Setup your environment

Do not forget to switch to your own working directory

```
setwd("~/YourName")
```

R - Basic Assignments

Installing and loading the R/qtl package:

There are three ways to install a package into the R environment. The easiest is to download it from CRAN, the main repository for R packages. This can be done by using a single command:

```
install.packages("qtl")          # Get the package from CRAN
```

Another option is to install the package from Bioconductor. An alternative repository for packages (some) not found in CRAN (they are less strict compared to CRAN). Also Bioconductor tends to be faster when a package update is available:

```
source("http://bioconductor.org/biocLite.R")
biocLite("qtl")
```

Sometimes packages are only available as source package (e.g. a *.tar.gz*, *.zip* file, or a directory), the only way to install these packages is to use the interface Tools -> Install Packages, or use the command line. To open a command line in MS windows go to Start-> Run -> then type: *cmd<enter>*. After you opened the command line go to the folder where you downloaded the package and install the package with the following command:

```
$ R CMD INSTALL packagename
```

You can check if the package is installed using the following command:

```
rownames(installed.packages())
```

This will list all the packages installed. Make sure "qtl" is in the list and then load the package by:

```
library("qtl")          # Or use require("qtl")
```

Assignment:

1a) Install the R/qtl package
1b) load the R/qtl package

Format your own data:

Loading and saving data is very important in R. R provides two main functions for loading data: read.table() and read.csv(). Both functions provide similar options for loading data from a file. However (for reasons unknown to me) read.csv seems to be almost 1.5 times faster compared to read.table().

With this tutorial you get two files: phenotypes.txt and genotypes.txt. First switch to the folder where you stored these two files with the setwd() command. Use the dir() command to make sure the work directory is switched and the files can be found:

```
setwd("/path/to/file")
dir()
```

After this, read.csv is able to load the data files into the R environment:

```
phenotypes <- read.csv("http://www.dannyarends.nl/data/phenotypes.txt")
genotypes <- read.csv("http://www.dannyarends.nl/data/genotypes.txt")
```

When data is loaded correctly, we can show part of the matrices by selecting elements from the first 5 to 10 rows and/or columns:

```
phenotypes[1:5, 1:5]
genotypes[1:5, 1:5]
```

With our current files everything is not read in correctly. The dimensions of the two matrices after loading show only 1 column, and are not formatted correctly. This means that currently we are not using the correct parameters when loading in the data. A file not loaded / parsed properly which is one of the most common problems when loading data in the R environment.

Assignment:

2a) Study the help file of read.csv() and read.table(). Use the correct separator and make sure that row names and matrix headers are read in correctly before continuing with the next step. Use the data available at http://www.dannyarends.nl/data/

Hint: Use the parameter *skip* to skip the first comment line(s)

2b) Read in only half of the genotype data.

Save the data for R/qtl

R/qtl supports multiple input formats. Here we are going to use the CSVR file format to save our genotype and phenotype data. Because we do not have a genetic map we place all markers on a single chromosome and use R/qtl to create a 'real' genetic map later on.

The CSVR format:

Name		Chr		Loc								
pheno_1	,		,		,	ind1	,	ind2	,	...	,	ind_n
pheno_1	,		,		,	ind1	,	ind2	,	...	,	ind_n
...	,		,		,	ind1	,	ind2	,	...	,	ind_n
pheno_n	,		,		,	ind1	,	ind2	,	...	,	ind_n
geno_1	,	1	,	1.0	,	ind1	,	ind2	,	...	,	ind_n
geno_2	,	1	,	2.5	,	ind1	,	ind2	,	...	,	ind_n
...	,	...	,	...	,	ind1	,	ind2	,	...	,	ind_n
geno_n	,	X	,	80.5	,	ind1	,	ind2	,	...	,	ind_n

To make our data compatible with the format we need to:

1) Add two empty columns to the phenotype data (hint: see the cbind() function)
2) Put all the markers on a (fake) chromosome 1, and (hint: see the rep() function)
3) Add locations for the markers.

We can easily do this with a 'for' loop, but try using matrix operation and the apply function to keep code to a minimum.

Assignment:

3) Combine the two files (genotypes and phenotypes) in R/qtl CSVR format using R. Then save this resulting matrix into a file called: crossCSVR.txt. Confirm that everything went OK by loading in the data using the R/qtl function read.cross()

Hint: To create a vector with fake chromosome locations use:

```
markerlocs <- 1:nrow(genotypes)
```

Hint: To translate a matrix use the t() function

Basic effect mapping

We are going to use the genotypes and phenotypes from the previous parts. We want to know what the effect is of having a certain genotype at a marker on the expression of a trait. It is easiest to express this effect as difference between the means.

To analyze the effect of the genotypes.txt on the phenotypes.txt we can implement a single R function. It takes as arguments a single phenotype and the whole genotype matrix. The function should then iterate through the markers and per marker calculates 2 means (M1 and M2) and subtracts them from each other *(D)*. The function should remember these *D* values and return them. The functions signature should look like:

```
mapeffect <- function(phenotype, genotypes){ }
```

Assignments:

4a) Create a function to perform effect mapping

Hint: Look at the slides from Monday, there is an example that uses a t.test, which can be used as inspiration.

4b) Use trait "Hydroxypropyl" as input and plot the resulting vector using the plot() function. For a better look set the parameter *type* to line t="l".

Linear models

In the presentation we used a t.test to assign significance to the differences observed. However a t.test is not the only option, or even the best option. Linear models will allow us to easily

compensate for more effects like sex, age, weight and so forth. When we have this data available we can use it in the model to have a better fit to the data. As an example lets take a look at the basic model used when mapping blood pressure:

```
lm(Trait ~ genotype)
```

The previous model will probably not be as powerful to detect QTLs, when compared against the following model, which includes weight and sex as covariates:

```
lm(trait ~ sex + weight + genotype)
```

Assignments:

5a) Create a function that will use the lm() function to create a linear model at each marker, then use the anova() function to extract the likelihood (p.value) of the observed trait differences at that marker. The signature of the maplikelihood function should look similar to the mapeffect function created in Assignment 3.

5b) Scan all 24 traits and create a matrix with p values using the function from Assignment 5a, transform it into LOD scores by taking the -*log10*(p.value). Use the image function to create a 'heatmap' and customize colors so that it shows the important regions

5c) (Extra) Use your own mapeffect and maplikelihood function and Assign a direction to the QTL likelihood. Generate a matrix from the mapeffects similar to the LOD matrix and when mean1 < mean2, negate the LOD score in the output matrix from maplikelihood. Create another plot using the image() function and show the difference in direction using *breaks* and *colors* (e.g. red, orange, white, blue, green).

Hint: Use the *sign* function, and matrix multiplication

R/qtl - Basic Assignments

Loading your own data can be done by using the read.cross() functionality provided by R/qtl. Note that the format in which the data is saved is called CSVR.

```
dataset <- read.cross(format="csvr", file="file.csvr", na.str="-", geno=c("1","2"))
dataset #Print the data set summary
```

Note: we need to explicitly mention the genotype coding used in the file by using the genotypes parameter.

R/qtl comes with a range of plotting options. The most used are: plot.map(), geno.image(), plot.rf() and plot.scanone(). The plot.map() shows a graphical representation of the location of the markers on the chromosomes. The function allows us to input multiple genetic maps to compare against each other if there are markers shared between maps. We can use the est.map() function to re-estimate the genetic map of using different distance measurements.

```
data(hyper)         # Load an example data set about hypertension
plot.map(hyper)     # Genetic map of mouse used in the hyper dataset
```

The geno.image() function shows the amount of missing data and gives an overview of the distribution of genotypes. The plot.rf() function looks at the segregation per marker or across chromosomes. Additionally if we want to get a matrix of genotype values we can pull them out of the cross object using the pull.geno() function.

```
geno.image(hyper)
genotypematrix <- pull.geno(hyper)
```

We can fill missing genotype values by using (1) multiple imputation or (2) expectation maximization. This is provided by the fill.geno() function in R/qtl. The fill.geno() function fills missing genotype values with the most likely values based on the surrounding marker data.

```
hyper_f <- fill.geno(hyper)
geno.image(hyper_f)
```

Using plot.rf() we can plot recombination frequencies to detect any segregation distortion. Furthermore looking at the recombination frequencies gives us an initial idea of the resolution we might obtain in mapping.

```
plot.rf(hyper)                          #Show the recombination break points
plot(pull.rf(hyper,"lod"),"D2Mit62")  #Show the resolution on marker D2Mit62
```

Assignments:

6a) Load the csvr data set from assignment 3 and create the different plots explained above (save them to file by using the png() function). We created a cross object without information about the genetic map. *How many chromosomes are there by looking at the RF and genotype plots?*

6b) The function formLinkageGroups() uses pairwise linkage information between markers as calculated by est.rf() to partition markers into linkage groups. Parameters need to be adjusted to obtain a suitable genetic map. Use the formLinkageGroups() function to assign chromosomes to the csvr data loaded in assignment 6.

Hint: The function provides the parameter *reorgMarkers*. When this parameter is set to TRUE a new cross object is returned

6c) Use the functions est.map() and replace.map() to move markers on the chromosomes to their position according to the recombination frequencies observed between them. Use the plot.map function to compare the different strategies of creating a recombination map (Haldane, Kosambi, Morgan). *Which chromosome is the longest, and how many cM is it?*

6d) (Extra) Ordering of markers inside a chromosome can be done by the orderMarkers() function. We can use it to confirm that the current ordering between markers is optimal (Observe that when we execute the function on our current cross no marker is reordered. This means that the current ordering is optimal). Ordering markers is done by swapping markers and confirming that the new ordering will lead to a larger chromosome length then the original ordering. Internally R/qtl uses the ripple() function for this purpose, it provides two different methods: countxo (fast) and likelihood (slow). *Use it on a small chromosome and study the output of the ripple function.*

Scanning for QTL

R/qtl provides many ways of scanning for QTL. The user can choose to use an EM algorithm, imputation, Haley-Knott regression, the extended Haley-Knott method, or marker regression. All of these methods can be applied to the cross object using the scanone() function. An example:

```
result_hk  <- scanone(hyper, pheno.col=1, method="hk")
result_em  <- scanone(hyper, pheno.col=1, method="em")
result_imp <- scanone(hyper, pheno.col=1, method="imp")

plot(result_hk, result_em, result_imp, lwd=c(4,3,1),col=c("red","green","blue"))
legend("topleft",c("hk","em","imp"),lwd=c(4,3,1),col=c("red","green","blue"))
```

To determine when a LOD scores can be called significant, we use the *n.perm* parameter of the scanone() function:

```
perm_hk <- scanone(hyper, pheno.col=1, method="hk",n.perm=1000)
plot(perm_hk)
summary(perm_hk)  # Print the thresholds
plot(result_hk)
abline(h=summary(perm_hk)[[1]],col="green",lty=2,lwd=2)
abline(h=summary(perm_hk)[[2]],col="orange",lty=2,lwd=2)
```

Assignments:

7a) Scan all 24 traits in the cross object from assignment 6b by using the scanone routine. To avoid excessive warnings first use the calc.genoprob() function. This function calculates the genotype probabilities which are used when mapping QTL.

7b) Create a heatmap using the image() function to show QTLs from all 24 traits. Compare the heatmap to the results obtained earlier (Assignment 5b). *What is the difference between using Haley-Knott regression compared to the anova on the linear model we performed earlier?*

7c) Use permutation to obtain threshold for the hyper dataset and the cross created in 6b, *is there a difference? Is this expected?*

7d) We can use the lodint() function to obtain LOD confidence intervals for the QTLs we observe. This function takes only a single scanone object (so where pheno.col was of length 1). Rescan all traits and use lodint() to create a matrix of ONLY significant QTL (use the threshold from assignment 7c) and their associated marker range and write this to a file called allSignQTL.txt. The header of the matrix should be:

```
colnames(allSignQTL) <- c("phenotype", "chr", "left", "top", "flank", "lod")
```

How many significant QTLs are there in total?

Multiple QTL Mapping

Multiple QTL mapping allows compensating for known genetic effects. Furthermore it provides an unbiased backward selection strategy to select markers most associated with a trait. In most cases MQM cannot handle all markers as cofactors (An overfit of the model because the number of individuals <= cofactors). So as a rule of thumb we can place (# individuals - 20) cofactors, which will be used in backward elimination. Also MQM needs complete genotype information, which one can create using the fill.geno() function of the mqmaugment() function. MQM augmentation makes a dataset unsuitable for usage with scanone, because individuals are augmented (duplicated) inside the cross object, here an example using the ***multitrait*** dataset.

```
data(multitrait) # We use another example dataset
multi_filled <- fill.geno(multitrait) # Use imputation, fill the missing genotypes
cofactors <- mqmautocofactors(multi_filled, 40, plot=T)
result_mqm_imp <- mqmscan(multi_filled, cofactors=cofactors, verbose=T, plot=T)

multi_aug <- mqmaugment(multitrait) #Use augmentation to fill the missing genotypes
geno.image(multi_aug)
nind(multi_aug)   #The augmented set contains more individuals then
nind(multitrait)  #The unaugmented dataset
result_mqm_aug <- mqmscan(multi_aug, cofactors=cofactors, verbose=T, plot=T)
```

Assignment:

8a) Use MQM (no cofactors) to map one of the traits from the Assigment 6b cross object and compare the results against scanone. *What do you observe?*

8b) Place 20 to 30 cofactors on the genome and use MQM, *what is the difference? Is this expected?*

8c) MQM allows scanning for additive and dominant effects; this only comes into play with an F2 cross. Load the ***listeria*** dataset and use the fill.geno() function to fill in missing data, then scan for QTL using both the additive model and the dominance model (no need to use cofactors), plot both results. *Are both scans equal? Why do they differ? What does this mean for more elaborate cross types (e.g. 4-way, Magic, Collaborative cross)?*

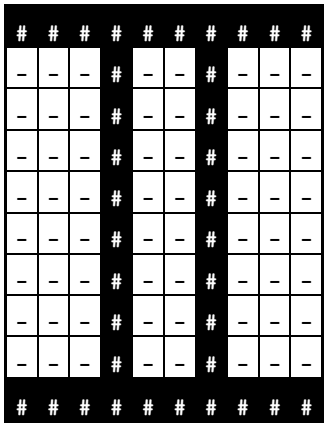
R - Additional assignments

Assignment LED Display

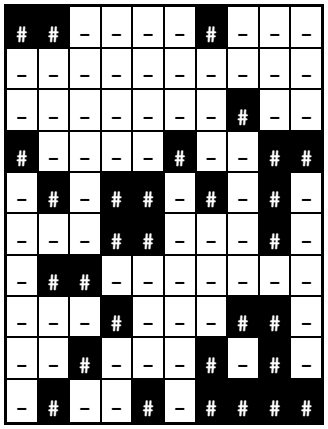
A1) A LED display has 10 rows and 10 columns. Try to create a function with a single parameter. This parameter is used to specify which LEDs should be on.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

```
on <- c(1:10, seq(4, 100,10),seq(7, 100,10), 91:100)
led(on)
```



```
on <- c(sample(100, 30))     # Random pattern
led(on)
```



R/qtl - Additional assignments

Assignment Fix a genetic map

A2) A genetic map needs to be suitable before using it for Multiple QTL Mapping. Duplicate markers at different positions or different markers at the same position will need to be removed prior to the analysis. If they are not removed this will result in the recombination frequencies between those markers being infinite or 0 leading to an over fit in the modeling / mapping procedure of MQM. I have prepared a csvr dataset cross object with a genetic map unsuitable for MQM mapping.

The assignment is to drop the correct (minimal amount of) markers from the cross object to make sure the mqmscan() function is able to scan all the chromosomes. *Which markers need to be removed?*