samples to obtain estimates of quantities such as standard errors or p-values
- **bootstrapping** provides a way of making no assumptions about the distribution of the data at all (except independence!)
- **randomization and permutation** tests are easy to do and provide good protection. If they are available in your favourite statistics package (e.g. SPSS) then **USE THEM**, especially for small sample problems such as 2×2 tables and chi-squared tests.