# Practical Selection Index

(UNE)

Multiple choice questions

1.

Information from relatives is most useful when

a. The heritability is high
b. The heritability is low
c. There is a lot of information known on each individual
d. One aims to avoid inbreeding
e. The environmental correlations among full sibs is high

2.

In the notes, the meaning of cov($X_i$,A) relates to

a. The degree of similarity of different selection candidates
b. The difference between family members
c. The similarity between a criterion and the breeding value to be estimated
d. The correlation between different traits

3.

Adding information from an extra selection criterion (Xj) to an existing criterion ($X_i$) will increase the accuracy of an EBV more when

a. This criterion is more similar to the other criterion: cov($X_i$, $X_j$) is high
b. This criterion is more different to the other criterion: cov($X_i$, $X_j$) is low
c. This criterion is more different from the breeding value (A): cov($X_i$, A) is low
d. The criterion is a measurement for a trait having a low correlation with A

4.

For an information sources that is a mean, the index weight is higher if

a. The additive genetic relationship of the criterion with the breeding value is lower
b. The criterion is based on a mean of many individuals
c. The criterion is a correlated trait
d. The heritability is lower

5.

The weighting for the progeny testing information ($2n/(n+\alpha)$)

a. Is larger when heritability is larger
b. Is smaller when heritability is larger

c. Has a maximum value of 1
d. Has a maximum value of 0.5

6.
The EBV of an animal is very high, but based on only very few progeny. It is

a. likely to drop as it gets more records on offspring
b. likely to increase as it gets more records on offspring
c. equally likely to drop than it is to increase

---

## Using relatives' information and selection index

### Exercise 1 Effect of using relatives' information on selection accuracy

Use the spreadsheet STEBVaccuracy.XLS (using the STSELIND tab) that is on Moodle. When open, click on 'enable content'.

Alternatively, use this app https://une-ers.shinyapps.io/MEBVaccuracy/

Use the following base parameters:

- heritability = 0.25
- repeatability = 0.25 (equal to heritability)
- common environmental correlation among full sibs ("c-squared") = 0.
  In the app, (put the "accuracy of the genomic test" to zero)

Note that with the spreadsheet, to obtain the answers to a set of new parameters, you'll have to press RUN.

Check the accuracy of the estimated breeding value for the following cases:

1) own performance record only
2) 10 own performance records. In which case would testing 10 records on the same genotype be a realistic option?
3) Increase the repeatability to 0.50. Explain the change in accuracy.

Now put the number of own performance records to 0, and put repeatability back to 0.25.

4) One performance record on dam (explain index weights and accuracy)
5) One performance record on both dam and sire (explain index weights and accuracy)

6) Give only the sire a large number of records (say 1000). This may seem unrealistic, but mimics the hypothetical situation that the EBV of the sire has a near perfect accuracy (How can the sire get a very large accuracy for his EBV?)
7) Give now also 1000 half sibs a record. How does the accuracy change? What is the maximum accuracy for having full information on sire and/or paternal halfsibs?
8) Explore the same situation with maximum information on both sire and dam, and ad many full sibs as well as half sibs. What is the maximum accuracy?
9) Add now information on 100 progeny. Does it help to have also information on collateral relives (=sibs) and/or parents?
10) Explain why information on 100 progeny could give a better estimate of an animals' breeding value than 100 own performance records.

## Matrix Commands in R

| | |
|---|---|
| Entering a vector | `X = c(1,6,2)` |
| Checking number of elements (length) | `length (x)` |
| Entering a matrix | `A=matrix (data=c(1,2,3,4) , nrow=2, ncol =2)` |

Note that we could omit typing "data=, nrow=, and ncol=" in the matrix() command above:
that is, we could just type

| | |
|---|---|
| | `A=matrix (c(1,2,3,4) ,2,2)` |
| checking by just typing | `A` |
| Or | `dim(A)` |
| Adding matrices: | `A+B` |
| Transpose: | `At = t(A)` |
| Multiplying matrices | `A %*% B` |

*Note that A * B is NOT a matrix multiplication*

| | |
|---|---|
| Inverse of A: | `AINV = solve(A)` |
| Subsets of matrices. Suppose we have | |
| | `A=matrix (1:16 ,4 ,4)` |
| one element | `d= A[2,3]` |
| two columns and two rows adjacent | `A1 = A[2:3 ,3:4]` |
| take 4 elements (2x2) not adjacent | `A2 = A[c(1,3) ,c(2,4) ]` |
| Combining matrices by columns | `C = cbind(A,At)` |
| By rows | `D = rbind(A,At)` |

The document summarizes R and R-Studio

- the basics
- reading data and checking it,
- plotting
- simple statistical analysis
- simulate data

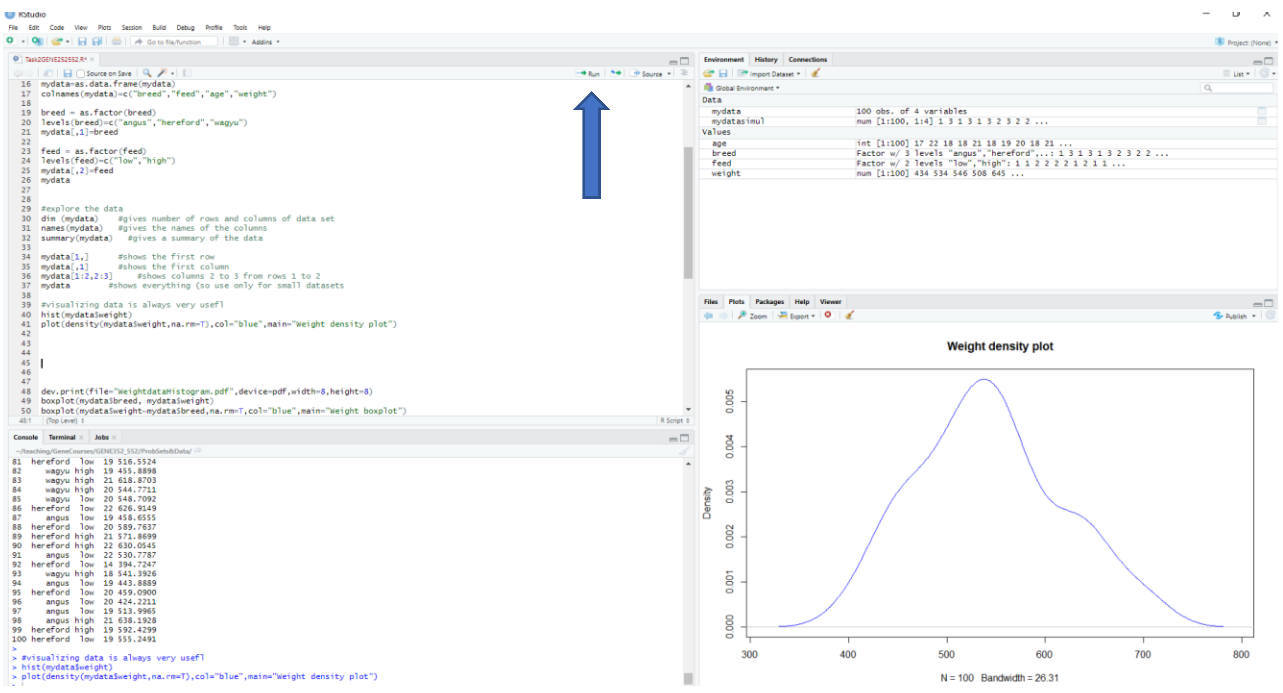
This will give you some great tools for data analysis, data simulation and ‘looking at data’. You should install R-Studio. This allows you to save instructions in a file, so you don’t have to remember them all. Many of these instructions can be copied from this practical. If you do a serious analysis, you should always work from a file, as that also gives you a record for later what you were doing.
So open a New File (working from R-Studio), and copy or type these instructions in that file. You can also try other instructions not listed here. A lot of things are possible in R and much can be worked out from either trial and error, or from suggestions from Google, or more specialized websites I have found this site very useful and easy to follow: http://www.countbio.com/index.html
But there are many more, e.g. https://bookdown.org/ndphillips/YaRrr/ (if you like pirates)

Once something works the way you want it, it is good to have a record, hence the use of programs. Initially it can be frustrating to make things work, but hopefully this practical will set you on course.

You will need the data set “`Task1Data.txt`” which can be downloaded from Moodle.

# Intro to R and R-Studio

It is easiest to work in R by setting up and interface program (integrated development environment - IDE)  called *R-studio*.
In R-studio you can put your instructions in one window, and run these, with the output in another window. This is a good way to develop code as you don't want to each time type in a whole string of instructions. An example is given below. In the left-top (LT) window you write your instructions and you can write a whole program (i.e. a whole string of instructions). If you highlight one or a number of lines, you can run these by clicking on the run button (see arrow), or use <Ctrl + Enter>. The left bottom (LB) window shows the results of these instructions. The LB window can be used to type in individual instructions, one by one. I often use this to try which instructions will work for a next step, once they work you can add them to your program.  You can also copy a number of lines from the LT window, paste them in LB and press <Enter> to execute them in the LB.
The right top (RT) window show the arrays that you have in memory, and if you click on them you can see detail appear in the LT window, the right bottom (RB) window is used for graphs and also for help. For example, if you type *?class* in the  LB window, the RB window will show you information on the topic "class". For a YouTube video, see
[https://youtu.be/EfzDGpmwagA?list=PLztamBUaO1aUzckPruc3NPizcKn-yKzow](https://youtu.be/EfzDGpmwagA?list=PLztamBUaO1aUzckPruc3NPizcKn-yKzow)



```r
mydata=as.data.frame(mydata)
colnames(mydata)=c("breed","feed","age","weight")

breed = as.factor(breed)
levels(breed)=c("angus","hereford","wagyu")
mydata[,1]=breed

feed = as.factor(feed)
levels(feed)=c("low","high")
mydata[,2]=feed
mydata


#explore the data
dim (mydata)    #gives number of rows and columns of data set
names(mydata)   #gives the names of the columns
summary(mydata)   #gives a summary of the data

mydata[1,]      #shows the first row
mydata[,1]      #shows the first column
mydata[1:2,2:3]     #shows columns 2 to 3 from rows 1 to 2
mydata        #shows everything (so use only for small datasets

#visualizing data is always very usefl
hist(mydata$weight)
plot(density(mydata$weight,na.rm=T),col="blue",main="Weight density plot")




dev.print(file="WeightdataHistogram.pdf",device=pdf,width=8,height=8)
boxplot(mydata$breed, mydata$weight)
boxplot(mydata$weight~mydata$breed,na.rm=T,col="blue",main="Weight boxplot")
```

```
81  hereford  low  19 516.5524
82     wagyu high  19 455.8898
83     wagyu high  21 618.8703
84     wagyu high  20 544.7711
85     wagyu  low  20 548.7092
86  hereford  low  22 626.9149
87     angus  low  19 458.6555
88  hereford  low  20 589.7637
89  hereford high  21 571.8699
90  hereford high  22 630.0545
91     angus  low  22 530.7787
92  hereford  low  14 394.7247
93     wagyu high  18 541.3926
94     angus  low  19 443.8889
95  hereford  low  20 459.0900
96     angus  low  20 424.2211
97     angus  low  19 513.9965
98     angus high  21 638.1928
99  hereford high  19 592.4299
100 hereford  low  19 555.2491
>
> #visualizing data is always very usefl
> hist(mydata$weight)
> plot(density(mydata$weight,na.rm=T),col="blue",main="Weight density plot")
```

```
Data
  mydata          100 obs. of 4 variables
  mydatasimul     num [1:100, 1:4] 1 3 1 3 1 3 2 3 2 2 ...
Values
  age             int [1:100] 17 22 18 18 21 18 19 20 18 21 ...
  breed           Factor w/ 3 levels "angus","hereford",..: 1 3 1 3 1 3 2 3 2 2 ...
  feed            Factor w/ 2 levels "low","high": 1 1 2 2 2 2 1 2 1 1 ...
  weight          num [1:100] 434 534 546 508 645 ...
```

Coding in R

In the following, the actual instructions are highlighted in yellow, the rest is just commenting on what you do. Commenting makes things a lot easier, for yourself later and for others, to understand. Comments are preceded by a hash " # " sign. You can comment in a separate line but also on the same line after the instruction

In the text below, all comments in the R-code are highlighted in yellow and all instructions are in blue. Normal text is just giving further explanation.

## Instruction File and Working Directory

You can type instructions in R but usually you want a whole bunch of instructions in a sequel (i.e. code, as in a program) so you want to save that in a file, You also may want to read data or write files, so you need to work in a certain directory, Once you start a New File (and save it under a name), then start with giving a name (title) to your R-code (LT window). Since this is a comment, and not an executable instruction, you put a hash before it, e.g.

```
# This is my program for Practical GENE422
```

```
# first go to the right directory
setwd("C:/Users/jsmith33/Documents/UNITS /GENE422/Tasks")
```

Note that you can copy an address from windows File Explorer, except all slashes will have to be forward (they are backward slashes in windows File Explorer)

```
# get (or check) the address of the working directory
getwd()
```

## Reading and writing data

```
#read data using read.table and give the data set a name, usually this is called a data frame in R
```

```
mydata=read.table("Task1Data.txt",header=T,sep=" ")
```

read.table

*some options are:*

```
    header=TRUE (or just T)   #ensure it reads first lines as header
    sep=","                   #columns are separated by a comma (as in CSV files)
    sep="\t"                  #columns are separated by a tab
    sep=" "                   #columns are separated by a space
```

- To make sure you have used the correct separation symbol, check the data, e.g. by using dim(mydata), and you should have the right number of rows and columns (100 and 4 here)

## Checking and exploring the data

```
#explore the data
dim (mydata)                  #gives number of rows and columns of data set
nrow(mydata)                  #gives number of rows of data set
ncol(mydata)                  # gives number of columns of data set
mydata                       #shows everything (so use only for small datasets)
head(mydata)                  #shows the first 6 rows (plus heading)
head(mydata,8)                #shows the first 8 rows
tail(mydata,12)               #shows the last 12 rows

mydata[1:4,]                 #shows the first four rows
mydata[,1]                   #shows the first column
mydata[1:2,2:3]              #shows columns 2 to 3 from rows 1 to 2
```

```
For example

> mydata[1:6,]
    breed feed age   weight
1    angus high  19 549.2625
2    wagyu high  17 468.1380
3    wagyu  low  25 603.8947
4 hereford  low  18 424.4124
5 hereford  low  18 442.2171
6 hereford  low  18 459.9860
```

```
> names(mydata)
[1] "breed"  "feed"   "age"    "weight"
```

You can refer to each variable by combining data -and column name.
For example, if the first column is called “breed” then you can refer to it as mydata$breed
mydata$breed[1:5] would give the levels of the first five rows in the column “breed”, i.e. the breeds of the animals in the first five records.

You will note here that some variables are read as continuous (age, weight) and some as factors or characters (breed, feed). We can confirm the class of each variable by using the class function

```
class(mydata$breed)
[1] "character"
class(mydata$age)
[1] "integer”
```

The lapply function apply the class function on each column of at the data frame.

```
lapply(mydata,class)       #getting the class of all columns in one line
```

A factor is a variable with a finite number of classes. In a statistical analysis these are usually fitted as class variables  (or indeed as factors, taking up k-1 degrees of freedom if there are k levels for the effect) whereas continuous variables are fitted as covariates, each taking up only one degree of freedom.

Sometimes you might want to change a variable class. For example, if you have four feeding treatments coded as 1,2,3 and 4, the R program R would read it as an integer, and if you fit it in a model it will see it as a covariate. If you want to consider it as a factor you can do

mydata$feed = as.factor(mydata$feed)

Also names could be read as strings, whereas  to summarize or plot data, you want these variables as factors, see next.

## Understanding the data

# get some summary statistics from the data:

```
summary(mydata)                   #gives a summary of the data
```

If the variables ‘feed’ and ‘breed’ as still characters, this won’t work well. So define them as factors

```
mydata$breed=as.factor(mydata$breed)
mydata$feed=as.factor(mydata$feed)
summary(mydata)                   #gives a summary of the data
```

```
> summary(mydata)
      breed       feed          age             weight     
 angus   :28   low :52   Min.   :16.00   Min.   :366.1  
 hereford:42   high:48   1st Qu.:18.00   1st Qu.:471.7  
 wagyu   :30             Median :19.00   Median :522.1  
                         Mean   :19.38   Mean   :526.2  
                         3rd Qu.:20.25   3rd Qu.:575.2  
                         Max.   :25.00   Max.   :678.4  
```

```
levels(mydata$breed)              #e.g. useful if you expect very many levels
table(mydata$breed)                #giving nr of records for each level

tapply(mydata$weight, mydata$breed, mean)          #giving mean for each level
tapply(mydata$weight, mydata$breed, sd)            #giving SD for each level
tapply(mydata$weight, mydata$breed, summary)       #giving everything
```

```
> tapply(mydata$weight, mydata$breed, summary)    #giving everything
$angus
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  436.0   487.7   532.5   538.3   588.5   678.4 

$hereford
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  366.1   474.9   526.9   525.0   570.6   677.7 

$wagyu
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  372.3   469.6   509.0   516.6   560.4   677.7 
```

## Visualizing the data

A powerful way to explore data is to visualize aspects of the data.
You can look at the distribution (only for numeric variables) via histograms or density plots, and the association between variables via plots and boxplots.
R gives nice graphs in the RB window. For example

```
hist(mydata$weight)
plot(density(mydata$weight))
```

```
# or plot both in the same graph and a bit more sophisticated with colour and labels
hist(mydata$weight,prob=TRUE,breaks=40,col="blue",main="Weight histogram", xlab="weight (kg)", ylab="frequency")
lines(density(mydata$weight,na.rm=T),col="red"
```

For colours, you can use the actual colour (col="blue"), or you can use a number (col=4), for colour codes type "palette() "

You can look at associations between numerical variables by the plot function

```
plot(mydata$age,mydata$weight)
plot(mydata$age,mydata$weight,col=mydata$breed)  #colour dots according to breed code
```

To explore factors you can use

```
barplot(table(mydata$breed),col=c(1,2,3))  # to look at the number of observations per breed
boxplot(mydata$weight~mydata$breed,na.rm=T,col=c(1,2,3),main="Weight boxplot") #to look at distribution of weight for each breed
```

Those nice pictures can be saved as a file, so you can use them in reports

```
boxplot(mydata$weight~mydata$breed,na.rm=T,col=c(1,2,3),main="Weight boxplot")
dev.print(file="WeightdatabybreedBoxplot.pdf",device=pdf,width=8,height=8)
```

or

```
pdf(file="WeightdatabybreedBoxplot.pdf", ,width=8,height=8)
boxplot(mydata$weight~mydata$breed,na.rm=T,col=c(1,2,3),main="Weight boxplot")
dev.off()
```

Often it is useful to have several graphs in that window. You can arrange that by the command par.

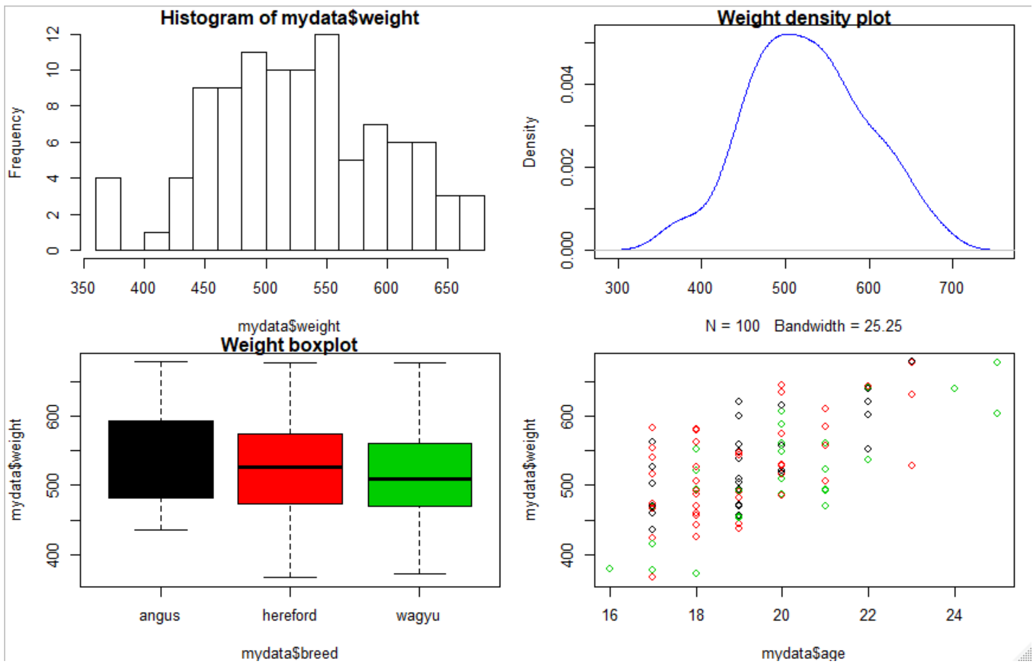
If you first type

```
par(mfrow = c(2,1), mar = c(4,4,1,1))  #two graphs horizontally aligned
par(mfrow = c(1,2), mar = c(4,4,1,1))  #two graphs vertically aligned
par(mfrow = c(2,2), mar = c(4,4,1,1))  #four graphs aligned in the window (2x2).
```

You can play around with the margins as well.

For example:

```
par(mfrow = c(2,2), mar = c(4,4,1,1))
hist(mydata$weight,breaks=20)
plot(density(mydata$weight,na.rm=T),col="blue",main="Weight density plot")
boxplot(mydata$weight~mydata$breed,na.rm=T,col=c(1,2,3),main="Weight boxplot")
plot(mydata$age,mydata$weight,col=as.numeric(as.factor(mydata$breed)))
```



## Data simulation

R is also a good platform to simulate data. Data simulation is a good tool for research, and it is also useful for exploring in small examples how things work.
We can simulate data on breeding values and phenotypes, similar (but with larger numbers) as in Chapter 2.

See here an example:

```
# simulate simple phenotypes

varP=1                        # Phenotypic variance
h2=0.1                        # heritability
N=100                         # number of individuals simulated

varA=varP*h2                  # additive genetic variance
varE=varP-varA                # residual variance
sdA = sqrt(varA)              # additive genetic standard deviation
sdE = sqrt(varE)              # residual standard deviation

gasdev = matrix(rnorm(N), nrow=N)     #Gaussian (= normally distributed) random numbers (set 1)
BV=gasdev * sdA                        # make breeding values
gasdev = matrix(rnorm(N), nrow=N)      #Gaussian (= normally distributed) random numbers (set 2)
E=gasdev *SDE                          # make residuals
Pheno=BV+E                             # make phenotypes
r=cor(Pheno,BV)                        # correlation
plot(BV,Pheno)

EBV = h2*P
r=cor(EBV,BV)
plot(EBV,BV)
```

## Task:

Run this simulation and write the correlation between phenotype and BV, as well as between EBV and BV for heritability values of 0.1, 0.3, 0.5 and 0.7

**Matrix calculations using Excel**

You can do some basic matrix calculations with MS Excel.

First put in the values of your matrices

To multiply two matrices:
- select an area of the size of the resulting matrix
- type: =**MMULT(**
- select the area of the first matrix
- type a comma (,)
- select area of the second matrix
- type a close bracket )
- press: Ctrl_Shift_Enter

To add or subtract a matrix (vector):
- select an area of the size of the resulting matrix
- type: = (
- select the area of the first matrix
- type a + or **-**
- select area of the second matrix
- type a close bracket )
- press: Ctrl_Shift_Enter

To invert a matrix:
- select an area of the size of the resulting matrix
- type: =**MINVERSE(**
- select the area of the first matrix
- type a close bracket )
- press: Ctrl_Shift_Enter

To transpose a matrix (vector):
- select an area of the size of the resulting matrix
- type: =**TRANSPOSE(**
- select the area of the first matrix
- type a close bracket )
- press: Ctrl_Shift_Enter