

R : Copyright 2003, The R Foundation for Statistical Computing  
Version 1.8.1 (2003-11-21), ISBN 3-900051-00-3

R is free software and comes with ABSOLUTELY NO WARRANTY.  
You are welcome to redistribute it under certain conditions.  
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.  
Type 'contributors()' for more information and  
'citation()' on how to cite R in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or  
'help.start()' for a HTML browser interface to help.  
Type 'q()' to quit R.

-----  
TeXmacs interface. Use v() to insert current graphic into  
TeXmacs buffer. Use q("yes") to quit and save.  
-----

$x^2 + y^2$

	1	2
<hline>1	0.00	0.00
2	0.00	0.00
<hline>		

> 3

3

[1] 3

```
> 4
```

```
[1] 4
```

```
>
```

## Lecture 7 - Bootstrap and Jackknife

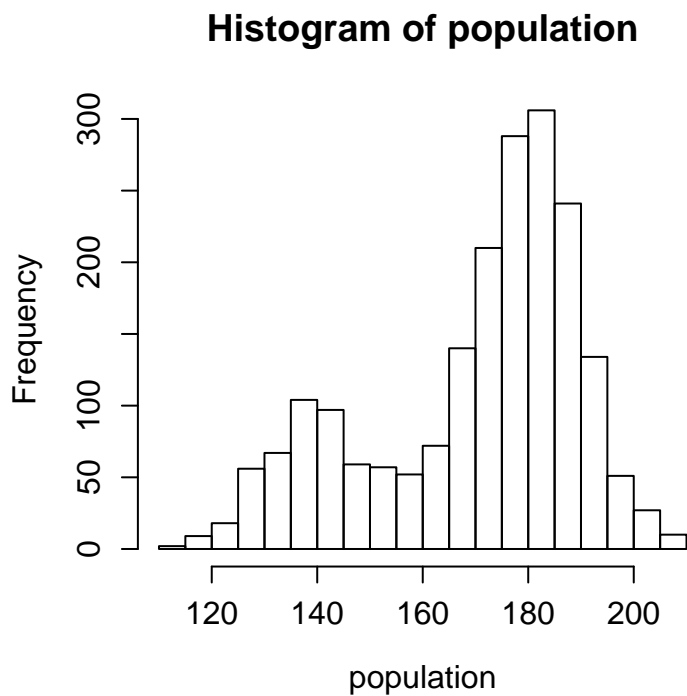
The principle of the bootstrap method is that the data that we gathered can represent the external world.

In order to get a distribution of possible values that we could have measured, we re-sample from the data.

### Example: Median

Let us say that we have a population of 2000 people, whose heights are the following:

```
> population=c(rnorm(1500,mean=180,sd=10),rnorm(500,mean=140,sd=10))
> hist(population,n=30);v()
```



```
>
```

Now let us assume that we measure the heights of 20 randomly chosen people exactly:

```
> measurements=sample(population,20)
> measurements
```

```
[1] 184.1049 186.0301 162.4606 176.1285 133.3787 135.8835 142.8405 175.5389
[9] 117.5551 187.8455 203.9851 141.7764 179.8468 176.3813 147.5747 119.2880
[17] 188.9668 173.2810 173.1390 184.8293
```

>

We want to know the median of the distribution. To approximate that, we measure the median of the measurements:

```
> mean(measurements)
```

```
[1] 164.5417
```

```
> @@@
```

```
[1] 164.5417
```

>

We know now the median of the measurements. How wrong are we?

If we had the whole population at our disposal, we could do the process of sampling many times, and see the distribution of the median:

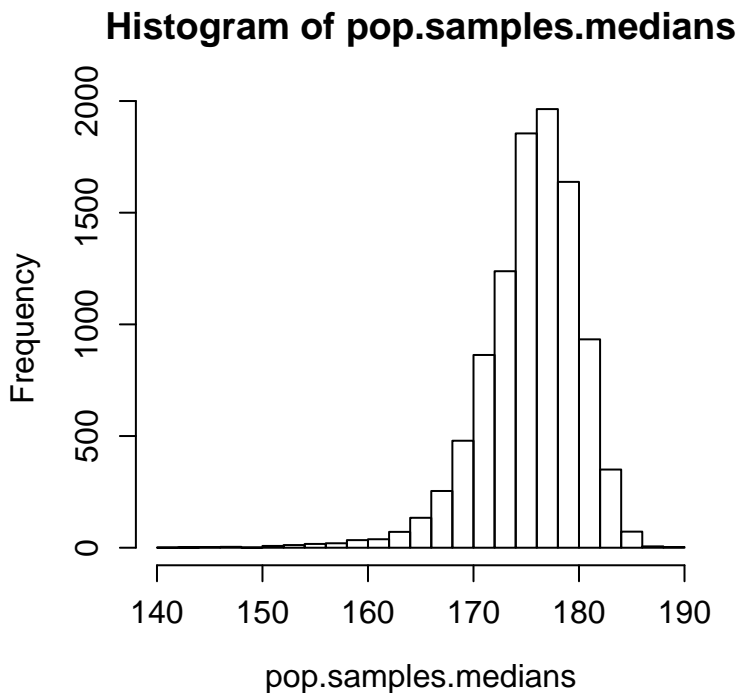
```
> pop.samples=t(sapply(1:10000,function(i) sample(population,20) ))
```

```
> dim(pop.samples)
```

```
[1] 10000    20
```

```
> pop.samples.medians=apply(pop.samples,1,median)
```

```
> hist(pop.samples.medians,n=30);v()
```



```
> median(population)
```

```
[1] 176.1048
```

```
> 2
```

```
[1] 2
```

```
>
```

We can see that we're likely to be about 10cm off the real median.

But we don't have the real population. We only have our samples. To estimate the distribution of medians, we sample from the sample, instead of from the real population:

```
> sample.samples=t(sapply(1:10000,function(i) sample(measurements,20,rep=T) ))
```

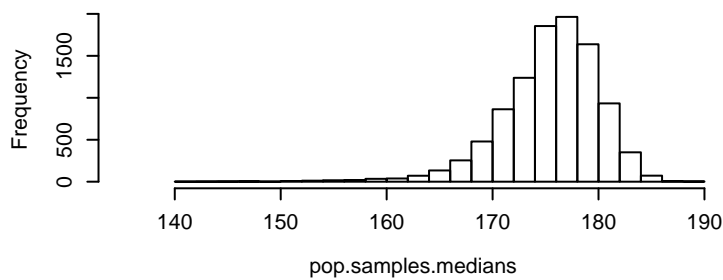
```
> sample.samples.medians=apply(sample.samples,1,median)
```

```
> layout(
```

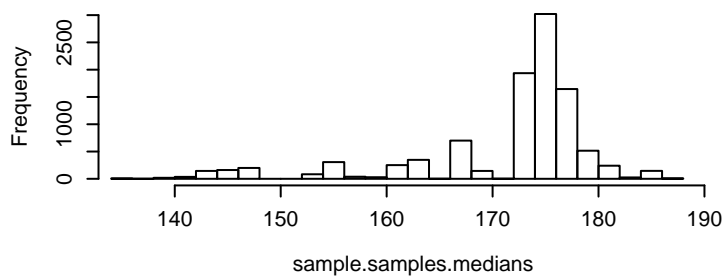
```
matrix(1:2,2,1));par(cex=0.7);hist(pop.samples.medians,n=30,xlim=c(135,190)
);hist(sample.samples.medians,n=30,xlim=c(135,190));v();layout(1)
```

```
+
```

**Histogram of pop.samples.medians**



**Histogram of sample.samples.medians**



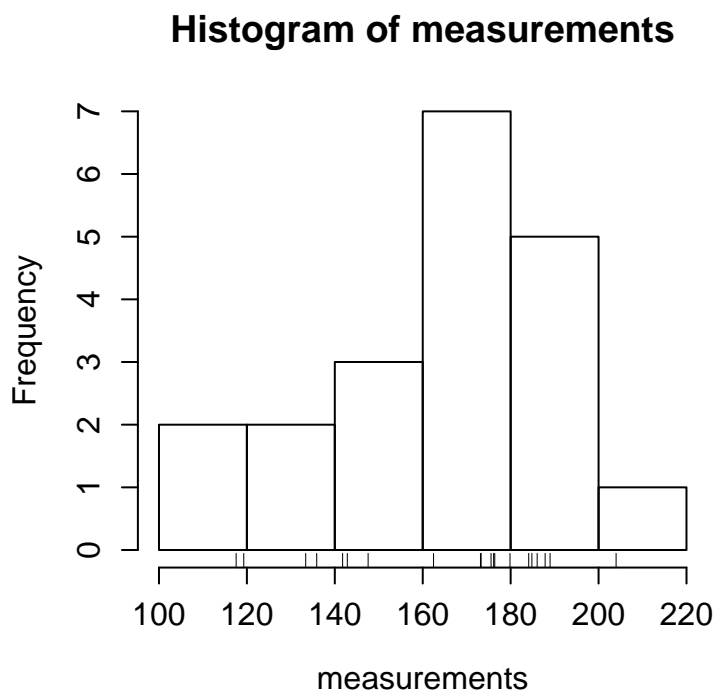
>

As you see, we do not get the same distribution! The first distribution was created by sampling from the real population. The second distribution was sampled from our sample.

## Parametric bootstrap

Instead of using the sample as our model for the world, we can base a model on the sample, and then sample from it. For example, let us look at our sample.

```
> hist(measurements);rug(measurements);v()
```



>

It obviously does not look normal - it has too much of a tail on the left.

We could still try to fit the best normal distribution we can find, and then bootstrap.

```
> a=measurements-mean(measurements)
```

```
> a=a/sd(a)
```

```
> ks.test(a,pnorm)
```

## One-sample Kolmogorov-Smirnov test

```
data: a
D = 0.2335, p-value = 0.2254
alternative hypothesis: two.sided
```

>

You can see that because of the small sample size, the KS test is actually not significant.

```
> mm=mean(measurements);mm
```

```
[1] 164.5417
```

```
> msd=sd(measurements);msd
```

```
[1] 25.19489
```

```
> parametric=matrix( rnorm(10000*20,mean=mm,sd=msd), 10000, 20)
```

```
> parametric.medians=apply(parametric,1,median)
```

```
> layout(
```

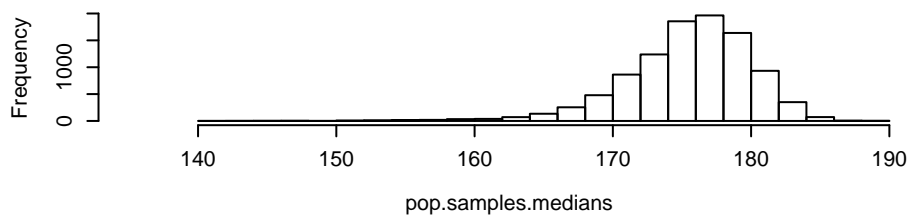
```
matrix(1:3,3,1));par(cex=0.7);hist(pop.samples.medians,n=30,xlim=c(135,190)
```

```
);hist(sample.samples.medians,n=30,xlim=c(135,190)
```

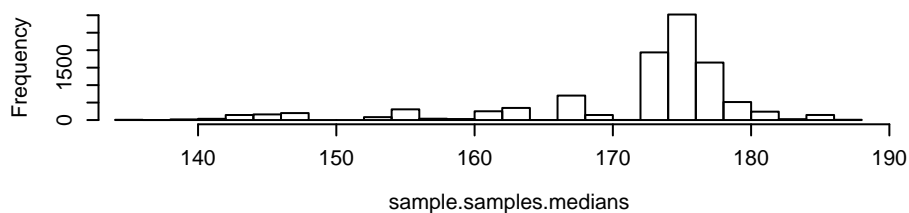
```
);hist(parametric.medians,n=30,xlim=c(135,190));v(width=5,height=5);layout(1)
```

+

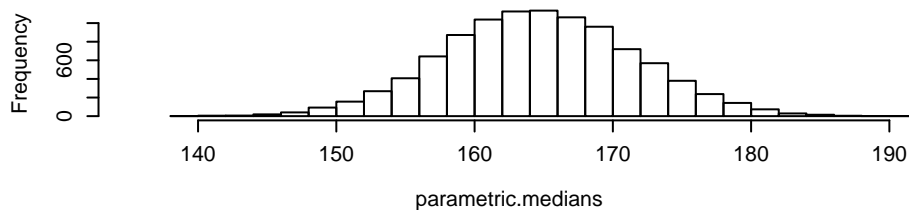
### Histogram of pop.samples.medians



### Histogram of sample.samples.medians



### Histogram of parametric.medians



```

> sd(sample.samples.medians)
[1] 8.75055
> sd(parametric.medians)
[1] 6.861419
> sd(pop.samples.medians)
[1] 4.687592
>

```

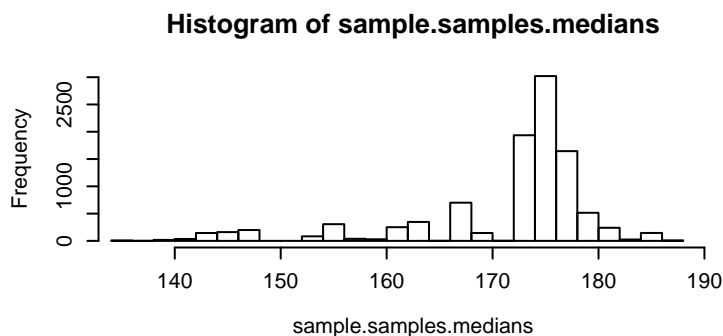
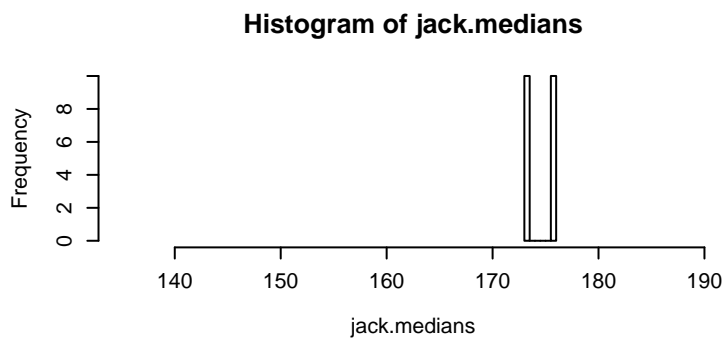
## Jackknife

The Jackknife method was invented before the bootstrap method, and is very similar. Instead of taking random samples from the distribution, we drop each of our sample points.

```

> jack= t( apply(1:length(measurements),function(i) measurements[-i] ) )
> jack[1:4,1:4]
      [,1]      [,2]      [,3]      [,4]
[1,] 186.0301 162.4606 176.1285 133.3787
[2,] 184.1049 162.4606 176.1285 133.3787
[3,] 184.1049 186.0301 176.1285 133.3787
[4,] 184.1049 186.0301 162.4606 133.3787
> jack.medians=apply( jack, 1, median )
> layout( matrix(1:2,2,1));par(cex=0.7);hist(jack.medians,xlim=c(135,190))
);hist(sample.samples.medians,n=30,xlim=c(135,190));v();layout(1)
+

```



>

We can see that the jackknife estimate of the error is too small. In fact it is too small by a factor of approximately  $\sqrt{20}$ , or  $\sqrt{(n-1)^2/n}$ .

```

> sd(jack.medians)

```

```

[1] 1.158305
> sd( jack.medians)*sqrt(20)
[1] 5.180097

```

It is actually clear that the median will move between 1 2 or 3 points at most:

```

> median( 1:3)
[1] 2
> median(1:2)
[1] 1.5
> median(c(1,3))
[1] 2
> median( 2:3)
[1] 2.5
>

```

And as we see in the above example, we get just 2 points.

```

> table( jack.medians )
jack.medians
173.280953077412 175.538905170459
          10          10
>

```

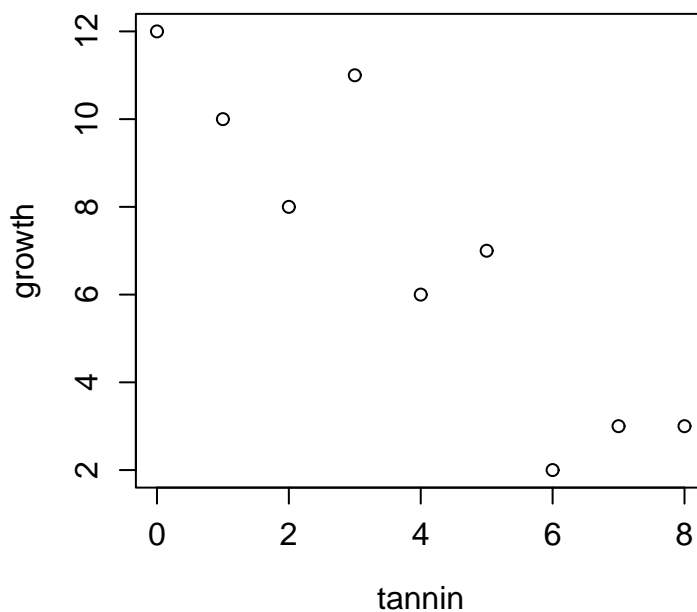
Because the median has this property that it “jumps”, the jackknife is a bad choice in this case.

Let us take another example:

```

> a=read.table("/home/dirk/data/regression.txt",head=T)
> plot(growth~tannin,data=a);v()

```



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

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

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

```
> names(reg)
```

```
 [1] "coefficients" "residuals"      "effects"        "rank"
 [5] "fitted.values" "assign"         "qr"             "df.residual"
 [9] "xlevels"      "call"          "terms"         "model"
```

```
> reg$coefficients[2]
```

```
      tannin
-1.216667
```

```
>
```

Let us say that we would like to know how exact this estimate of the slope is

First, let us write a function that given growth and tannin tells us the slope:

```
> slope=function(gr,ta){reg=lm(gr~ta);reg$coeff[2]}
> slope(a$gro,a$tan)
```

```
      ta
-1.216667
```

```
>
```

Now let us write a function that given indexes, calculates the slope for those points of data from a:

```
> index.slope=function(i) slope( a$gr[i], a$tan[i] )
```

```
> 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
```

```
> index.slope(1:9)
```

```
      ta
-1.216667
```

```
> index.slope(1:3)
```

```
      ta
      -2
```

```
> index.slope(-1)
```

```
      ta
```

-1.190476

>

Now the jackknife is very simple:

```
> jack.slopes=sapply(-(1:9), index.slope)
```

```
> jack.slopes
```

```
      ta      ta      ta      ta      ta      ta      ta
ta
-1.190476 -1.253133 -1.270270 -1.161359 -1.216667 -1.242038 -1.117117 -
1.200501
      ta
-1.321429
```

>

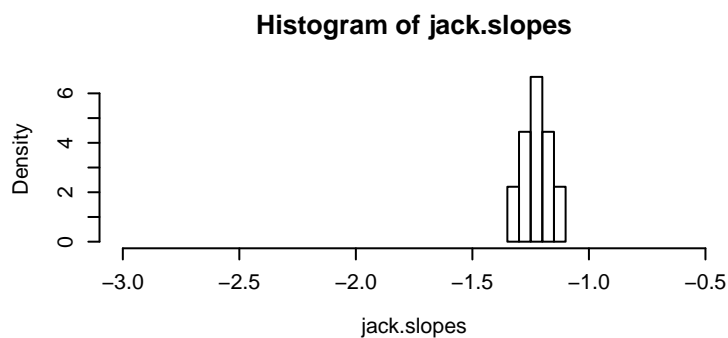
Bootstrap is a bit more complicated:

```
> boot=t( sapply(1:200,function(i) sample(1:9,rep=T) ) )
```

```
> boot.slopes=apply(boot,1,index.slope)
```

```
> layout(matrix(1:2,2,1));par(cex=0.7);hist(boot.slopes,prob=T,xlim=c(-3,-0.5))
```

```
> hist(jack.slopes,prob=T,xlim=c(-3,-0.5));v();layout(1)
```



```
> sd(boot.slopes)
```

```
[1] 0.1943785
```

```
> sd(jack.slopes)
```

```
[1] 0.06090926
```

```
> sd(jack.slopes)*sqrt(9)
```

```
[1] 0.1827278
```

>

## Confidence intervals

A general rule of thumb says that to estimate the error, 200 bootstrap samples are enough. To estimate 95% confidence intervals, one needs around 1000.

```
> boot=t( sapply(1:1000,function(i) sample(1:9,rep=T) ) )
> boot.slopes=apply(boot,1,index.slope)
> quantile(boot.slopes,0.025)
      2.5%
-1.609121
> quantile(boot.slopes,0.975)
      97.5%
-0.9084386
>
```

There are actually more accurate methods for calculating confidence intervals using quantiles. It can be done with the function `bcanon` in the library `bootstrap`:

```
> library(bootstrap)
> bcanon(1:9,1000,index.slope)
$confpoints
  alpha bca point
[1,] 0.025 -1.6774194
[2,] 0.050 -1.5909091
[3,] 0.100 -1.4782609
[4,] 0.160 -1.4000000
[5,] 0.840 -1.0920330
[6,] 0.900 -1.0520833
[7,] 0.950 -1.0061728
[8,] 0.975 -0.9530201

$z0
[1] -0.06521854

$acc
[1] -0.001202818

$u
[1] -1.190476 -1.253133 -1.270270 -1.161359 -1.216667 -1.242038 -1.117117
[8] -1.200501 -1.321429

$call
bcanon(x = 1:9, nboot = 1000, theta = index.slope)

Warning message:
multi-argument returns are deprecated in: return(confpoints, z0, acc, u, call
= call)
>
```

## Permutation tests

Permutation tests are very similar to bootstraps. They test if two distributions are the same:

```
> x=rnorm(10)
```

```
> y=rnorm(20)+1
> mean(x)-mean(y)
[1] -0.7245577
>
```

In a permutation test, we permute the data, and ask how often it has such a difference between the samples:

```
> perm=t( sapply( 1:1000, function(i) sample( c(x,y) ) ) )
> perm.diff = apply(perm, 1, function(xy){ mean(xy[1:10])-mean(xy[11:30]) } )
> sum( abs(perm.diff) > abs((mean(x)-mean(y))) )
[1] 36
> 36/1000
[1] 0.036
>
```

So this permutation test sees the data as significantly different

```
> t.test(x,y)$p.value
[1] 0.07868077
>
```

The permutation test tests the hypothesis that the two distributions are the same.

With a bootstrap test, we could do a very similar test, except that we sample **with replacement**.