Chapter 1

Simulation

1.1 Simulation - what is it?

Simulation is a way of generating pseudo random numbers based on a numeric algirithm. If the algorithm is good enough we will in practice not see a difference compared to real randomness. Pseudu random numbers are most often generated as independent random observations from a uniform distribution between 0 and 1. Based on this we can generate random numbers from any distribution.

We have ealier used the function **sample** to get a subset of a data set. A random sample is taken meaning that each observation have the same probability to be chosen. It can also be used to do simulation.

1.2 Generate random samples

What about throwing a die. We use replace=T because the throws are independent of each other.

```
die<-c(1:6)
die

[1] 1 2 3 4 5 6

sample(die,1) # one throw

[1] 5

sample(die,1) # another throw

[1] 3

y<-sample(die,size=15,replace=T) # result of 15throws
y

 [1] 4 6 5 6 3 1 1 3 2 2 4 2 3 2 4
```

Now we have a chance to calculate the probability of events. What is the proability to get a sum larger than or equal 12 with 3 throws. We can as above get the result of 3 throws and calculate the sum. But that is just one observation. We need a large samle but here we use a small example of 5 observations. To calculate the probability more exact we can e.g. set n=10000.

```
set.seed(1001)      # explained further down
n<-5
throws<-sample(die,size=3*n,replace=T)
throws
```

```
 [1] 3 3 6 6 4 4 4 4 4 5 5 6 3 2 2
```

```
y<-data.frame(matrix(throws,ncol=3))
# using matrix to specify dimensions
# data is stored by column

y         # Each row is three throws with the die.
```

```
  X1 X2 X3
1  3  4  5
2  3  4  6
3  6  4  3
4  6  4  2
5  4  5  2
```

```
ys3<-y[,1]+y[,2]+y[,3]
ys3
```

```
[1] 12 13 13 12 11
```

```
ys3b<-apply(y,MARGIN=1,FUN=sum)  # However, this may be a better alternative
ys3b
```

```
[1] 12 13 13 12 11
```

```
# repetition of sam commands but large size calculation (n)
n<-10000
y<-matrix(sample(die,size=3*n,replace=T),ncol=3)
str(y)
```

```
 int [1:10000, 1:3] 4 3 4 1 2 6 6 3 5 6 ...
```

```
ys3b<-apply(y,MARGIN=1,FUN=sum)
str(ys3b)
```

```
 int [1:10000] 16 13 12 7 10 13 13 8 9 15 ...
```

```
# the proportion of TRUE is the probability we wanted to calculate
table(ys3b>=12)/n
```

```
 FALSE   TRUE
0.6213 0.3787
```

```
# or similarly
prop.table(table(ys3b>=12))
```

```
 FALSE   TRUE
0.6213 0.3787
```

If we repeat the same simulation the result will not be the same as the first time although the simulation is made similarly. To ensure we can get the same result every time we can use **set.seed**

```
set.seed(1001)
sample(die,size=10,replace=T)
```

```
 [1] 3 3 6 6 4 4 4 4 4 5
```

```
sample(die,size=10,replace=T)

 [1] 5 6 3 2 2 4 3 4 1 2

set.seed(1001)
sample(die,size=10,replace=T)

 [1] 3 3 6 6 4 4 4 4 4 5

set.seed(1001)
```

Now let us look at the statistician's favorite; the urn with balls in different colors. Assume an urn with 10 black, 5 white and 7 red balls. Let us simulate a randomly chosen sample of 5 balls from the urn. This is an example of sampling without replacement.

```
set.seed(1001)
urn<-c(rep("black",10),rep("white",5),rep("red",7))
urn

 [1] "black" "black" "black" "black" "black" "black" "black" "black" "black" "black"
[11] "white" "white" "white" "white" "white" "red"   "red"   "red"   "red"   "red"
[21] "red"   "red"

sample(urn,size=5,replace=F)  # Observe replace=F

[1] "black" "white" "red"   "black" "red"

sample(urn,size=5,replace=F)  # another sample

[1] "white" "black" "white" "black" "white"
```

1.3 Generate observations with common distributions like normal, uniform etc.

We now take a look at the simulation functions for the most common distributions; uniform, normal and binomial. Observe that very similar functions can be used for calculation of the density, distribution and quantiles of the chosen distribution depending on the first letter. The ranomisation function names all start with r. Uniformly distributed values is generated by **runif, rnorm** and **rbinom**, respectively.

```
set.seed(1001)
runif(5,min=0,max=1) # uniform distribution

[1] 0.9856888 0.4126285 0.4295392 0.4191722 0.4265066

x<-runif(1000,min=-2,max=2)

rnorm(5) # per default standard normal distribution

[1] -1.7513564  0.8658837 -1.9754660 -1.7968151 -0.1928661

y1<-rnorm(1000)
summary(y1)

     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-3.204607 -0.671588 -0.009193 -0.002227  0.633417  3.444678
```

```r
sd(y1)
```

```
[1] 1.016013
```

```r
y2<-rnorm(1000,mean=2,sd=0.1)
summary(y2)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.649   1.939   2.004   2.006   2.070   2.304
```

```r
sd(y2)
```

```
[1] 0.09871234
```

```r
# illustration of the number of times we get "six" when throwing a die 5 times.
# The   experiment is repeated 1000 times. The probablility of each throw=1/6
z<-rbinom(n=1000,size=5,p=1/6)
table(z)
```

```
z
  0   1   2   3   4
403 407 161  28   1
```

```r
# the estimated probabilities of the number of getting "6" in five throws
prop.table(table(z))
```

```
z
    0     1     2     3     4
0.403 0.407 0.161 0.028 0.001
```

```
par(mfrow=c(3,1))
hist(x,main="Uniform(2,2)",breaks=seq(-2.5,2.5,by=0.25))
hist(y1,main="Normal(0,1) i.e. standard normal",breaks=seq(-4,4,by=0.2))
hist(y2,main="Normal(2,0.1)",breaks=seq(1.5,2.5,by=0.025))
```

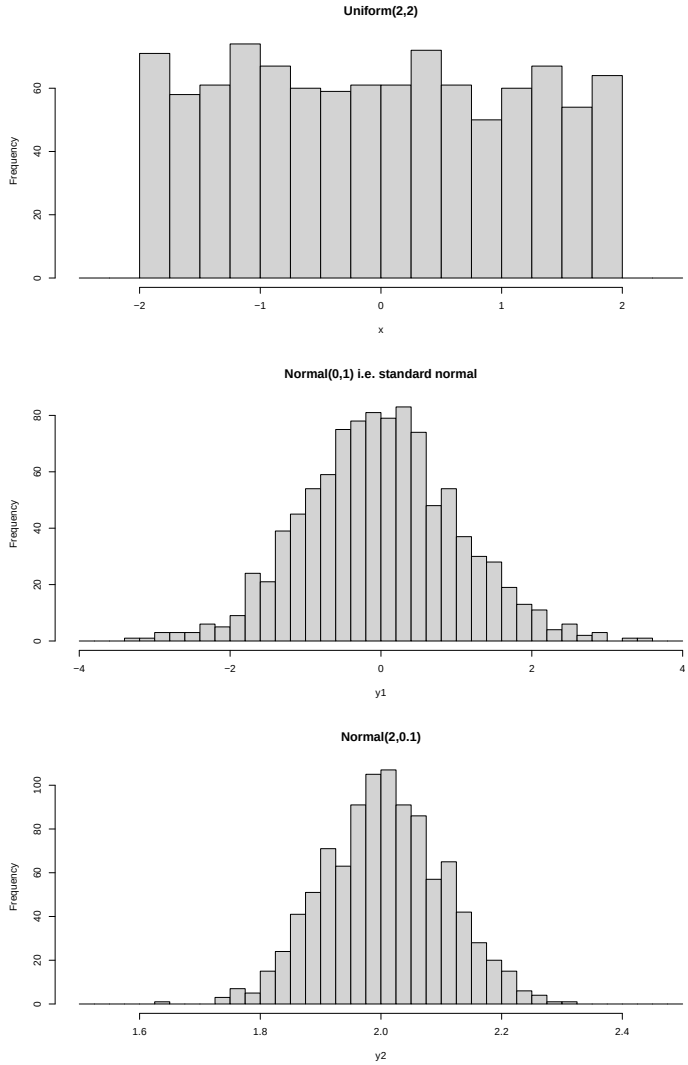

Figure 1.1: Histograms

1.4 Generating observations with other distributions

Random numbers can be generated for a large number of distributions. Similarly functions for the density/mass function, cumulative distribution function, quantile function can be found with corresponding names starting with d, p or q, respectively.

1.5 Bootstrapping

Bootstrapping is a metod where random sub-samples are drawn from the original data (with replacement) and estimations are repeted a number of times based on the subsamples. This enables information about the random variation of the results and calculation of for example confidence intervals is possible. We may do this using simulation and some programming but there is a function for this **boot::boot** which makes it more convenient. However, some preparations are necessary. The method is illustrated by an example using the **mtcars** dataset. The linear regression model $mpg = a + b * wt$ is estimated and we want to calculate a confidence interval for R^2, i.e. coefficient of determination. It describes how much of the total variation of the dependent variable that is explained by the model. The variables are described below. However, the variable disp is not used in this example.

mpg	Miles/(US) gallon
wt	Weight (1000 lbs)
disp	Displacement (cu.in.)

```
head(mtcars)

                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1

nrow(mtcars)

[1] 32

fit0<-lm(mpg~wt,data=mtcars)
summary(fit0)

Call:
lm(formula = mpg ~ wt, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-4.5432 -2.3647 -0.1252  1.4096  6.8727

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
wt           -5.3445     0.5591  -9.559 1.29e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.046 on 30 degrees of freedom
Multiple R-squared:  0.7528,Adjusted R-squared:  0.7446
F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

```r
set.seed(1001)
library(boot)

# This is a function to estimate the model and as result give the R-Squared
rsq <- function(formula, data, indices) {
  d <- data[indices,] # indices allows boot to select a specific sample
  fit <- lm(formula, data=d)
  return(summary(fit)$r.square)
}

# Example of the function rsq where all observations in the original data is used
# Compare with R-squared in the result (fit0) above
rsq(mpg~wt,data=mtcars,1:32)
```

```
[1] 0.7528328
```

```r
# bootstrapping with 1000 replications
results <- boot(data=mtcars, statistic=rsq,
                R=1000, formula=mpg~wt)

results
```

```
ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = mtcars, statistic = rsq, R = 1000, formula = mpg ~
    wt)


Bootstrap Statistics :
     original      bias    std. error
t1* 0.7528328 0.006813838  0.05781024
```

```r
results$t0  # this is the result from the original sample, compare R-squared
```

```
[1] 0.7528328
```

```r
# this is the simulated 1000 estimates of R-squared
length(results$t)
```

```
[1] 1000
```

```r
mean(results$t)-results$t0    # compare the bias in the printed result
```

```
[1] 0.006813838
```

```r
sd(results$t)     # compare the standard deviation in the printed result
```

```
[1] 0.05781024
```

```r
# there are 5 possible types of confidence intervals, the default is "all"
boot.ci(results, type="perc")
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = results, type = "perc")

Intervals :
Level     Percentile
95%   ( 0.6268,  0.8525 )
Calculations and Intervals on Original Scale
```

```
# compare with quantiles, it seems very similar
quantile(results$t,c(0,0.025,0.975,1))

       0%      2.5%     97.5%      100%
0.4901092 0.6323692 0.8521974 0.9048545
```

```
par(mfrow=c(1,1))
hist(results$t,breaks=seq(0.3,1,by=0.05))
```

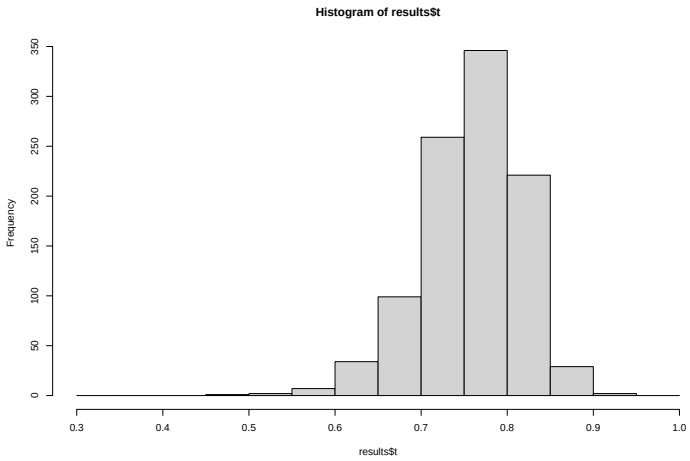

Figure 1.2: Histogram of estimated R-square