Quantitative methods in plant breeding: the digital tutorials.

NIAB 2021

2021-02-22

This is a PDF copy of the QMPB digital tutorials from the 2021 course. Please do not share outside of the course.

Contents

Chapter 1

Introduction

Welcome to the online tutorial booklet written for the QMPB course at NIAB. The book has been written using *markdown* and formed using a R package called **Bookdown**. You will be prompted to a chapter/section in this book for each tutorial.

This is our first year using digital tutorials, while we happy with the end product, if you spot or experience any *teething issues* please let us know and we'll work to resolve the problem.

For each tutorial, you will typically need the corresponding dataset which will enable

you to work alongside the online tutorials on your own personal computers. To download the datasets, please follow this link: https://tallywright.github.io/jekyll/update/2021/01/05/QMPB-data-repository.html

Via the link, download the .zip folder containing the data files. The folder should be organised within chapters and each tutorial will prompt you towards which dataset to use and also how to use it.

Please don't be afraid to ask for help where needed throughout these tutorials. We're here to help!

Chapter 2

Acknowledgements and Contents

2.1 Acknowledgements

This is the 13th time NIAB has run its Quantitative Methods in Plant Breeding course. The course was devised and created at NIAB by Professor Ian Mackay (now of Implant Consultancy Ltd. and SRUC, i.j.mackay@gmail.com[1]) from discussions with Mike Kearsey, Noel Ellis, Wayne Powell and Andy Greenland about the need for training for plant breeders and what and how NIAB could contribute. Ian also wrote this course manual.

This is the 7th time in which I have taught on the NIAB QMPB course. Like so many participants over the years, Ian has stimulated my thinking on all aspects of plant breeding and I have learnt a great deal from him. I hope I can continue to add to the wealth of knowledge and practical learning that he has developed in the course.

Over the years, the course has also benefitted from the input of several guest lecturers, as well as assistance in lecturing and demonstrating from colleagues in the John Bingham Laboratory at NIAB. In particular, we would like to recognise the enthusiastic contributions of our former NIAB colleague, Greg Mellers, who sadly passed away in late 2019.

[1] mailto:i.j.mackay@gmail.com

All the course administration has been handled by NIAB's Mary MacPhee. I would also like to thank Tally Wright, Nick Fradgley, Camila Zanella, Joachim Nwezeobi and Yeorgia Argirou for their roles in preparing and teaching the course and Joseph Amosu for managing the IT support.

We would like to thank the authors of the `bookdown` package (Xie, 2020) which has been used to form this online book, which was built with R Markdown and `knitr` (Xie, 2015).

Please feel free to contact me with any questions: `keith.gardner@niab.com`[2]. For more details about Genetics and Breeding at NIAB please see: `https://www.niab.com/research/agricultural-crop-research-0`

Keith Gardner

NIAB

2.2 Digital Tutorial Chapters:

- Chapter 1: Introduction
- Chapter 2: Acknowledgements and Contents
- Chapter 3: R and Rstudio: a general introduction with statistics.
- Chapter 4: Trial design

[2] `mailto:keith.gardner@niab.com`

- Chapter 5: Trial analysis
- Chapter 6: Cross site analysis
- Chapter 7: Population genetics
- Chapter 8: Imputation
- Chapter 9: Association Mapping.

Chapter 3

R and Rstudio: a general introduction with statistics.

3.1 Introduction section on R

R is free software. It can be used as an overpowered calculator, to carry out many standard statistical analyses, and to develop applications carrying out complex analyses in specific subject areas. Many of these applications and much other useful information are freely available from the Comprehensive R Archive network (CRAN) website[1]. In particular, the guide "An Introduction to R" is available under the documentation section and can be used to supplement the outline given here. Another excellent source is "Introductory Statistics with R" by Peter Dalgaard.

R is just one among several excellent statistical packages. Its major strength is that it is free and this has led to an ever expanding user base and to the development of more and better applications. R is commonly used in many commercial and academic establishments: knowledge of R is a useful and transferable skill to acquire.

For many users, the major disadvantage of R is that commands need to be typed at a prompt, rather than selected from a menu. However, most commands are short, and as we shall see, much repetitive typing can be avoided. Over the last few years, RStudio, an interface to R, has become so popular that it is now the standard way by which users are introduced to the package. RStudio retains the look and feel of R: it still requires commands to be typed but is more accessible. In this tutorial, we shall use Rstudio, but if you can use R you can use Rstudio, and vice-versa.

The intention of this guide is twofold. Firstly, it is to help the beginner to get started. It will show you how to get data into RStudio, carry out basic analyses, produce graphs,

[1] http://cran.r-project.org

and save results. It is hoped that this will allow users to dip directly into more complete guides as required. This guide is a long way off being thorough and many of the definitions and descriptions provided here are not wholly accurate. The hope is that the guide will allow you to carry out useful analyses as quickly as possible and provide a base from which additional knowledge can be acquired without too much extra effort.

Secondly, the guide is intended to act as revision of some basic statistical methods. For this purpose we shall be studying some properties of a panel of European winter wheat lines; the TriteceaeGenome panel which was collected as part of a European collaborative project in association mapping, the results of which have been described by (Bentley et al., 2014). A copy of the paper has been supplied in the references folder on the course website[2]. We shall use this dataset for various analyses throughout the course.

The subset of the TG panel data which we shall use today are in "TG data for day 1.xlsx". Open the file and have a look and I shall describe the data. We can discuss what sorts of hypotheses we would be interested in testing with these data.

3.1.1 Getting the data ready

The first step of this tutorial will be to open this excel sheet (in excel) and save a copy of it as a comma-separated values (CSV) file. This can be completed in excel via the following steps:

1. Opening the file called: "TG data for day 1.xlsx"
2. Going to 'save as' and then changing the file type to: CSV (Comma delimited) (.csv).
3. Save this file in the same folder as the ".xlsx" file. You can keep the same name, as you are creating a different file type. You can only save a single excel sheet as a CSV file and that file won't contain any formulas or formatting, only values. Now that file is converted to a CSV file, we can move on to using RStudio and come back to it later.

You will need to complete the above section before moving on to the rest of the tutorial.

3.2 The Rstudio layout, workspace and enviroment.

Next open Rstudio which should be installed on your laptop. Typically Rstudio will open with three windows (possibly four if your script from your last session opens automatically).

[2] https://tallywright.github.io/

If you only have three windows, open a new script by pressing: *Ctl + Shift + N*

You should now see four screens:

1. A console or command window. This is on the bottom left in the screen-shot. In this window we type commands and the result of those commands is listed. If you were to work in R, rather than R studio, this is the only window you would see.

2. A script editing window: for writing scripts. This is shown in the top left. Commands can also be run from here.

3. An environment / workspace and history window (top right). Within the environment window you will see user defined objects (data frames, lists, vectors ect.) This workspace can be saved for later use.

4. A window for files, plots, packages and help (bottom right).

You can re-size these windows, move about between them and essentially do everything that is possible in an ordinary R session, including developing your own programs and scripts. The idea is that this is easier and more convenient than working in R directly.

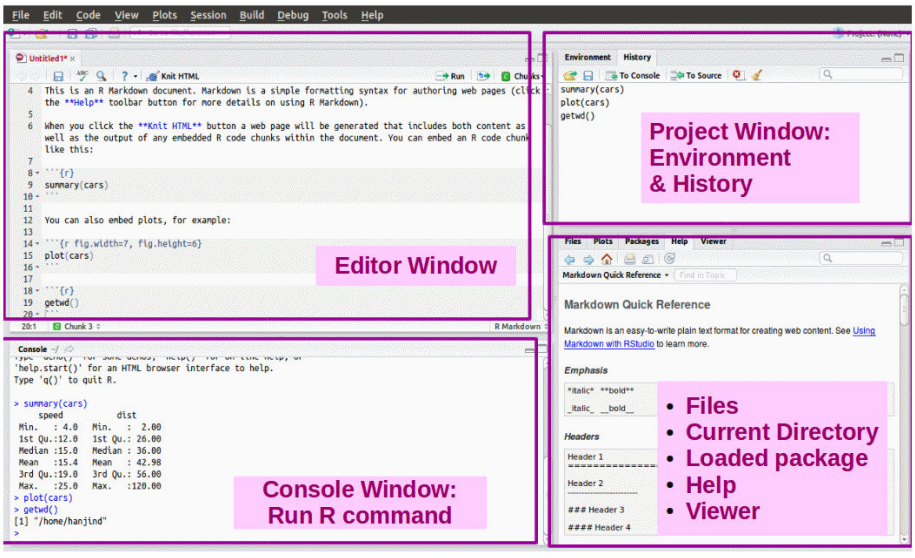


Figure 3.1: The structure if RStudio

3.2.1 Working directory

The current working directory can be shown by the `getwd()` function. This is ultimately the folder on your computer that your current R session is working from. You can type this in the script window and then press **Ctrl + Return** to run, the cursor needs to be on the line you want to run. In the working directory, files will be loaded when called automatically and files will be automatically saved to this folder. You can change your working directory manually using the function `setwd()`. Use the folder path to prompt where the working directory is set. However, sometimes it's easiest just to click on the tab at the top of the window and use the route of the drop-down menu: *Session / Set Working Directory / Choose Directory*. Which works just as well.

1: You want to be working from a folder where the data you will be using is saved in. You could create a new folder for this tutorial and then add the CSV file "TG data for day 1.csv" that you created earlier. You can do this outside of RStudio.

Next, with RStudio, change the working directory to the folder containing the CSV file "TG data for day 1.csv" that you created earlier. Then you can run the line `list.files()` in the script window to ensure that the file is present in your current working directory.

3.3 Basic R Syntx

Write your commands into the *script editing window*, to then run a command you can leave the cursor on the line you wish to run and press *Ctrl + Enter*. You can also press *Run* at the top right of the window. If you want to run multiple lines at once, highlight the desired lines and click *Run* or press *Ctrl + Enter*.

When you run the line in the *script editing window* you will see your command appear in the console and the result of what you have run.

For example entering 9+1 will return:

```
9+1
```

```
## [1] 10
```

You are now fully trained to use R as a calculator! The syntax is essentially the same as in Excel. Thus `^2` will square numbers, `sqrt()` will take the square root and so on. For example:

```
sqrt(2)
```

```
## [1] 1.414214
```

Returns the square root of 2. The `[1]` is just a line number.

Note that R is case sensitive, running `LOG10(2)`. Will produce an error.

Whereas:

```
log10((9+1)^2)
```

```
## [1] 2
```

Does not produce any errors.

If you make a mistake, you can easily edit the line that contains the error in the *script editing window*. Some of the new versions of Rstudio will flag up mistakes in a line in the *script editing window*.

Alternatively if you are working directly in the *console or command window* and you make a mistake, it can be corrected by tapping the up arrow on the keyboard to bring back the last typed command, then moving along the command line using the left and right arrows, deleting characters using the backspace or delete key, inserting correct numbers or text and hitting the carriage return key. Multiple hits of the up arrow will bring back successively earlier commands. However, an easier way of finding and re-entering commands used previously is to click on the "history" tab at the top right. This shows the previous commands you have run. If you select one and press *Ret*, then that command will be passed to the console window. Press *Ret* again and the command will be executed. It is even possible to select several non-contiguous commands in the history window, pass them to the console and execute them all as group.

To avoid this slightly laborious approach, just write and run your scripts through the *script editing window*. This way you will have a complete record of your code preserved.

Note also that it is possible to cut from windows outside RStudio and paste into it. So R commands listed in another document, for example this one, can be copied and pasted into the *console or command window* or the *script editing window* and will run (after hitting carriage return). You will find this useful during the course.

3.3.0.1 Brackets

We have seen brackets used above in simple formulae - just as in Excel formula. However, brackets are used more extensively in R: all commands include brackets. For example:

Running `quit()` will end your R session. However:

```
quit
```

```
## function (save = "default", status = 0, runLast = TRUE)
## .Internal(quit(save, status, runLast))
## <bytecode: 0x000000001648c8c8>
## <environment: namespace:base>
```

Outputs the above. Note how the use of the brackets changes the outcome. The above is the section of computer code which is run when you type `quit()`. Unless you are an enthusiast, it is not necessary to know what this means.

3.3.0.2 Equals symbols

R has four sets of symbols all of which loosely mean "equals" but which have differences in use. This can take some getting used to.

The most commonly used set is

`<-`

or sometimes

`->`

These symbols are typed using the dash (the minus sign) and the left or right arrow. They are used to assign results from one side of the arrow to the other. For example:

```
my_first_result <- sqrt(log10((9+1)^2))
```

Won't return a result and just returns the prompt in the command window. But then typing:

```
my_first_result
```

```
## [1] 1.414214
```

Returns the desired result. We have created a variable my_first_result and assigned the result of our calculation to it, using <-. The same result would be achieved by:

```
sqrt(log10((9+1)^2))-> my_first_result
```

The entry stored in `my_first_result` is now available for additional manipulation. For example:

```
my_first_result^2
```

```
## [1] 2
```

There are historical reasons why <- was introduced for this purpose, but R also allows the use of "=" in addition to <- in assigning values to variables. `x<-3` and `x=3` both return the value of 3 if you run `x`. Note however that `2=x` will return an error. Because of its lack of ambiguity with the other use of = (see below), I prefer to stick to <- and -> in assigning values, though this can seem artificial and unnecessarily geeky at first.

The second equals symbol is the tilde ~ (above the # on my keyboard). This is used in R in statistical analysis to distinguish between x and y variables; that is to say between what is being analysed (e.g. the phenotype) and what factors and variates it is being analysed with (eg marker data and environmental factors or variates). So for example:

```
P~G+E
```

is R syntax to state that a variate P is to be explained by two variates or factors G and E. If you try this in R now nothing will happen. The exact context in which this syntax is used will be given later. For now, note that

```
P~G
```

would represents linear regression of P on G Interactions can also be included:

```
P~G+E+G:E
```

includes an interaction term between G and E (represented by G:E). This can be abbreviated to:

```
P~G*E
```

Yet more complex models, for example with nested factors, can be described by including bracketed terms.

The third equals symbol is =. This is most often used within commands to provide information about specific parameters. For example we shall come across:

```
xlab= "what you put here is used to label the x axis in a graph"
```

used in commands to generate graphs.

The final equals description is ==. This truly means "is equal to", and is used to test relationships: is A equal to B is written as

```
A==B
```

Try this sequence of commands

```
A<-2

B<-3

A==B
```

Related to == are:

`!=` not equal to

`>` greater than

`<` less than

`>=` greater than or equal to

`<=` less than or equal to.

These, together with `==` itself, are entered here for completeness. Try `A!=B`.

3.3.0.3 Continuation character.

If an R command is incomplete when the carriage return is depressed, you are prompted with a + to continue the command on the next line. Typing `sqrt(log10` returns: `+`

The rest of the command can be entered after the `+`:

```
sqrt(log10

+ ((9+1)^2))

[1] 1.414214
```

You might find it hard to generate these errors in Rstudio, as it helpfully inserts a `)` as soon as you type a `(`.

3.3.0.4 Text and numbers

Text is distinguished from numbers and from R commands by the use of quotes. Either single quotes 'text' or double quotes "text" will do, but the quotes must match. Again, RStudio help you with this.

3.3.0.5 Summary of syntax

Arithmetic: `+ - / * ^ ( )` just like Excel

Commands use brackets: `quit()` is correct. `quit` is wrong.

Equals:

- `<-` stores a result
- `~` is described by
- `=` options in commands
- `==` logical equivalence

For text use quotes: `"hello"` or `'hello'`.

This introduction should provide sufficient syntax to get you going. Additional syntax is introduced, as required, in the discussion of specific commands and operations in the remainder of this document.

2 Assign the result of `(9.75 * 80) - 34` to an object called: `my_second_result`. Then print out `my_second_result`. Lastly divide `my_second_result` by 2.

Assign the word `harvester` to an object called: `Combine`. Then run `Combine`.

3.4 Reading in data

There are many ways of entering the data you wish to analyse. For example, R can read data from the clipboard, so you can copy data directly from Excel. It will read tab separated files (files in which different columns of data are separated by tabs, and it will read comma separated files (csv files). There are more advanced methods to interface directly with databases, to capture data from the web, and so on. There are also several different formats for reading data in.

Earlier we changed our working directory to the location of our TriticeaeGenome data files, which we saved as a CSV file. If you missed this, do it now:

From the top of the Rstudio window select:

Session / Set Working Directory / Choose Directory

Then browse to locate your directory (choose the folder where you saved the data as a CSV file). When we exit RStudio, your work will be saved here (as long as you remember to actually save it), making it easier to return later and pick up where you left of. We shall explain this later.

If you follow the above process, in the Console window, you will see echoed the command that you could have typed to change directories. For me, this was:

```
setwd("C:/Users/x991625/OneDrive - NIAB/a). NIAB WORK/QMPB 2021/R_bookdown_local/bookdown-demo-master")
```

If you copy this line to the top of your *script editing window* and if you had to re-run the script this line would set the working directory for you. It can also serve as a reminder on where your files are saved. There may be times when you wish to change directories as part of a larger sequence of commands that you want to run, in which case you would use the typed form. Note - A sequence of commands is commonly called a "script": in essence it is a small computer program.

We may not cover Rprojects over these days but an alternative available within RStudio, but not R, is to create a project. From the menu at the top, select:

File / New Project / Existing Directory /

Then browse to the directory you want to work in and *Create project*

Later, when you quit RStudio, you will find a file with the extension .Rproj has been created in the selected directory. If you click on that, then RStudio will open with all your previous work saved.

Now we have our working directory set we can read in the data. Personally I always think it's safest to read in the data as a comma-separated values (CSV) file. Earlier I asked you to save the data as a CSV file and that file should be in our working directory. We can check what files are in our working directory by running:

```
list.files()
```

```
##  [1] "_book"                          "_bookdown.yml"
##  [3] "_bookdown_files"                "_build.sh"
##  [5] "_deploy.sh"                     "_output.yml"
##  [7] "_render.R"                      "02-Acknow.Rmd"
##  [9] "03-R_and_stats.Rmd"             "04-Trial_design.Rmd"
## [11] "05-Trial_analysis.Rmd"          "06-Cross_site_analysis.Rmd"
## [13] "07-Population_genetics.Rmd"     "08-Imputation.Rmd"
## [15] "09-Assocation_mapping.Rmd"      "10-references.Rmd"
## [17] "aug_blocks.csv"                 "book.bib"
## [19] "bookdown-demo.aux"              "bookdown-demo.bbl"
## [21] "bookdown-demo.blg"              "bookdown-demo.idx"
## [23] "bookdown-demo.ilg"              "bookdown-demo.ind"
## [25] "bookdown-demo.out"              "bookdown-demo.Rmd"
## [27] "bookdown-demo.Rproj"            "bookdown-demo.toc"
## [29] "bookdown-demo_files"            "cover.jpg"
## [31] "data"                           "delete.csv"
```

```
## [33] "DESCRIPTION"                    "Dockerfile"
## [35] "images"                         "index.Rmd"
## [37] "journals.bib"                   "latex"
## [39] "lattice_25.csv"                 "lattice_9.csv"
## [41] "LICENSE"                        "now.json"
## [43] "packages.bib"                   "preamble.tex"
## [45] "README.md"                      "style.css"
## [47] "tables"                         "TG data for day 1.csv"
## [49] "this will be the file name.csv" "toc.css"
```

In my example there are a number of files in my working directory. That's because my working directory contains all the files used to build this book. We are interested in reading in the file: `TG data for day 1.csv`. To read the data in we would use a command like:

```
TG_data_for_day_1 <- read.csv("TG data for day 1.csv",header=T)
```

Looks complicated? It's not really. We are only using the function `read.csv` which takes the the name of the csv file you want to read in as the first argument. Note this name has to be in speech marks and requires the .csv at the end. The second argument is `header=T`, this ensures that the top row in the excel file is read in as the column headers in R. The other part of the command was to assign the data to an R object called `TG_data_for_day_1`.

A key point is to consider how any program you are using handles `NA` (missing data). For R to treat excel cells with no data as `NA` those cells should contain `NA` and not be left blank. This was done for you with this data. However, that won't always be the case.

It's always good to make sure that the data was read in correctly and there are no obvious issues. This can be done with some simple functions. I always like to make sure the headers (column headers were read in correctly):

```
names(TG_data_for_day_1)
```

```
##  [1] "variety"     "year"        "ORIGIN"      "Rht2"        "PpdD1"
##  [6] "yield"       "CALLOW_2011" "FRANCE_2010" "FRANCE_2011" "LGE_2010"
## [11] "LGE_2011"    "NIAB_2011"   "FT"          "HT"          "AWNS"
```

Looks good, we have 15 distinct headers. One of the many strengths of R is the help documentation. Throughout this tutorial and your own research, if you come across a function you are unsure about, or want to know what arguments the function uses or want some examples of using the function, you can just enter the function name into **help()**. For example try running:

```
help("names")
```

This opens a help window in the Rstudio pane on your right, where you should see all the required information.

We should also check the dimensions of the data, which can often indicate if there's an issue. From the excel sheet we have 377 rows and 15 columns. We check the dimensions of our object in R like this:

```
dim(TG_data_for_day_1)
```

```
## [1] 376  15
```

Rows are shown first, followed by columns.

The rows are one less than expected. Why? Because we read in the first row as a column headers using `headers=T` in the `read.csv()` function.

Additionally, you should also see the data in the *Environment Window* in the top right pane of RStudio. You can see the number of rows (obs.) and columns (variables) there too. You can click on this (or run `View(TG_data_for_day_1)`) to view the data displayed in a spreadsheet format in the editing window. You can apply filters to the various columns to see subsets of the data if desired. Don't do this if you are working with large datasets! It will freeze RStudio or takes a very long time to load. R is not designed for viewing large data files in this way.

We now have our data ready and waiting for analysis. To display a variable (a column), say yield, we should type `TG_data_for_day_1$yield`. However, this would print out 376 numbers and to save paper, we can only request the first five values using `[1:5]` after our command, like so:

```
TG_data_for_day_1$yield [1:5]
```

```
## [1] 103.23  88.55  90.76  82.24  96.66
```

Notice the use of the `$` to access a variable from your data frame.

Writing out the name of the data object each time is tedious. To speed things up, we can `attach` our data:

```
attach(TG_data_for_day_1)
```

R will now assume that when you type yield, you mean `TG.data.for.day.1$yield`.

This can sometimes be dangerous: if you have another dataset you are working with at the same time which also has a variable called yield, then it can be confusing to know which you are working with.

To stop working, by default, with the TG_data_for_day_1 dataset you could type: `detach(TG_data_for_day_1)`. If you do this now, don't forget to attach it again!

We could had read our data in using `read.table()`, this is how we would read in data saved as a .txt file. We will use this function throughout the course. It is very similar to `read.csv`. Use the help documentation in R, to the read further information on the `read.table()` command, i.e. run: `help(read.table)`.

You can also use the `read.table()` function to read from your clipboard. This is achieved by copying cells in excel (your data table) and then running:

```
read.table(file = "clipboard", header=TRUE)
```

Note that this is not a reproducible method and can lead to some downstream issues if you forget where your data came from.

3 You can also read data from a .txt file using the function `read.table()`. The function will take the same arguments as `read.csv()`. Save the data in excel as a tab delimited file (.txt) and then read this file in using `read.table()`, save it as: my_data2.

Use the functions I showed above to ensure `read.table()` was read in properly.

3.5 Understanding data types and R objects

Now that we have imported our data into R, it becomes important to understand some of the basic data types.

```
class(TG_data_for_day_1)
```

```
## [1] "data.frame"
```

If we run the function `class()`, R will tell us what type of data object we are working on. The data.frame is a common form of data for plotting, analysis and statistics. The `read.csv` function will form a data.frame automatically. Within a data.frame the columns are vectors. A vector contains a series of numbers (or characters). A vector is the most common and simple data form of R. You can assign a vector manually, using the function `c()`. This is a generic function which combines its arguments. For example:

```
my_first_vector <- c(167,212,358,973,214,558)
my_first_vector
```

```
## [1] 167 212 358 973 214 558
```

Above we have produced a numeric vector of 6 numbers. We can confirm this by using `class()` function.

```
class(my_first_vector)
```

```
## [1] "numeric"
```

We could also make a vector containing just characters:

```
my_second_vector<- c("Tios", "Paragon", "Skyfall", "Grafton", "Graham", "Robigus" )
my_second_vector
```

```
## [1] "Tios"    "Paragon" "Skyfall" "Grafton" "Graham"  "Robigus"
```

We've produced a character vector that contains 6 names.

```
class(my_second_vector)
```

```
## [1] "character"
```

A Vector can also be logical, as in it contains either TRUE or FASLE:

```
my_third_vector <- c(FALSE, FALSE, TRUE, TRUE , TRUE , TRUE)
class(my_third_vector)
```

```
## [1] "logical"
```

We can now actually combine these vectors into a data.frame. A data.frame can handle a combination of variables which are of different modes(numeric, logical, ect.).For example if we use the `str()` function on our previously loaded data.frame, we can see the different types of each variable

```
str(TG_data_for_day_1)
```

```
## 'data.frame':    376 obs. of  15 variables:
##  $ variety    : chr  "AARDEN" "AARDVARK" "ABELE" "ABO" ...
##  $ year       : int  2003 1997 1970 1977 1998 2007 2004 2007 2006 1992 ...
##  $ ORIGIN     : chr  "DEU" "GBR" "GBR" "FRA" ...
##  $ Rht2       : int  1 NA 0 0 1 1 1 0 1 1 ...
##  $ PpdD1      : int  0 1 0 1 0 1 1 1 0 0 ...
##  $ yield      : num  103.2 88.5 90.8 82.2 96.7 ...
##  $ CALLOW_2011: num  114.2 96.7 95.3 87.1 102.9 ...
##  $ FRANCE_2010: num  136 122 124 115 132 ...
##  $ FRANCE_2011: num  100.2 96.4 92.9 85.8 101.5 ...
##  $ LGE_2010   : num  93.5 75 82.8 77.5 84.7 ...
##  $ LGE_2011   : num  96.2 75.7 83.2 68.4 87.5 ...
##  $ NIAB_2011  : num  79.4 65.3 66.1 59.5 71.3 ...
##  $ FT         : num  156 153 154 149 157 ...
##  $ HT         : num  67.6 66.3 81.1 69.6 66.7 ...
##  $ AWNS       : int  0 0 0 0 0 1 0 0 0 0 ...
```

It seems that the variables (columns in the data.frame) are either in the form of factors, numeric or integers. Depending on their content. Let's form a data.frame from our three vectors:

```
my_DF<-data.frame(my_second_vector, my_first_vector, my_third_vector)
my_DF
```

```
##   my_second_vector my_first_vector my_third_vector
## 1             Tios             167           FALSE
## 2          Paragon             212           FALSE
## 3          Skyfall             358            TRUE
```

```
## 4          Grafton            973           TRUE
## 5           Graham            214           TRUE
## 6          Robigus            558           TRUE
```

Note the length of the variables need to be the same. We can now use `class()`, `str()` and `dim()` to inspect the `data.frame`.

```
class(my_DF)
```

```
## [1] "data.frame"
```

```
str(my_DF)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ my_second_vector: chr  "Tios" "Paragon" "Skyfall" "Grafton" ...
##  $ my_first_vector : num  167 212 358 973 214 558
##  $ my_third_vector : logi  FALSE FALSE TRUE TRUE TRUE TRUE
```

```
dim(my_DF)
```

```
## [1] 6 3
```

So we have a new data.frame, with 6 rows and 3 columns. The first column contains factors (which is typically how text is handled) and the second is numeric. The third is logical. Factors are categorical data and are stored as integer vectors.

We can use a combination of `names()` and `c()` to change our column names:

```
names(my_DF)<- c("Line","Block","Winter_Habit")
my_DF
```

```
##      Line Block Winter_Habit
## 1    Tios   167        FALSE
## 2 Paragon   212        FALSE
## 3 Skyfall   358         TRUE
## 4 Grafton   973         TRUE
## 5  Graham   214         TRUE
## 6 Robigus   558         TRUE
```

We can force R to change the type of a column. For instance if we wanted "Block" to be treated as a factor rather then numeric, we could use:

```
my_DF$Block<-as.factor(my_DF$Block)
str(my_DF)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ Line        : chr  "Tios" "Paragon" "Skyfall" "Grafton" ...
##  $ Block       : Factor w/ 6 levels "167","212","214",..: 1 2 4 6 3 5
##  $ Winter_Habit: logi  FALSE FALSE TRUE TRUE TRUE TRUE
```

Note the use of the `$` sign to specify a column (variable) of a data.frame. We will use this a lot throughout. We could work in reverse now and use this to extract a column from a data.frame and save it as a vector:

```
my_V<-TG_data_for_day_1$yield
length(my_V)
```

```
## [1] 376
```

We have extracted the yield column from the data.frame we loaded earlier and saved it as a variable consisting of 376 numbers. If we run `my_V` we would see all of these numbers in the console.

To save space, let's output only the 5th number from the 376 numbers, using `[]` brackets like so:

```
my_V[5]
```

```
## [1] 96.66
```

```
my_V      [5]
```

```
## [1] 96.66
```

```
my_V            [5]
```

```
## [1] 96.66
```

Note how the variable uses of spaces makes no difference, that's a nice element of R.

Let's say we only wanted the first five elements:

```
my_V[1:5]
```

```
## [1] 103.23  88.55  90.76  82.24  96.66
```

We can reverse the output like so:

```
my_V[5:1]
```

```
## [1]  96.66  82.24  90.76  88.55 103.23
```

We can also pick and choose using `c()`:

```
my_V[c(121, 200, 345)]
```

```
## [1]  98.83 102.75 100.24
```

The use of square brackets is the same for a data.frame, except a data.frame consists of rows and columns:

```
dim(TG_data_for_day_1)
```

```
## [1] 376  15
```

We have 376 rows and 15 columns and with this any element of a data.frame can be called using: `data.frame[row, column]`.

```
TG_data_for_day_1[5,6]
```

```
## [1] 96.66
```

That's the data point in the 5th row and the 6th column. We could look at the whole of the 5th row like so:

```
TG_data_for_day_1[5,]
```

```
##   variety year ORIGIN Rht2 PpdD1 yield CALLOW_2011 FRANCE_2010 FRANCE_2011
## 5  ACCESS 1998    GBR    1     0 96.66      102.86      132.06      101.49
##   LGE_2010 LGE_2011 NIAB_2011    FT    HT AWNS
## 5    84.68     87.5     71.34 156.6 66.69    0
```

We can't save this as a vector as there are different data types present (factor and numeric), so assigning this row from our data.frame to a new object would create a new data.frame, with 1 row and 15 columns.

3.5.1 Summary

I've only focused on a subset of data types and classes. Here are some more examples and formal definitions that I have copied from a book called "R in action" by Robert Kabacoff. You can find a free PDF online.

In R, an object is anything that can be assigned to a variable. This includes constants, data structures, functions, and even graphs. An object has a mode (which describes how the object is stored) and a class (which tells generic functions like print how to handle it).

A data frame is a structure in R that holds data and is similar to the datasets found in standard statistical packages (for example, SAS, SPSS, and Stata). The columns are variables, and the rows are observations. You can have variables of different types (for example, numeric or character) in the same data frame. Data frames are the main structures you use to store datasets.

Factors are nominal or ordinal variables. They're stored and treated specially in R.

Vectors are one-dimensional arrays that can hold numeric data, character data, or logical data.

Matrix is a two-dimensional array in which each element has the same mode (numeric, character, or logical). Matrices are created with the matrix()function.

Arrays are similar to matrices but can have more than two dimensions.

Data frame is more general than a matrix in that different columns can contain different modes of data (numeric, character, and so on). It's similar to the dataset you'd typically see in SAS, SPSS, and Stata. Data frames are the most common data structure you'll deal with in R.

Lists are the most complex of the R data types. Basically, a list is an ordered collection of objects (components). A list allows you to gather a variety of (possibly unrelated) objects under one name. For example, a list may contain a combination of vectors, matrices, data frames, and even other lists. You create a list using the list() function.

4 Form three vectors. The first, a character vector containing the names of each day of the week. The second, a numeric vector containing 7 random numbers (hint: try using `runif(7, 1, 10)`). The third, a logical vector corresponding to whether you would typically visit work on each day of the week (TRUE or FALSE).

Save the above as a data.frame and change the column names to: day_of_week, coffee_consumed and attendance.

Change the mode of the data in the Coffee_consumed from numeric to factor.

3.6 Summarising and displaying data

'Time spent in reconnaissance is never wasted' - Napoleon (attrib).

It is always worthwhile to spend time scanning and summarising new datasets before starting formal statistical analysis. Simple methods such as studying the range, the distribution and the relationships between variables can often reveal unexpected structure or the presence of errors in a dataset.

Let's look at the data. Try some or all of the commands below.

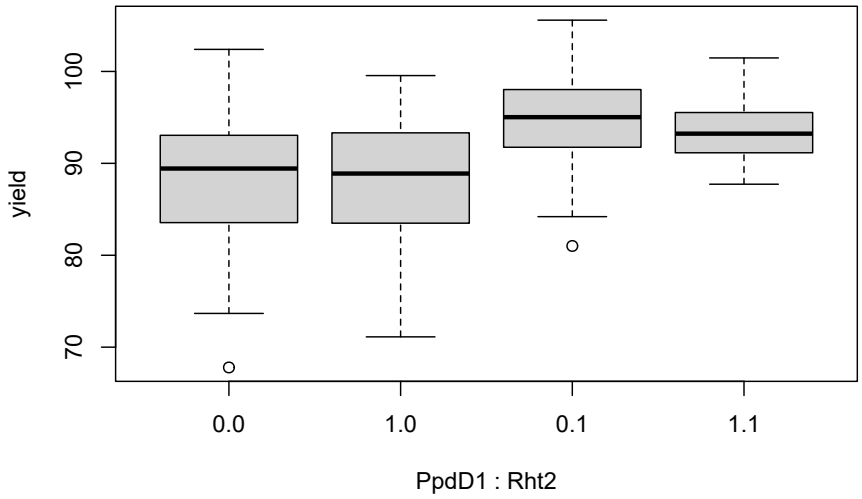
```
hist(yield)

plot(yield~year)

plot(yield~Rht2)
```

It is very easy to produce basic informative graphs in R. Titles, font sizes, colours and so on can all be changed by adding additional options to the basic commands above, but we shall introduce those as we go along.

`plot(yield~Rht2)` is not great. Try `boxplot(yield~Rht2)` which produces a box plot, or box-and-whisker plot. The central bold line is the median. The boxes show, approximately, the first and third quartiles. The lines extending from the boxes to the horizontal bars then show the distance to the maximum and minimum observations. However, any data viewed as outliers are plotted separately. These plots must not be used as statistical test for outliers. The definition of what is plotted as an outlier can be

changed with additional arguments. The default is 1.5 times the interquartile range (the width of the middle 50% of observations).

```
boxplot(yield~PpdD1+Rht2)
```



We now have boxplots for the four classes of +/- ppd and +/- rht2 alleles. The group "0.1" carries the wild-type ppdD1 allele for late flowering and the dwarfing allele at rht2, and would appear to be the highest yielding, and largest, group of lines. Substitute FT (flowering time) for yield in the boxplot to get something a little more exciting.

To find out what options are available for commands, switch from the "Plots" tab to "Help", then type in the name of the command you are interested in. The help is often cryptic, but with practice, its interpretation becomes easier.

A useful command for displaying relations between all variables in a data set is "pairs", we'll introduce the command and then refine it. Try running:

```
pairs(TG_data_for_day_1)
```

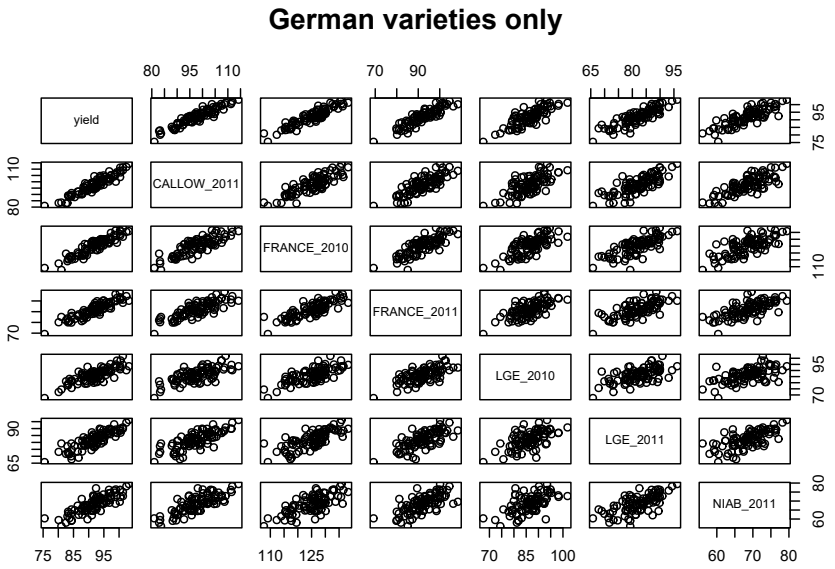
It produces an error message? Some of the data are non-numeric. This is being caused by the names in the "variety" column. We'll remind ourselves what the names of the data columns are:

```
names(TG_data_for_day_1)
```

```
##  [1] "variety"     "year"        "ORIGIN"      "Rht2"        "PpdD1"
##  [6] "yield"       "CALLOW_2011" "FRANCE_2010" "FRANCE_2011" "LGE_2010"
## [11] "LGE_2011"    "NIAB_2011"   "FT"          "HT"          "AWNS"
```

We are only interested, in plotting the yield data, ignoring the other traits for now. We'll also select the subset of lines which were registered in Germany. These are labeled DEU in the origin column. Finally we'll add a title:

```
pairs(TG_data_for_day_1[(ORIGIN=="DEU"),6:12],main="German varieties only")
```



This is easier to interpret and we can see that the yield of German varieties is strongly correlated across all the sites. Note the use of the square brackets `[ ]` to reference rows and columns of our data set. We have seen this in the last section too. As revision:

- `[1,2]` refers to a single cell - row one, column 2.
- `[2:5,]` refers to rows 2 to 5 inclusive, and to all columns.
- `[,-3]` refers to all rows and all columns except column 3.
- `[,c(1,4:6)]` refers to all rows and to columns 1, 4, 5, and 6.

You cannot mix omission of rows or columns by using "-" with inclusion using anything else, but otherwise this provides you with a flexible way of selecting subsets of data.

`c(... , ... , ... ,)` as used above, is a method of concatenating data into a single entity and is used quite extensively. It can also be used as a method of entering small amounts of data directly into R:

```
fred<-c(1:2, "Buckle my shoe")
fred
```

```
## [1] "1"             "2"             "Buckle my shoe"
```

Another way of selecting a subset of data is to use the subset command

```
fred<-subset(TG_data_for_day_1,select=c(variety,ORIGIN,yield))
head(fred)
```

```
##    variety ORIGIN  yield
## 1   AARDEN    DEU 103.23
## 2 AARDVARK    GBR  88.55
## 3    ABELE    GBR  90.76
## 4      ABO    FRA  82.24
## 5   ACCESS    GBR  96.66
## 6    ACCOR    FRA  90.05
```

The new command `head`, gives the same result as: `fred[1:6,]`. It prints the first 6 rows.

In R, the simplest non-graphical method of generating a summary of data is:

```
summary(TG_data_for_day_1)[,1:6]
```

```
##    variety               year         ORIGIN               Rht2
##  Length:376         Min.   :1946   Length:376         Min.   :0.0000
##  Class :character   1st Qu.:1988   Class :character   1st Qu.:0.0000
##  Mode  :character   Median :1997   Mode  :character   Median :1.0000
##                     Mean   :1994                      Mean   :0.5508
##                     3rd Qu.:2003                      3rd Qu.:1.0000
##                     Max.   :2007                      Max.   :1.0000
##                                                       NA's   :22
##      PpdD1            yield
##  Min.   :0.0000   Min.   : 67.79
##  1st Qu.:0.0000   1st Qu.: 88.60
##  Median :0.0000   Median : 92.63
##  Mean   :0.2261   Mean   : 91.83
##  3rd Qu.:0.0000   3rd Qu.: 96.32
##  Max.   :1.0000   Max.   :105.58
##
```

Again, to save space, I've limited this command to columns 1 to 6, by using: `[,1:6]`.

For a numeric column such as yield, the output typically shows:

- Min. : the minimum value
- 1st Qu.: the first quantile
- Median : the median
- Mean : the sample average
- 3rd Qu.: the third quantile
- Max. : the maximum value.
- NA's : the number of missing values

The first quartile, the median and the third quartile give the values of the observations, 1/4, 1/2, and 3/4 of the way down a sorted list of each variable. These values, together with the minimum, maximum and average, give a simple assessment of the distribution of the traits. You can also just get the summary information for one variable by using:

```
summary(HT)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   57.52   69.77   73.79   75.25   80.12   97.15
```

The summary statistics given collectively by summary are also available as separate commands, listed below (substitute the variable name of your choice for "x"):

```
mean(x)

median(x)

quantile(x)

min(x)

max(x)
```

Note that `quantile` returns the quartiles. This is the default for the command, which can be altered.

This is an important quirk of R:

```
mean(Rht2)
```

```
## [1] NA
```

R does not ignore missing values by default (although some functions do). Because Rht2 has some missing data (the NAs) R returns "NA" rather than the mean of the available data. This is often irritating, but it is safe: you will not forget that you have missing data. The rather longwinded way to cope with this is as follows:

```
mean(Rht2,na.rm = TRUE)
```

```
## [1] 0.5508475
```

The additional logical variable `na.rm` (meaning: not available, remove) is set to the value TRUE. In English: remove the NA values before calculating the mean. A number of other commands require this option too.

One dangerous exception to the default behavior of R is the command length, which counts the number of entries in a vector:

```
length(Rht2)
```

```
## [1] 376
```

The value of 376 is returned: this includes the missing values, which is logical but often misleading. To count the number of observations, excluding the NAs:

```
sum(!is.na(Rht2))
```

```
## [1] 354
```

Breaking this down, `is.na()` is the command to return the logical value TRUE or FALSE depending on whether each value of a variate exists or not.

```
is.na(Rht2)[1:5]
```

```
## [1] FALSE  TRUE FALSE FALSE FALSE
```

I've only shown the first five logical values, but this would continue for every row. The use of `!is.na()` switches this around to return TRUE if the value is NA (ie does not exist). `!` is the R symbol to negate the following argument. (We introduced `!=` to mean "not equal to" earlier.)

```
!is.na(Rht2)[1:5]
```

```
## [1]  TRUE FALSE  TRUE  TRUE  TRUE
```

For arithmetic purposes, the logical TRUE has a value of 1 and FALSE has a value of 0, `sum()` returns the sum or total, so `sum(!is.na(Rht2))` returns the value of 354.

Some other useful summary commands are listed below:

`sum(yield,na.rm=T)` returns the total yield

`sum(yield,na.rm=T)` returns the of yield variance

`sd(yield,na.rm=T)` returns the standard deviation.

`rowMeans(dataset,na.rm=T)` returns means across rows of a dataset

`colMeans(dataset,na.rm=T)` returns means across columns of a dataset

`rowSums(dataset,na.rm=T)` returns totals across rows of a dataset

`colSums(dataset,na.rm=T)` returns totals across columns of a dataset

These commands have all been listed including the `na.rm=T` option. If the dataset is complete this option need not be included.

5 Let's say we measured leaf length from 10 plants from one of our plots. For whatever reason one leaf was missing from the measurements:

Create this vector: `leaf_sizes<-c(30.1, 32.3, 36.8, 34.1, 32.0, NA, 32.4, 30.1, 29.9, 32.6)`

Plot a histogram showing the distribution of our leaves. Then add a title and figure legends (measurements are in cm), see `help(hist)` if you get stuck.

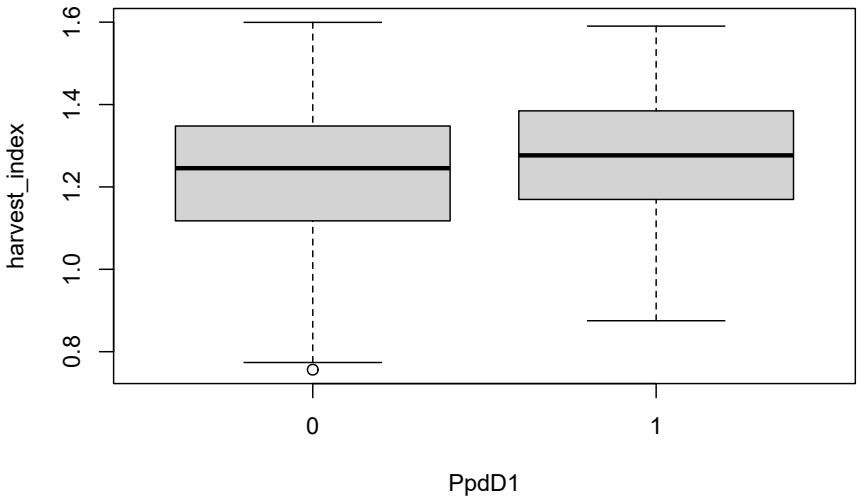
What's the mean of the 9 leaves?

It's pointless really to keep the NA in the vector here, produce a vector of 9 numbers without the NA.

3.7 Calculations and data manipulation

Once data have been read into R, the vectors (columns of data) which contain each variable can be manipulated in the same manner as individual numbers. For example:

```
harvest_index<-yield/HT
boxplot(harvest_index~PpdD1)
```



The late flowering allele (coded 1) seems to have a slightly higher harvest index. Whether this difference is likely genuine or just due to chance is something shall see how to test shortly.

3.7.1 Sorting data

Sorting data and inspection of high and low values is also of assistance in detecting errors. In R this is carried out using the command sort:

```
sort(yield) [1:5]
```

```
## [1] 67.79 71.12 73.67 74.12 74.38
```

To save paper I've added the `[1:5]` to only produce the first 5 values, or in this case the lowest 5 values.

More generally, it is usual to sort a block of data with respect to one or more columns. Routinely, this may be more easily achieved in Excel. In R: First select yield and some descriptors for ease of display:

```
TG.sub<-subset(TG_data_for_day_1,select=c(variety,ORIGIN,year,yield))
```

Then display the top few lines of the new dataframe:

```
head(TG.sub[order(ORIGIN,year),])
```

```
##          variety ORIGIN year yield
## 190         IBIS    DEU 1963 89.61
## 234 MIRONOVSKAJA    DEU 1963 84.43
## 45        ASTRON    DEU 1965 92.19
## 97        CARIBO    DEU 1968 88.29
## 310      SENATOR    DEU 1970 95.76
## 229       MERKUR    DEU 1973 86.37
```

We used `order(ORIGIN,year)` to sort the whole dataframe, indexed by country of origin and then year of release. Alphabetically, the German varieties are first.

3.7.2 Relationships between variates.

The correlation coefficient ranges from zero to one and measures the strength of the relationship between two variables:

```
cor(yield,year)
```

```
## [1] 0.6218021
```

We can enter a block of data and generate a table of correlation coefficients in a single command. There are some traps for the unwary however:

```
cor(TG_data_for_day_1[,4:6])
```

```
##       Rht2      PpdD1      yield
## Rht2     1         NA         NA
## PpdD1   NA  1.0000000 -0.1556021
## yield   NA -0.1556021  1.0000000
```

```
cor(TG_data_for_day_1[,4:6],use="complete")
```

```
##             Rht2      PpdD1      yield
## Rht2   1.0000000 -0.2461724  0.5021948
## PpdD1 -0.2461724  1.0000000 -0.1728991
## yield  0.5021948 -0.1728991  1.0000000
```

```
cor(TG_data_for_day_1[,4:6],use="pairwise.complete")
```

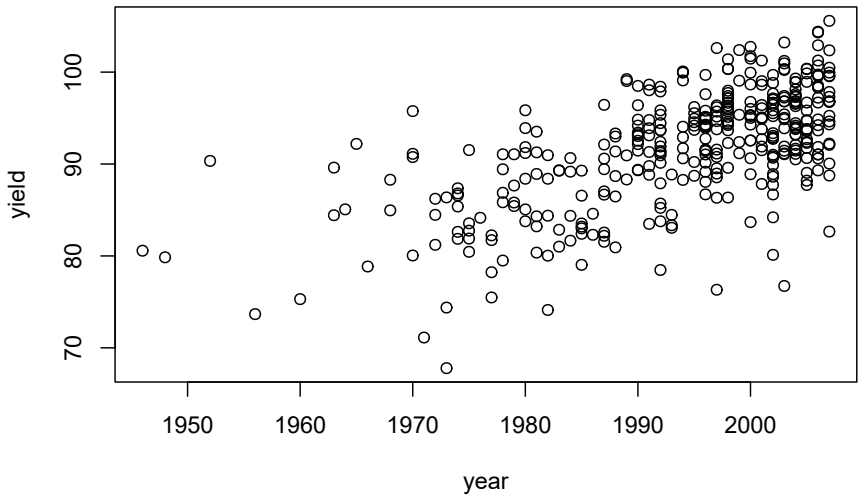
```
##             Rht2      PpdD1      yield
## Rht2   1.0000000 -0.2461724  0.5021948
## PpdD1 -0.2461724  1.0000000 -0.1556021
## yield  0.5021948 -0.1556021  1.0000000
```

The first call gives no results for Rht2 because it has missing data. The second method uses "complete observations", that is to say only records with no missing data for any field. This is the default method for some commercial statistical software systems. In the example here, 22 records of data are discarded. The third method does not discard complete records. It excludes from the analysis only those pairs of observations in which at least one of the pair is NA. In this example, correlations among PpdD1 and yield will be based on all varieties as there are no missing data for these two traits. This option can be particularly useful with extensive sets of genotype data: even if marker calling rates are high, with multiple markers it may be rare for a single individual or line to have no missing data.

In passing, we note that the calculation of a correlation coefficient is not usually a particularly sensible or conventional way to study the relationship between two binary variables (here Rht2 and PpdD1). More conventional would be to tabulate the data in a 2 x 2 contingency table. However, for marker data, the squared correlation coefficient is very commonly used as a measure of linkage disequilibrium between two loci, so the example given here has some justification (although a 2 x 2 table of observations is still informative and is demonstrated later on).

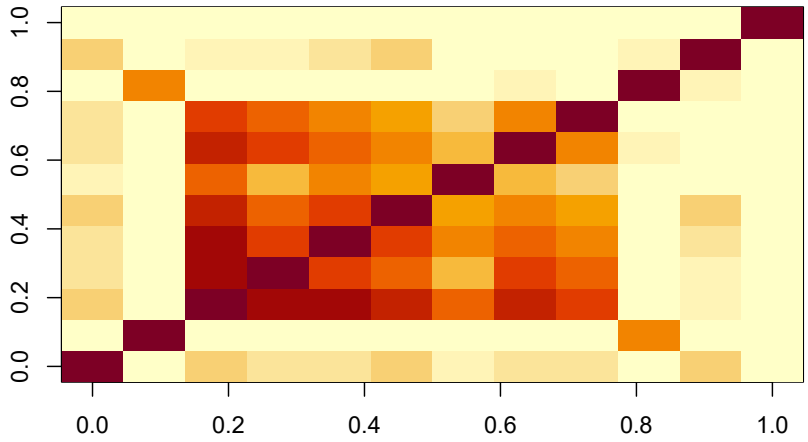
Correlation coefficients are a simple way of quantifying relationships between two variables. However, as we have already commented, it is often better to visualise the data in a scatter plot:

```
plot(year,yield)
```



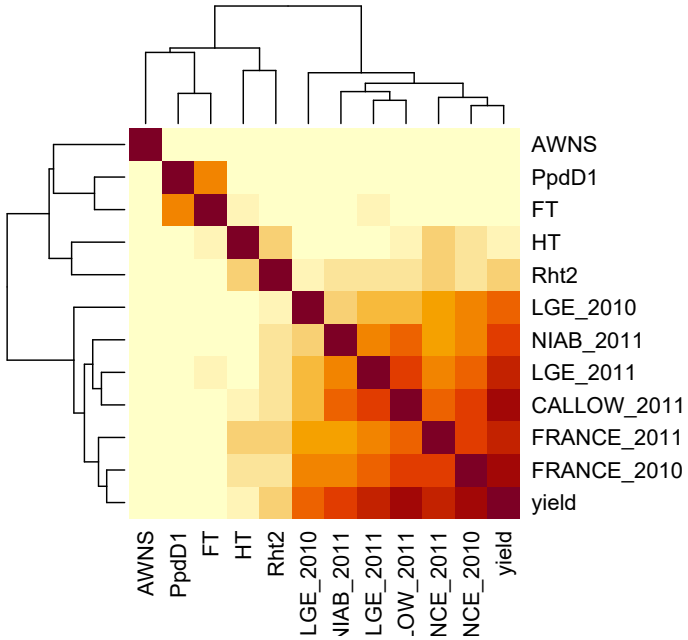
R also provides other methods of displaying these correlations graphically. `Image` creates a coloured grid of a matrix with colours depending on the values in the matrix. First we shall save the correlation matrix, working only on the trait data:

```
TG.cor<-cor(TG_data_for_day_1[,c(-1:-3)],use="pairwise.complete")
image(TG.cor^2)
```



Various colours are schemes are possible: type help(image)for more detail. Note we have squared the correlation coefficient - so all values are positive - otherwise the interpretation with both large negative and large positive correlations present can become difficult. Here, the block of pale colours in the centre of the plot indicates very high correlations among the yield measurements, with lower correlations among the other traits. The function `heatmap()` is a useful extension to this which also plots a dendogram showing estimated relationships between entries.

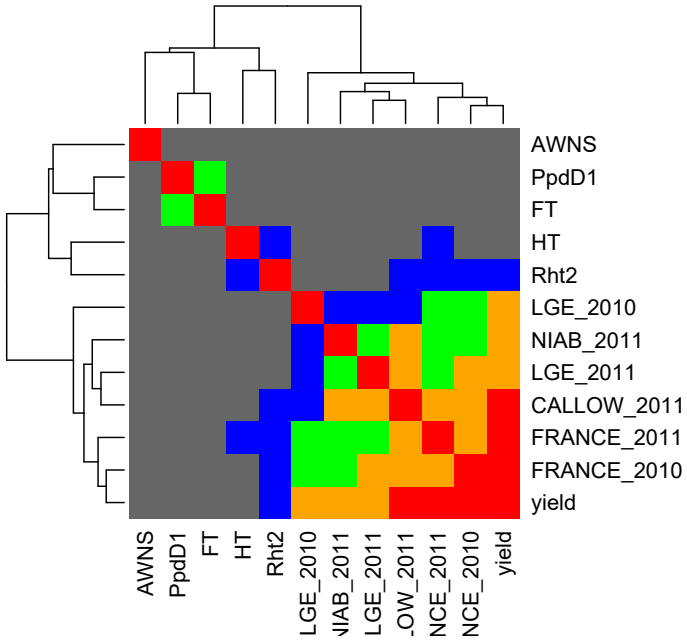
```
heatmap(TG.cor^2,symm=T)
```



This plot alters the row and column order. `symm=T` is required for heatmap to recognize that it is working on a symmetrical matrix.

The interpretation seems clear. The yield measurements correlate highly. PpdD1 and flowering time are associated, as are Rht2 and height. The relationships show up even better using a different colour scheme (which I found someone else using and copied):

```
heatmap(TG.cor^2,symm=T,col=c(grey(0.4), "blue", "green", "orange","red"))
```



This highlights a weak relationship between height and yield, at least in some environments.

3.7.3 Exporting your script and workspace

This is not as straightforward as you would think as the script and workspace (your data in the Environment window) need to be saved separately.

To save your workspace (the working environment you have created) in your working directory, you can just click:

Session / Save Workspace As

Or you could run:

```
save.image(file="File_name_for_your_data.RData")
```

Note that this will save an `.RData` file.

Next save the script you have created in the *script editing window* as an R file. You can then come back to your results by loading both the R file and the saved workspace. You simply save the script by clicking on *File / Save*. Or you could press: **Ctrl + S**

To reload your workspace you can go to: *Session / Load Workspace*. Or use `load("myfile.RData")` if you are in the correct working directory. To open your script file, you would simple click on it and it will open RStudio (or open in RStudio if you already have the program open).

3.7.4 Saving a data table

There are two simple methods of this, the simplest is to cut and paste from R to Word or Excel, exactly as you would for any other Windows application.

However, a better method would be to use R to do it. If you want to export a table you have created as a csv file (other formats are available):

```
#create a subset of the main data, in this case we just take varieties with a GBR origin
GBRorigin<-subset(TG_data_for_day_1, ORIGIN=="GBR")

#save this as a new csv file
write.csv(GBRorigin, "this will be the file name.csv")
```

That file will be saved in your directory now. Notice my use of # here. You will see this a lot in R. Putting a # at the start of a line means that R won't try to run that line, which means that you can add notes to yourself within your scripts.

6 You now should of seen the use of `subset()` several times. This is a very useful function which subsets large data.frames to smaller ones based on criteria. Firstly use `subset()` to form a new data.frame of only the unawned lines. Ask for help if you are stuck.

Re-order your new data.frame by origin.

Using your new data, form a plot showing the relationship between height (HT) and yield. Use `cor()` to test the correlation between the two variables.

3.8 Basic statistical analysis

3.8.1 The t-test

The t-test is a simple and robust method to test if the difference in means between two samples, or the difference between the mean of a sample and a known constant, is statistically significant. In other words, does the difference look too large to have occurred as a result of bad luck in selecting the samples for analysis?

t = difference / standard error of the difference.

For large sample sizes, a value of t > 1.96 will only occur by chance, in the absence of any genuine difference, in about 5% of experiments. A value > 1.96 is therefore judged to

be improbably large: the difference in means is declared to be statistically significant at the 5% level.

The t-test assumes that the sampling error of the difference being tested is normally distributed. In real data sets, this condition is often met. Firstly, the trait being measured is itself often normally distributed, and secondly, even if the trait has a non-normal distribution, mean trait values will be close to normal provided the sample size is moderately large (greater than about 10).

For example, the plot below shows the distribution of 1000 numbers. Each number was generated by taking the mean of 10 uniformly distributed random numbers. It is clear that although the distribution of original random numbers was very non-normal, the mean of a sample of 10 such numbers is pretty close to normal. In fact, in the early days of computing, normally distributed random numbers were often generated in this way. The tendency for the distribution of means to be normally distributed is called the Central Limit Theorem. It explains the popularity of the normal distribution in statistics and also the tendency for many traits in nature to be roughly normally distributed - for example if variation in a phenotype results from variation at multiple genes, the phenotype itself will often inevitably be normally distributed.

The t-test is very simply invoked in R. To test the difference in means between yield in Germany in the two years of testing:

```
t.test(LGE_2010,LGE_2011)
```

```
##
##  Welch Two Sample t-test
##
## data:  LGE_2010 and LGE_2011
## t = 9.8055, df = 745.63, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  4.019585 6.032010
## sample estimates:
## mean of x mean of y
##  85.79497  80.76918
```

The output provides a value for t and its associated p-value to test the significance of the difference in means, and also the means themselves. In addition, 95% confidence intervals are provided. These refer to the difference between the two means. Statisticians can get quite hot under the collar about what, exactly, 95% confidence intervals actually are. We can state that over a long lifetime of calculating 95 % confidence intervals for parameter estimates, they will have included the true parameter value in 95 % of cases. It is best not to worry too much about this.

Note that the degrees of freedom (df) is 745.63 and not a whole number. This is because the default setting for R is to assume that, whether or not the means of the two groups

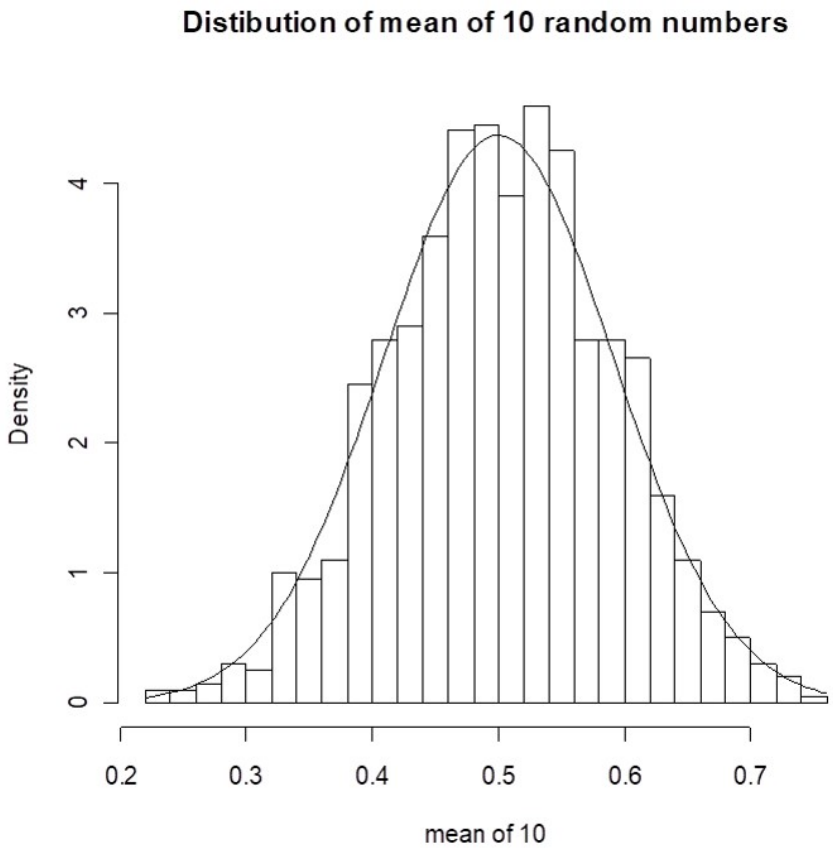


Figure 3.2: Central Limit Theorem example

being tested are different, the variances themselves are different. In accounting for this we end up with fractional degrees of freedom. This is the Welch variant of the t-test - stated in the first line of the output.

To test whether the variances in the two groups are similar, we can use a variance ratio test, or F test - dividing one variance by the other and estimating whether the deviation from the expected value of 1 is attributable to chance or is indicative of something else:

```
var.test(LGE_2010,LGE_2011)
```

```
##
##  F test to compare two variances
```

```
##
## data:  LGE_2010 and LGE_2011
## F = 0.85785, num df = 375, denom df = 375, p-value = 0.1381
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.700429 1.050643
## sample estimates:
## ratio of variances
##          0.8578466
```

The F-ratio of 0.86, with 375 and 375 degrees of freedom is not significant (p-value 0.1381).

If var.test(LGE_2011,LGE_2010) was called (the order of the two trials is reversed), the F-ratio would be 1.17 but the p-value would be unchanged. When testing the ratios of two variances in an F test, care is required because we are not testing whether the numerator variance is significantly greater than the denominator, which is the usual use of the F-ratio in an nova. Here we are carrying out a two-tailed test of significance rather than a one-tailed. Fortunately using var.test() takes care of this for you. If you were to do this by hand; dividing the larger variance by the smaller, then looking up the probability associated with that value of F in tables (or you could use an R function, p(f), then the probability would need to be doubled.

Since the variances are not significantly different (IE they are homogeneous), `t.test` can be called in a form to take this into account:

```
t.test(LGE_2010,LGE_2011,var.equal = T)
```

```
##
##  Two Sample t-test
##
## data:  LGE_2010 and LGE_2011
## t = 9.8055, df = 750, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  4.019595 6.032001
## sample estimates:
## mean of x mean of y
##  85.79497  80.76918
```

This is the more usual form for the t test - as given in most text books. It is more powerful test than Welch's variant, provided the variances are homogeneous. Note that the degrees of freedom is now an integer.

A more biologically interesting comparison is whether Rht2 and PpdD1 have an effect on yield. The problem here is that data for the two groups to be compared (yields for

one allele and yields for the other) are no longer in separate variables. To tell R that data to be analysed are in one variate but are described by data in another, we use the tilde operator (~) introduced in the section on syntax:

```
t.test(yield~PpdD1)
```

```
##
##  Welch Two Sample t-test
##
## data:  yield by PpdD1
## t = 3.0435, df = 136.67, p-value = 0.002806
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.8154943 3.8410701
## sample estimates:
## mean in group 0 mean in group 1
##        92.35522        90.02694
```

The difference in mean yield between the two PpdD1 alleles is significant, but quite small. PpdD1 could have a direct effect on yield or it could lie close to a QTL for yield, or the significant result could be due to something we don't know about concerning the origins of the varieties under test. Disentangling trait-marker associations due to the presence of a closely linked QTL from other spurious causes of association is the challenge of association genetics. We can also see if the effect of PpdD1 is consistent across all sites. There are six sites, in columns 6-12 of the dataset. We could analyse each site by editing the name of the variable before running the command, but we can automate this procedure using a loop.

This would be achieved by running:

```
for(i in 6:12) { print(t.test(TG_data_for_day_1[,i]~PpdD1)) }
```

Give this a go on your laptops.

What do you make of the output? It's possible to select individual components of the output (such as the means and p-values) so we can summarise many analyses concisely.

`for(i in 6:12)` means repeat the commands which follow a number of times by varying an index variable, here "i", from 6 to 12 (seven times in total), incrementing it by 1 each repeat. Then, because we are analysing have TG.data.for.day.1[,i], on each iteration [,i] is substituted with [,6] then [,7] ... [,12] and all our analyses are done. We are beginning to write scripts!

The commands we want to loop over does not have to be restricted to a single line, provided they are enclosed in curly brackets. Note that as these commands are run within a loop, the output does not get echoed to the screen unless we include the "print" statement as above.

3.8.2 Linear Regression

We have already come across the command to correlate two traits: `cor()`. To fit a straight line to a data set we use the R command `lm()`; for linear model. Suppose we want to study the effect of year on yield:

```
lm(yield~year)
```

```
##
## Call:
## lm(formula = yield ~ year)
##
## Coefficients:
## (Intercept)         year
##   -583.2088       0.3386
```

The output is somewhat sparse. A feature of R, in contrast to many statistical packages is that by default it does not deliver multiple pages of output by default, from which you may only wish to extract a single figure. Here the output gives you the best fitting straight line:

$$yield = -583.4 + (0.3386 * year)$$

More output is available but we need to be explicit that we wish R to produce it. First we shall rerun the analysis, but save the results:

```
yield_year_regression<-lm(yield~year)
```

No output is generated. We can produce some using `summary()`:

```
summary(yield_year_regression)
```

```
##
## Call:
## lm(formula = yield ~ year)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.1783  -3.2680   0.7123   3.1559  12.6978
##
## Coefficients:
```

```
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -583.20878   43.96487  -13.27   <2e-16 ***
## year           0.33855    0.02205   15.35   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.915 on 374 degrees of freedom
## Multiple R-squared:  0.3866, Adjusted R-squared:  0.385
## F-statistic: 235.8 on 1 and 374 DF,  p-value: < 2.2e-16
```

The most interesting part of the output is given at the end: the F-statistic and p-value for the significance of the regression - hugely significant in this example. The Multiple R-Squared is the proportion of the total sum of squares accounted for by the regression. It is also the square of the correlation coefficient between yield and year of origin. The Adjusted R-squared is the proportional reduction in variance after fitting the regression (i.e. 1-residual variance/total variance). Both these figures give an indication of how effective the regression has been in accounting for the observed variation: a significant regression does not imply that a relationship is particularly important (although it is here). Equally, with very small experiments, large proportions of variation may be accounted for, but the regression is still non-significant. This is generally an indication that you should have designed a larger experiment.

A more conventional display of the regression analysis is given as:

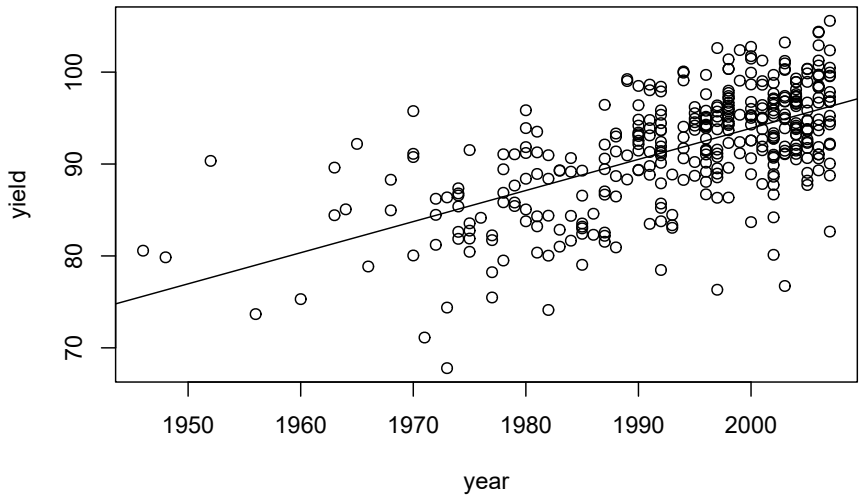
```
anova(yield_year_regression)
```

```
## Analysis of Variance Table
##
## Response: yield
##            Df Sum Sq Mean Sq F value    Pr(>F)
## year        1 5694.7  5694.7  235.75 < 2.2e-16 ***
## Residuals 374 9034.0    24.2
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This uses the `anova()` command. To R, anova is the name given to a form of tabular output. Formally, the analysis of variance itself is just a particular type of multiple regression analysis, and that is exactly how R treats it, as we shall see shortly.

We can look at a plot of the data with our fitted line a follows:

```
plot(year,yield)
abline(yield_year_regression)
```



When using plot, give the name of the variate you want plotted on the x axis comes first. The additional command, `abline()`, adds the best fitting straight line. There is a lot of variation around the best fitting line, but as we have varieties from three different countries of origin, and we have tested them in three different countries, this is not surprising.

Finally, we look at how to extract residual values and fitted values from a regression. Large residual values for particular observations are often of use in searching for errors in data. Also, identification of the individuals or varieties responsible for large residuals may sometimes suggest some other factor which needs to be considered in the analysis. Fitted values and residuals are extracted as:

```
fitted(yield_year_regression)[1:5]
```

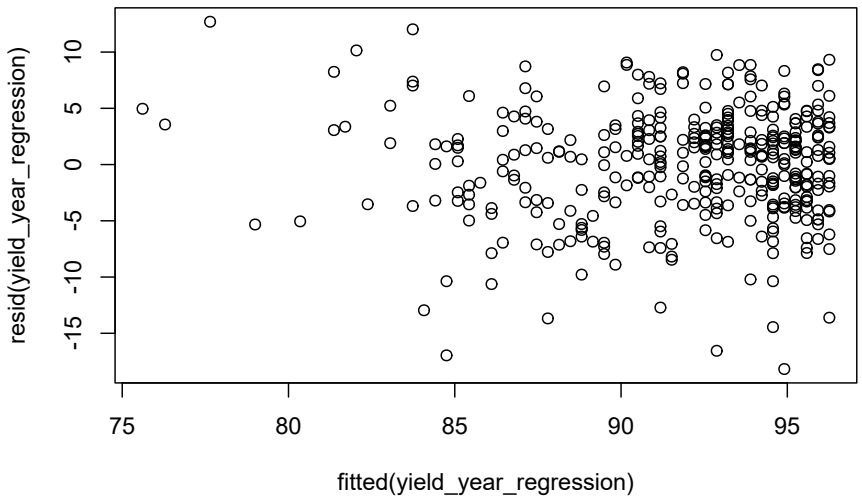
```
##        1        2        3        4        5
## 94.90825 92.87695 83.73608 86.10594 93.21550
```

```
resid(yield_year_regression)[1:5]
```

```
##         1         2         3         4         5
##  8.321746 -4.326949  7.023919 -3.865936  3.444500
```

Note I've only plotted values 1 to 5 (`[1:5]`) to save space. These could be saved to other variables, or plotted:

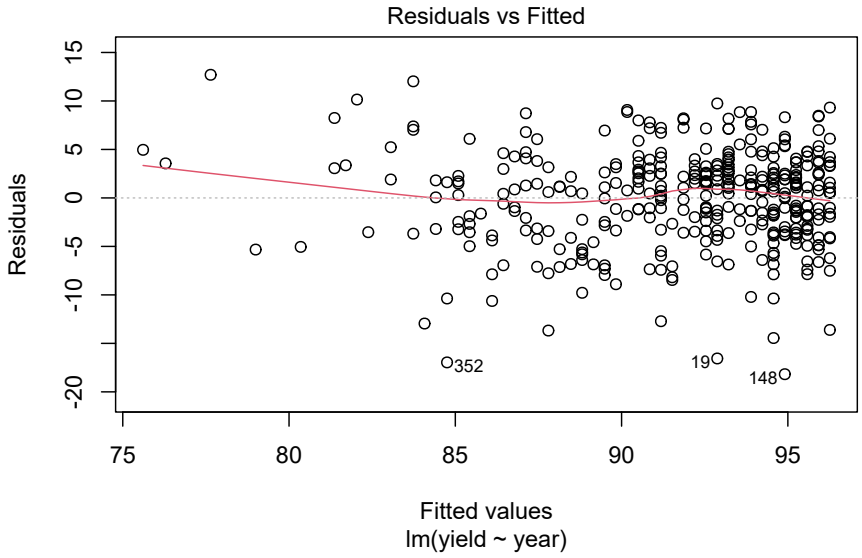
```
plot(fitted(yield_year_regression), resid(yield_year_regression))
```

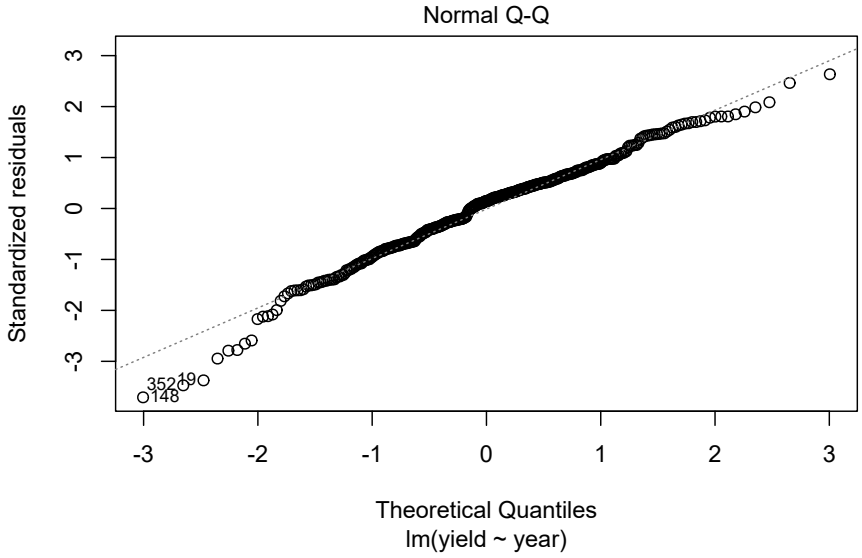


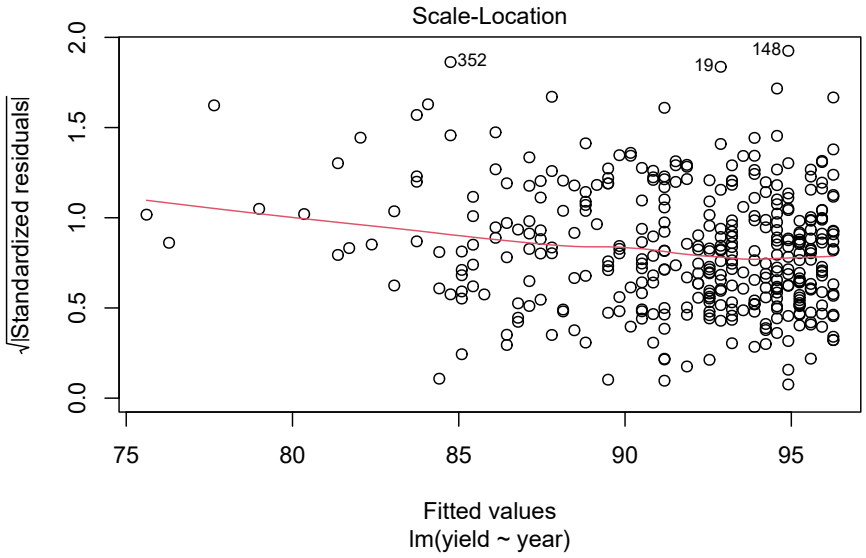
A plot of residuals against fitted values is often informative. If large residuals tend to be associated with large fitted values, for example, this indicates that the error variances are not homogenous and we treat our results with more caution. Residuals which increase in magnitude with increasing fitted values are often an indication that transforming the data to logarithms before analysis may be warranted.

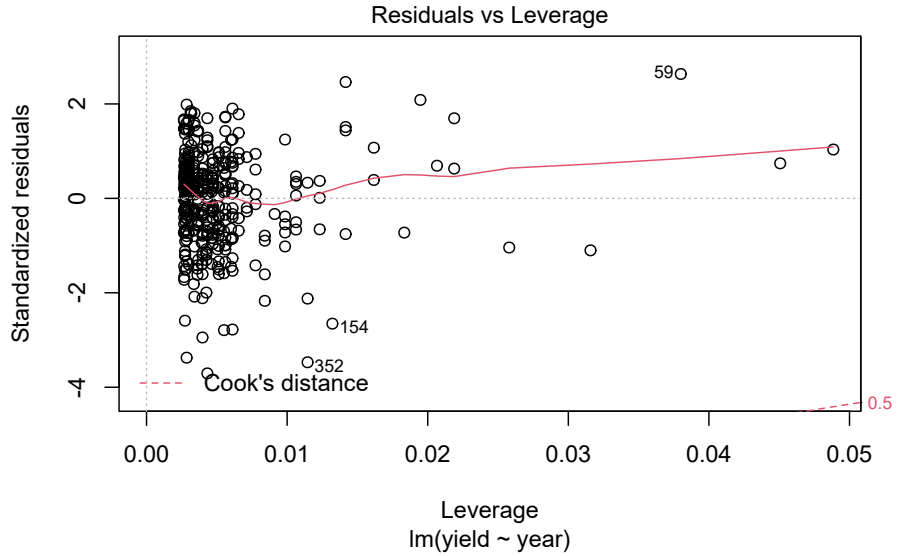
For linear regression in R, however, we can produce four of diagnostic plots very easily:

```
plot(yield_year_regression)
```


Residuals vs Fitted
Residuals
15
10
5
0
-5
-10
-20
352
19
148
75
80
85
90
95
Fitted values
lm(yield ~ year)


Normal Q-Q
Standardized residuals
3
2
1
0
-1
-2
-3
352
19
148
-3
-2
-1
0
1
2
3
Theoretical Quantiles
lm(yield ~ year)





The first two plots are the most important and the easiest to interpret:

The first plot is of residuals against fitted values as previously described. There is nothing untoward here.

The QQ plot takes the residuals, sorts them and standardises them by dividing them

by the standard deviation, so they have a mean of zero and a variance of 1. It is possible to work out what the expected value of these sorted normal deviates should be if the error distribution was truly normal. Plotting the observed against the expected gives you a QQ plot (QQ for quantile-quantile). For well behaved data, the points should lie on a straight line with a gradient of one. The data here look acceptable.

The Scale-Location plot is essentially looking at the same thing as the plot of residuals against fitted values.

The Residuals vs Leverage shows the strength of data points on the parameters of the regression. Data points with very high or low values tend to have more effect (leverage) on the regression equation. This can be quantified. If a data point with high leverage also has a big residual (positive or negative) it is an indication that the data for that observation wants checking for errors, and maybe even deleting. Cook's distance, which is also mentioned in this plot, is a measure of the improvement in error sum of squares if a particular data point were dropped. In the plot above, there is nothing to be checked: the data are good.

3.8.3 Multiple regression

Multiple regression in R requires little more than simple linear regression. The `lm()` command is still used. All that is required is to specify a more complex model using the syntax described in the Basic R Syntax section of this guide:

```
yield_mult_reg<-lm(yield~year+ORIGIN+PpdD1+Rht2+FT+HT+AWNS)
yield_mult_reg
```

```
##
## Call:
## lm(formula = yield ~ year + ORIGIN + PpdD1 + Rht2 + FT + HT +
##     AWNS)
##
## Coefficients:
## (Intercept)         year    ORIGINFRA    ORIGINGBR        PpdD1         Rht2
##   -460.4045       0.2639      -0.5753       0.9456      -0.2138       2.2100
##          FT           HT         AWNS
##      0.2335      -0.1403      -0.4815
```

Thus yield is predicted as:

-461.2409
+ 0.2645 x year
+ -0.5637 if French origin
+ 0.9510 if UK origin + -0.2157 x Pppd1

+ 2.2112 x Rht2
+ 0.2311 x FT
+ -0.1402 x HT + -0.6402 x Awns

Note this form of analysis is acceptable for PppD1 and Rht2, SNPs and other binary markers provided the markers are coded numerically. 0 and 1 are ideal codes, but 0 can sometimes be confused with a missing value, especially if data are to be analysed in packages other than R. 1, 2 coding is also acceptable. Numeric coding cannot be used at all for multiallelic markers since it implies that alleles coded with a higher number have a higher value than those with a lower number.

Note the order in which the variates are supplied to lm does not affect the estimated coefficients:

```
lm(yield~ year+ORIGIN+FT+Rht2+PpdD1+HT+AWNS)
```

```
##
## Call:
## lm(formula = yield ~ year + ORIGIN + FT + Rht2 + PpdD1 + HT +
##     AWNS)
##
## Coefficients:
## (Intercept)         year    ORIGINFRA    ORIGINGBR           FT         Rht2
##   -460.4045       0.2639      -0.5753       0.9456       0.2335       2.2100
##       PpdD1           HT         AWNS
##     -0.2138      -0.1403      -0.4815
```

Will give the same answer. However, this is not the case for estimates of significance:

```
anova(yield_mult_reg)
```

```
## Analysis of Variance Table
##
## Response: yield
##            Df Sum Sq Mean Sq  F value    Pr(>F)
## year        1 5177.9  5177.9 253.1041 < 2.2e-16 ***
## ORIGIN      2  663.2   331.6  16.2102 1.879e-07 ***
## PpdD1       1  121.4   121.4   5.9356  0.015348 *
## Rht2        1  554.0   554.0  27.0786 3.375e-07 ***
## FT          1   33.0    33.0   1.6122  0.205051
## HT          1  172.5   172.5   8.4344  0.003921 **
## AWNS        1    5.1     5.1   0.2475  0.619158
## Residuals 342 6996.4    20.5
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(lm(yield~ year+ORIGIN+FT+Rht2+PpdD1+HT+AWNS))
```

```
## Analysis of Variance Table
##
## Response: yield
##            Df Sum Sq Mean Sq  F value    Pr(>F)
## year        1 5177.9  5177.9 253.1041 < 2.2e-16 ***
## ORIGIN      2  663.2   331.6  16.2102 1.879e-07 ***
## FT          1  164.7   164.7   8.0498  0.004822 **
## Rht2        1  543.2   543.2  26.5504 4.349e-07 ***
## PpdD1       1    0.5     0.5   0.0261  0.871836
## HT          1  172.5   172.5   8.4344  0.003921 **
## AWNS        1    5.1     5.1   0.2475  0.619158
## Residuals 342 6996.4    20.5
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In the two analyses we've swapped the order in which PpdD1 and FT were fitted. In the first anova table, the sum of squares for PpdD1 is 121.4 and 33.0 for FT. In the second analysis, the sum of squares is 0.5 for PpdD1 and 164.7 for FT. The sums of squares for Rht2 has changed a bit but all the others, and most importantly the Residual SS, are unchanged. In unbalanced experiments such as this, where combinations of PpdD1 and height (and most other things too) are not all equally represented, the results from the analysis of variance depend on the order in which the terms are represented in the model. However, there is an easy way to interpret this table. Taking output from the last analysis a line at a time:

The effect for year is fitted first, and is found to be statistically significant:

year `P < 2.2e-16 ***`

After fitting the year, origin it is fitted next,

ORIGIN `P =  1.867e-07 ***`

then FT:

FT `P =  0.004813 **`

The p-value of 0.0049 for FT is the significance for FT, *after* accounting for any effect of year and origin. By the time we get to PpdD1, its effect after fitting everything above it in the table is non-significant:

PpdD1 `P = 0.871809`

In designed experiments, where different combinations of treatments and factors are usually equally represented, or balanced, the order in which terms are fitted makes no difference; the terms are said to be *orthogonal*. Balance not only has the

property of making the terms orthogonal, it also makes the arithmetic very much easier. This was very important before the advent of readily available computers. However, the requirement for balance, solely from the point of view of data analysis, is now no longer required and many contemporary designs for variety trials (for example alpha-designs) are not balanced and would be impossible to analyse without a computer. A readable account of a contemporary approach to experimental design is given in the book Mead, Gilmour and Mead: "Statistical Principles for the Design of Experiments".

In this example, the interpretation is clear: PpdD1 is known to be the major locus determining flowering time in wheat, but it doesn't control all the variation. If we had to choose between Ppd or FT in the model, we would chose FT because its sum of squares is bigger: we account for more of the variation in the data. But here it makes sense to fit Ppd first and then to test if there is a residual effect of FT on yield. If you have time, try the same thing for Rht2 and HT.

Our prior knowledge isn't always so good and the interpretation and order of fitting isn't always so clear. Selecting the order in which terms are fitted, and selecting which terms to include in the final model and which to exclude is something of an art, which we shall not develop here. There are formal methods to assist in this process. These too are not covered here, but are available within R. Generally, with genetic analysis, it is usual to account for variation attributable to causes other than genes or markers first, and then fit the genetic effects. This is a conservative approach. If we had followed it here we would not have detected an effect for PpdD1.

Note that `resid()`, `fitted()`, `plot()` and `summary()` work for multiple regression exactly as for simple linear regression. Before we leave regression, compare these two simple analyses of the relationship between yield and PpdD1

```
t.test(yield~PpdD1,var.equal=T)
```

```
##
##  Two Sample t-test
##
## data:  yield by PpdD1
## t = 3.0463, df = 374, p-value = 0.002481
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.8254243 3.8311401
## sample estimates:
## mean in group 0 mean in group 1
##        92.35522        90.02694
```

```
anova(lm(yield~ PpdD1))
```

```
## Analysis of Variance Table
##
## Response: yield
##            Df  Sum Sq Mean Sq F value   Pr(>F)
## PpdD1       1   356.6  356.61    9.28 0.002481 **
## Residuals 374 14372.1   38.43
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The p-values are identical. For a regression analysis on a single variate with only two values or classes, the two tests are equivalent. In fact:

$t^2 = F$

In this case

$3.0463^2 = 9.280.$

3.8.4 The analysis of variance

In the t-test, we test if the difference between two treatment means is statistically significant. This is a special case of the Analysis of Variance in which we test if differences among multiple treatments are jointly statistically significant. We could compare multiple treatments by carrying out multiple t tests, but this increases the risk that at least one test will be declared significant by chance alone - the so called problem of multiple testing. In addition the interpretation of results becomes increasingly complex. (There is, in fact, an R command that automates this procedure and includes an adjustment for multiple testing: pairwise.t.test). The omnibus test for significance of all means, considered together, that the Analysis of Variance offers is therefore of great value.

The principal of the Analysis of Variance is that, in the absence of any genuine difference among means, the variability among those means can be predicted from the variability from observation to observation within each treatment. This argument is little more than saying that the variance of a mean is just the variance among the observations that contribute to that mean divided by the number of observations contributing to the mean:

$V\hat{x} = Vx/n$

We therefore have independent estimates of $\hat{x}$, one directly from the means, and one from the within treatments variance (/n). If there are no true differences in the means of the treatments, these two estimates of variance are expected to be the same though

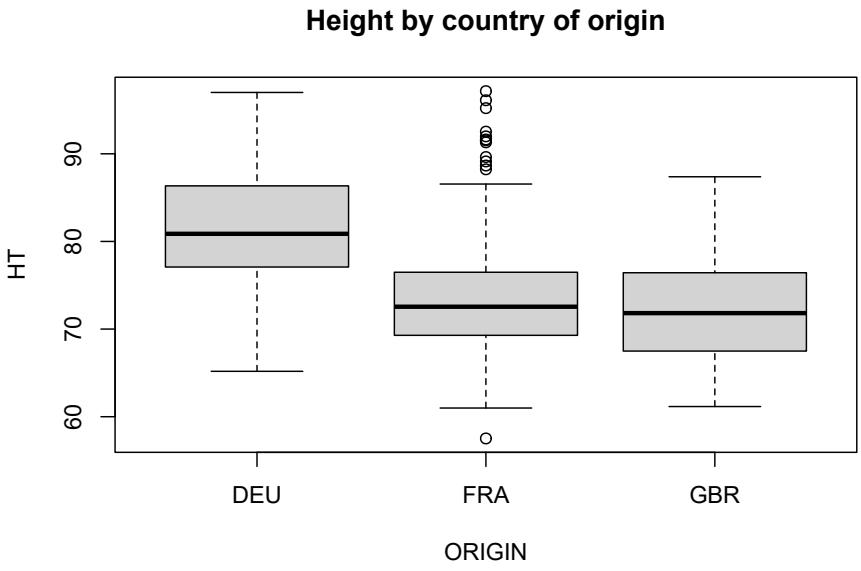
they will still differ through sampling variation. However, if the means differ, in addition, through differential responses to the treatments we no longer expect them to be the same. Rather:

$V\hat{x} > Vx/n$

The test for statistically significant differences among the means is therefore a variance ratio, or F test: the variance among treatment means is divided by the expected variance calculated within treatments. This is an oversimplification: differences in the number of observations within treatments must also be taken into account and with more complicated experimental designs the analysis is also more complicated, but the basic principal remains the same: differences in means inflate the estimate of variance between treatment means compared to the independent estimate of variance from observations within treatments.

We'll start with a simple example. Does height differ significantly between countries of origin?

```
boxplot(HT~ORIGIN,main="Height by country of origin")
```



It's clear from this that there is a difference - German lines tend to be taller. The command for an analysis of variance is just like that for linear regression:

```
anova(lm(HT~ORIGIN))
```

```
## Analysis of Variance Table
##
## Response: HT
##            Df  Sum Sq Mean Sq F value    Pr(>F)
## ORIGIN      2  4274.3 2137.17  47.307 < 2.2e-16 ***
## Residuals 371 16760.6   45.18
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

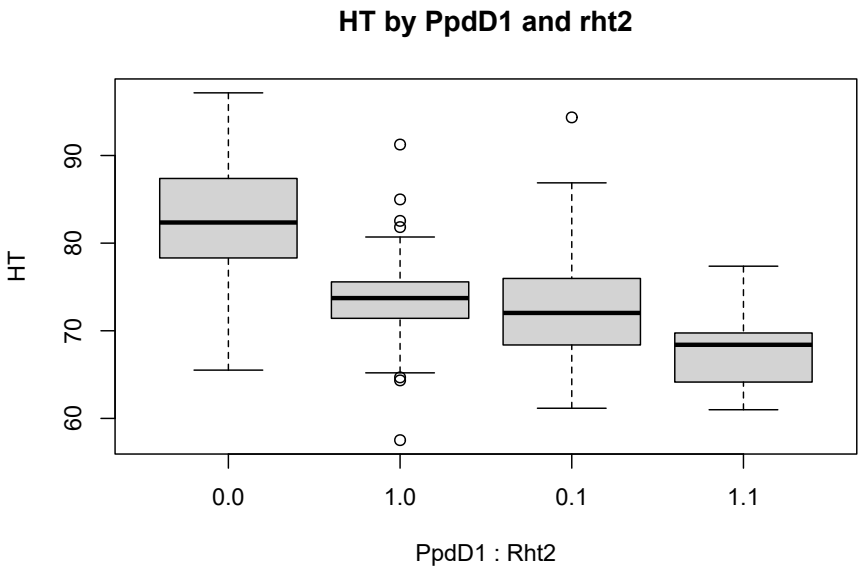
There are three countries of origin: so we have a test with 2 degrees of freedom. That is because if we know a line didn't come from France or Germany, then we know (for this dataset) that it must have come from the UK: there are only two independent countries of origin. We've seen this before: in the earlier multiple regression analysis of yield we included origin and saw that it gave an effect for the UK and an effect for France but not for Germany. The German effect was the default - the mean - to be adjusted by the appropriate effect if a line wasn't German. In fact if we had two extra columns of data, one with zeros everywhere except if a variety was from the UK, when we had a 1, and the second with zeros except if a variety was French, when we had a 1, then including these two columns in the analysis instead of "origin" would give exactly the same answer. And if we swapped out the "UK column" and inserted a new "German" column, we would still get exactly the same answer. Try it if you don't believe me!

To get more information from our analysis of variance, we can use all the methods introduced earlier for linear and multiple regressions: as stated, the analysis of variance is just a special case of multiple regression.

Let's finish off the analysis of variance with something more interesting. We know that the three countries tend to have different requirements for optimum flowering times and height. France tends to require earlier flowering lines and many varieties carry the PpdD1 early allele to achieve this. German lines tend to be taller because German wheat is often treated wheat with a grown regulator which reduces height, so there is less requirement for varieties to be bred for reduced height, commonly by using a dwarfing allele at one of the rht loci. After adjusting HT for country of origin, can we detect the effects of these major phenological loci? Do they interact?

Let's have a look at the data first:

```
boxplot(HT~PpdD1*Rht2,main="HT by PpdD1 and rht2")
```



Note the coding within the command to indicate we wanted to see four classes. 0.0 represents the late Ppd allele and the tall rht2 allele. 1.0 is the Ppd early and Rht2 short allele, and so on. It's clear that both loci have a big effect on height, but you can see also that although 1.1 is the shortest class, there is a bigger drop from 0.0 to 0.1 and 1.0 than there is from 1.0 and 0.1 to 1.1

Analysing the data, while taking into account country of origin?

```
anova(lm(HT~ORIGIN+Rht2*PpdD1))
```

```
## Analysis of Variance Table
##
## Response: HT
##             Df  Sum Sq Mean Sq  F value    Pr(>F)
## ORIGIN       2  4141.7  2070.8  68.2521 < 2.2e-16 ***
## Rht2         1  3691.8  3691.8 121.6777 < 2.2e-16 ***
## PpdD1        1  1576.7  1576.7  51.9647 3.575e-12 ***
## Rht2:PpdD1   1   119.8   119.8   3.9501   0.04766 *
## Residuals  346 10498.0    30.3
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It appears there is some weak evidence that Rht2 and PpdD1are interacting. But we are ignoring other confounding factors such as flowering time itself, and many possible interactions. We can fit a complete model:

```
anova(lm(HT~ORIGIN*FT*Rht2*PpdD1))
```

```
## Analysis of Variance Table
##
## Response: HT
##                 Df Sum Sq Mean Sq  F value    Pr(>F)
## ORIGIN           2 4141.7  2070.8  76.8732 < 2.2e-16 ***
## FT               1 1049.5  1049.5  38.9589 1.312e-09 ***
## Rht2             1 4681.8  4681.8 173.7978 < 2.2e-16 ***
## PpdD1            1  152.2   152.2   5.6487 0.0180309 *
## ORIGIN:FT        2  307.4   153.7   5.7049 0.0036625 **
## ORIGIN:Rht2      2   76.2    38.1   1.4146 0.2444857
## FT:Rht2          1  308.9   308.9  11.4673 0.0007928 ***
## ORIGIN:PpdD1     2  133.7    66.8   2.4813 0.0851744 .
## FT:PpdD1         1   24.3    24.3   0.9025 0.3428055
## Rht2:PpdD1       1    5.5     5.5   0.2033 0.6523338
## ORIGIN:FT:Rht2   2  113.2    56.6   2.1016 0.1238698
## FT:Rht2:PpdD1    1   36.2    36.2   1.3455 0.2469017
## Residuals      334 8997.4    26.9
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

3.9 Categorical data - *the chi-squared test*

Suppose we wish to test if there is an association between Rht2 and PpdD1. They are not linked, but perhaps breeders have favoured particular combinations of +/- at the two loci. Or an association might arise because of the way the varieties have been selected for inclusion in the panel of lines. The standard method of analysis of such data is the contingency chi-squared test. First we need to format our data into a table:

```
rht.ppd.table<-table(Rht2,PpdD1)
rht.ppd.table
```

```
##     PpdD1
## Rht2   0   1
##    0 106  53
##    1 170  25
```

From this table, we can see that the allele frequency of allele 1 at PpdD1 is:

(53+25)/(106+170+53+25) or 0.22

and similarly the allele frequency of allele 1 at Rht2 is 0.55

If these loci are independent of each other in this sample, the allele carried by a variety at Rht2 will be independent of the allele carried by the same variety at PpdD1. In this case, the expected frequency of (Rht2 allele 1 + PpdD1 allele 1) individuals will be 0.22 x 0.55 or 0.12. The predicted number of "11" haplotypes is therefore 0.12 x (106+170+53+25) or 42. We have observed 25. This same exercise can be carried out for each of the other three combinations of alleles at the two loci. (These combinations are commonly called haplotypes, though this term strictly only applies if the two loci are linked). If deviations between observed and expected numbers are sufficiently large, we draw the conclusion that genotypes at the two loci are not independent of each other. The statistical test is a chi-squared with 1 degree of freedom, calculated as:

$$\chi^2 = \sum \frac{(O-E)^2}{E}$$

O represents the observed numbers and E the expected.

This chi-squared test is simply carried out as:

```
chisq.test(rht.ppd.table)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  rht.ppd.table
## X-squared = 20.275, df = 1, p-value = 6.706e-06
```

The result is obviously hugely significant; the p-value is vanishingly small. Do you think this is evidence that Rht2 and PpdD1 are genetically linked?

Chi squared tests can have a problem with low numbers of observations in some cells. Try this:

```
rht.ppd.table.small<-round(rht.ppd.table/10)
rht.ppd.table.small
```

```
##      PpdD1
## Rht2  0  1
##    0 11  5
##    1 17  2
```

We've created a table 1/10 of the size of the original. `round()` rounds real numbers to integers in the standard manner (with 1/2 always rounded down).

```
chisq.test(rht.ppd.table.small)
```

```
## Warning in chisq.test(rht.ppd.table.small): Chi-squared approximation may be
## incorrect
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  rht.ppd.table.small
## X-squared = 1.2161, df = 1, p-value = 0.2701
```

If you run this, you should see this warning message:

Chi-squared approximation may be incorrect

The warning message is given because some cells have expected counts less than 5. Under this threshold, there is a chance that the chi-squared test will give misleading results. Note it is the *expected* count that matters, not the *observed* - which can legitimately be zero and often is when we are considering closely linked SNPs for example, where we may observe only two or three of the four possible haplotypes. There will be more on this later in the course when we come to discuss linkage disequilibrium. We can extract and examine the expected values:

```
chisq.test(rht.ppd.table.small)$expected
```

```
## Warning in chisq.test(rht.ppd.table.small): Chi-squared approximation may be
## incorrect
```

```
##     PpdD1
## Rht2    0   1
##    0 12.8 3.2
##    1 15.2 3.8
```

```
chisq.test(rht.ppd.table.small)$observed
```

```
## Warning in chisq.test(rht.ppd.table.small): Chi-squared approximation may be
## incorrect
```

```
##     PpdD1
## Rht2  0  1
##    0 11  5
##    1 17  2
```

The expected count in one cell is less than five. There is an approximate correction, Yates's correction, that takes into account the potential failure of the test statistic to follow a chi-squared distribution when the expected numbers were low. This has been applied automatically. To see results without Yates' correction:

```
chisq.test(rht.ppd.table.small,correct=F)
```

```
## Warning in chisq.test(rht.ppd.table.small, correct = F): Chi-squared
## approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  rht.ppd.table.small
## X-squared = 2.3314, df = 1, p-value = 0.1268
```

The `correct = F` option turns Yates' correction off and the chi-sq test statistic is larger.

It is increasingly common, and better, to derive the distribution of the test statistic empirically, by repeated randomisation or permutation of the observed data, followed by recalculation of test statistic. The proportion of times the randomised test statistic is greater than or equal to the observed test statistic is then the empirical p-value. Here the randomisation procedure is very simple: the data for Rht2 are randomised over subjects, while the data for PpdD1 are held constant. For a chi-squared test, R automates this procedure:

```
chisq.test(rht.ppd.table.small,simulate.p.value = T,B=10000)
```

```
##
##  Pearson's Chi-squared test with simulated p-value (based on 10000
##  replicates)
##
## data:  rht.ppd.table.small
## X-squared = 2.3314, df = NA, p-value = 0.2068
```

The value of B, the number of randomisations, can be set by the user. The default value is 2000, but I've run 10,000 which is still very quick when the number of observations is small.

An alternative test to the chi squared for contingency tables with small expected numbers is Fisher's exact test. This compares the probability of observing the actual 2x2 table, calculated from the multinomial distribution, with the cumulated probability

of observing other, less likely, tables. This probability is also calculated from a multinomial distribution with the same marginal frequencies: here the overall allele frequencies at SNP1 and SNP2). In R:

```
fisher.test(rht.ppd.table.small)
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  rht.ppd.table.small
## p-value = 0.2075
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##  0.02194865 2.00594857
## sample estimates:
## odds ratio
##  0.2692599
```

The p-value is very similar to that from permutation testing. The odds ratio is much favoured in medical statistics and in epidemiology. It need not concern us.

The contingency chi-squared test will easily accommodate larger tables; a 10 x 10 table for example. In the context of genetic markers, this could be used to test for association between pairs of microsatellites with multiple alleles. Fisher's exact test is computationally hard to calculate, even on a computer, for large contingency chi-squared tables, so empirical methods are often favoured.

3.10 More useful information

3.10.1 Plotting Graphs

Graphical methods were introduced in context in the sections on summary statistics and on basic statistical analysis. The following methods have been used:

`hist()` produces a histogram

`plot()` produces a scatter graph

`abline()` adds a line of best fit to a scatter graph

`boxplot()` produces a box-and-whisker plot.

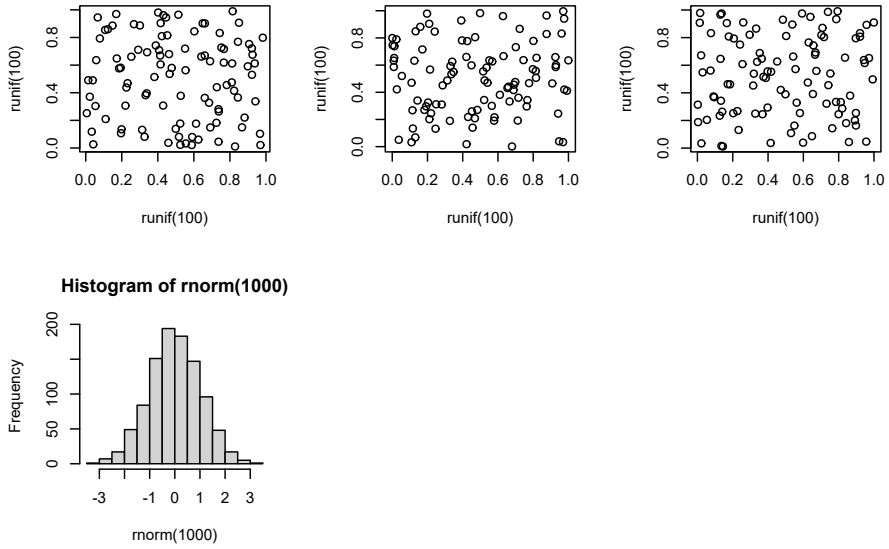
`pairs()` produces a matrix of scatter plots.

Control of many graphics parameters - font sizes, colours and so on - is given by the command `par()`. Type `help(par)` for more details. One useful option is:

`par(mfrow=c(x,y))`

This will plot graphs successively in a window x graphs wide by y graphs deep. Once the panel of graphs is complete, the next graph will overwrite the whole window and start filling the panel again. For example, the following commands set up a panel two graphs deep by three wide and fills in the first four slots.

```
par(mfrow=c(2,3))
for (i in 1:3) plot(runif(100),runif(100))
hist(rnorm(1000))
```



As you can see, the graphs need not all be of the same type. The related version of the command `par(mfcol=c(x,y))` will produce a graphical window of the same dimensions but fill it in column order rather than in row order.

A benefit of RStudio over native R is that graphs are saved and you can scroll back and forwards over previous efforts and export them, either as files or to the clipboard. In native R, the export of graphs is no more complicated, but they are overwritten so you cannot scroll to inspect previous plots but must recreate them.

Although R generates graphs very quickly and simply, the labeling and formatting are often not ideal. To alter graph titles, include the options:

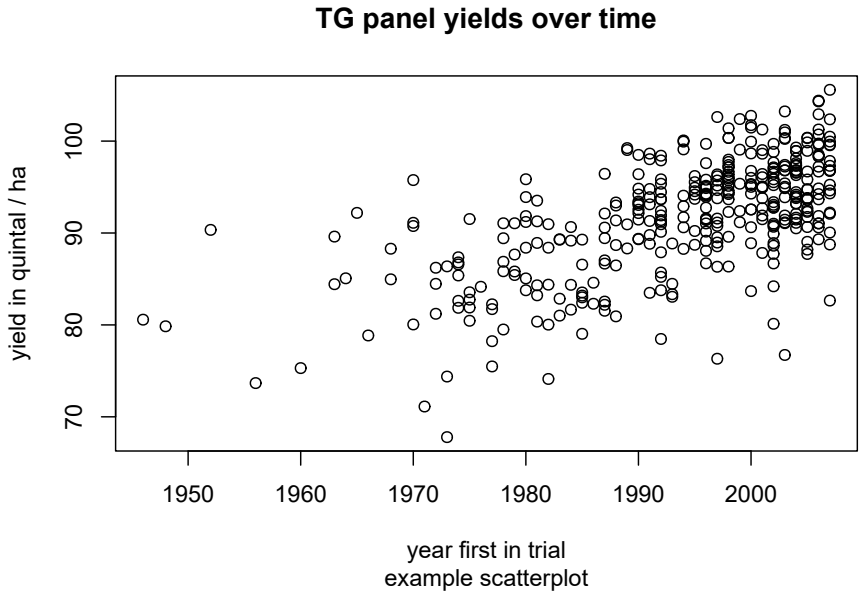
`main="Main title"`

`sub="subtitle"`

```
xlab="x-label"

ylab="y-label"
```

For example:

```
#switch back to plotting a single plot
par(mfrow=c(1,1))
#make plot
plot(year,yield,main="TG panel yields over time",sub="example scatterplot",ylab="yield in quintal / ha",xlab="year first
```

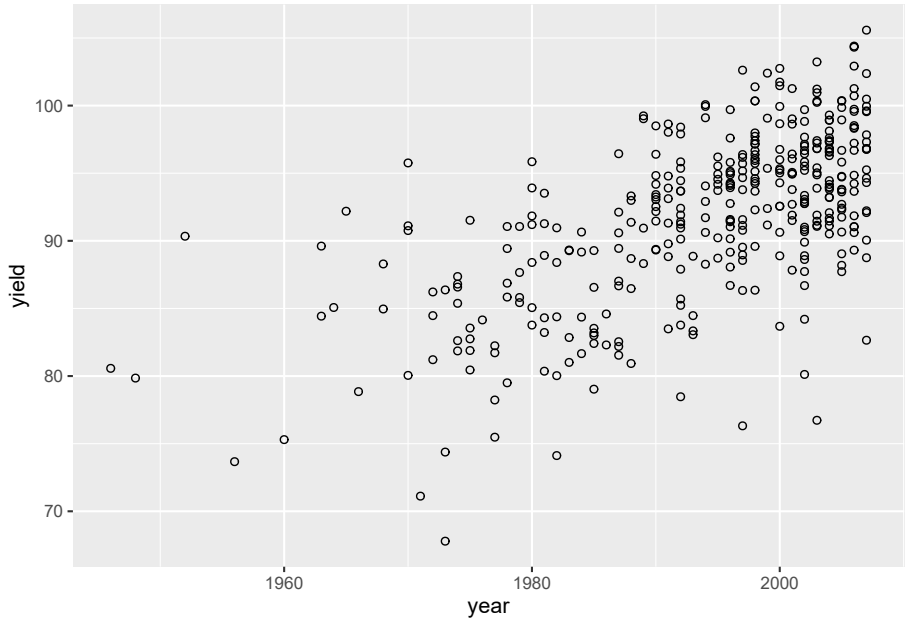


We have only used the basic plotting commands which come with the core R package. Very highly rated, and increasingly used is a package called "ggplot2" `http://ggplot2.org/`. This is generally included as part of the tidyverse too.There is no time go into this in detail. Select ggplot2 from the "Packages" tab in the bottom right window to active the package. If not installed, click on the "Packages" tab in the bottom right window, then click "install" and enter ggplot2 as the package you wish to install. Alternatively, you can run this line: `install.packages("ggplot2")`, then load the package from your library:

```
library("ggplot2")
```

Then:

```
ggplot(data=TG_data_for_day_1,aes(x=year,y=yield)) + geom_point(shape=1)
```



`aes` stands for aesthetics. If you think this look better and is easier, then ggplot2 is for you. See the web site for more details.

There's plenty more you can do with ggplot2. There are some good cheat sheets[3] available too.

3.10.2 Probability distributions

In this section we describe how to use R to look up the p-value associated with the most commonly used test statistics and how to look up test statistics associated with p-values.

Firstly the p-value associated with chi-squared:

```
pchisq(3.84,1,lower.tail=F)
```

```
## [1] 0.05004352
```

[3] https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

The parameters given to `pchisq`, in order, are:

`3.84` value of the test statistic.

`1` the degrees of freedom

`lower.tail` If set to `T` (the default) the result is the cumulative distribution up to the value of the test statistic - `0.95` in the example. For significance testing we require the area of the upper tail: 1-0.95 and so set `lower.tail=F`.

To calculate a chi squared from a p-value we use the command qchisq:

```
qchisq(0.05,1,lower.tail=F)
```

```
## [1] 3.841459
```

The syntax is identical to that for `pchisq()`, so we are required to include `lower.tail=F`. Examples for the F distribution are shown below.

```
pf(3.84,1,1000,lower.tail=F)
```

```
## [1] 0.05032099
```

```
qf(0.05,1,1000,lower.tail=F)
```

```
## [1] 3.850775
```

The examples here are for 1 degree of freedom for the numerator and 1000 degrees of freedom for the denominator. The results are identical to those for a chi-squared test with 1 df. In fact, a chi-squared test with n degrees of freedom is identical to an F test with n degrees of freedom in the numerator and a very large number of degrees of freedom (ideally infinite) in the denominator.

Values for probabilities associated with a normal distribution are:

```
pnorm(1.96,lower.tail=F)
```

```
## [1] 0.0249979
```

This probability is for a standardised normal distribution: with a mean of zero and a variance of 1. The probability is for a single tail of the distribution. Generally, we would require the result for a two tailed test - the probability of values higher than 1.96 and lower than -1.96. This probability is just double that for a single tail: 0.05 in this case. Again, this is the same value as for chi-squared with 1 degree of freedom. If a variate has a standardised normal distribution, the variate squared has a chi-squared distribution with 1 degree of freedom:

$1.96^2 = -1.96^2 = 3.84$

Probabilities associated with normally distributed variables with different means and variances are produced by specifying the mean and variance.

`pnorm(q,mean=x,sd=y)` with the inverse function: `qnorm(p,mean=x,sd=y)`

`lower.tail=F` can be added if required. The values for mean and standard deviation are now user specified (substitute for x and y in pnorm and qnorm). The default values are 0 and 1.

Finally, the t-test:

```
pt(1.96,1000,lower.tail=F)
```

```
## [1] 0.02513659
```

```
pt(-1.96,1000)
```

```
## [1] 0.02513659
```

We usually carry out a two sided t test - so we require the sum of the lower and upper tail probabilities: equal to two times the single tailed probability. The inverse function follows the usual format and nomenclature:

```
qt(0.025,10000,lower.tail=F)
```

```
## [1] 1.960201
```

3.10.3 Random number generation

R has excellent random number generators for several distributions:

```
rnorm(n, mean=0, sd=1)
```

```
runif(n, min=0, max=1)

rchisq(n, df, ncp=0)

rt(n, df, ncp)
```

`rf(n, df1, df2, ncp)` and there are others. These are self explanatory except for `ncp`. This is the non-centrality parameter. For most use, this will be zero. It is used in power calculations to provide the distribution of the test statistic under an alternative hypothesis. The `ncp` is the value you would get if you plug values of parameters or observations you expect under the alternative hypothesis into the formula for the test. For example, with a t test, the value under the null hypothesis is zero. If you expect the true difference between your means to be three, then this is the `ncp`. For a chi-squared test, you would create expected numbers from expected proportions from an alternative hypothesis and calculate a chi-squared value which will be the `ncp` for power calculations.

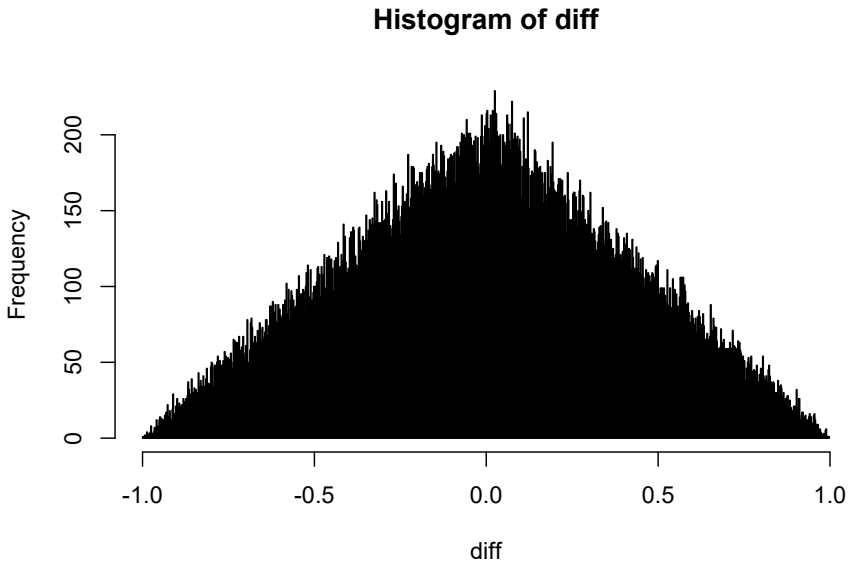
3.10.4 Loops: *second example*

A block of commands, each on a separate line, can be saved in a file and later pasted into R where it will execute (i.e. run as a program). There is, of course, much more to programming in R than this, but this is enough to get you started. The most common requirement is to repeat an operation multiple times. Below is the format for a simple loop which can be easily edited to fit your own purposes.

```
no_times<-100000
diff<-NA
for (i in 1:no_times) {
diff[i]<-runif(1)-runif(1)
}
```

This code plots the distribution of the difference in value between a pair of uniformly distributed values. It takes a while to run: you may prefer to reduce the number from 100,000. If you cut and paste the code into R you should get something like this:

```
hist(diff,breaks=1000)
```



Note that the beginning and end of the loop (or any R program block) is defined by curly brackets. The indentation isn't obligatory but is good programming practice. The program creates a vector, `diff` which is then indexed by `i`. If `diff` wasn't first created outside the loop, the program would fail.

Loops in R can run slowly. Often, they can be avoided by the use of the command `apply()`:

```
apply(TG_data_for_day_1,2,sd)
```

```
## Warning in var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm =
## na.rm): NAs introduced by coercion

## Warning in var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm =
## na.rm): NAs introduced by coercion

##     variety        year      ORIGIN        Rht2       PpdD1       yield
##          NA  11.5105231          NA          NA   0.4188383   6.2670999
## CALLOW_2011 FRANCE_2010 FRANCE_2011    LGE_2010    LGE_2011   NIAB_2011
##   7.5343555   7.6016464   8.0943785   6.7535158   7.2916383   5.1220707
##          FT          HT        AWNS
##   3.2931899   7.5069727          NA
```

The first argument to apply is the dataset or matrix we want to work on, the second is whether we want to loop over rows (1), or columns (2 - as in this example). The third

is the function you wish to loop over. So here we loop over columns to calculate the standard deviation of each.

The missing data for Rht2 has caused the standard deviation to be returned as NA. We need to include `na.rm=T` as an option. Unfortunately, there is no way I know to do this within apply. Instead we need to define our own function (which also allows us to look at how to define functions in R in general - another useful lesson in learning to program in R).

```
sd.na.rm.is.t<-function(x) sd(x,na.rm=T)
```

`function(x)` defines a set of arguments that our new function `sd.na.rm.is.t` requires. Here we only need one argument; the name of the dataset, which will be equated to x when we run the function. Following that, on the same line, is the set of commands we want to run. Here we are running an inbuilt function, `sd(x,na.rm=T)`,which will take the data vector x as the source of data.

Try: `sd(AWNS)` `sd(AWNS,na.rm=T)` `sd.na.rm.is.t(AWNS)`

So now:

```
apply(TG_data_for_day_1,2,sd.na.rm.is.t)
```

```
## Warning in var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm =
## na.rm): NAs introduced by coercion
```

```
## Warning in var(if (is.vector(x) || is.factor(x)) x else as.double(x), na.rm =
## na.rm): NAs introduced by coercion
```

```
##     variety        year      ORIGIN        Rht2       PpdD1       yield
##          NA  11.5105231          NA   0.4981119   0.4188383   6.2670999
## CALLOW_2011 FRANCE_2010 FRANCE_2011    LGE_2010    LGE_2011   NIAB_2011
##   7.5343555   7.6016464   8.0943785   6.7535158   7.2916383   5.1220707
##          FT          HT        AWNS
##   3.2931899   7.5069727   0.2716556
```

This returns the desired standard deviation of each column of data. We still get NA's and warnings for attempting to calculate the standard deviation for non-numeric data (for Variety name and country of origin). We could specify which columns of `TG_data_for_day_1` to work on to avoid this.

```
apply(TG_data_for_day_1[,-1:-3],2,sd.na.rm.is.t)
```

The `[,-1:-3]` informs R that we need all rows but to drop the first three columns. This is long winded, but functions are frequently longer sequences of commands rather than just one as in our example. The commands for a long function, which will run over many lines are bracketed with `{` and `}`, just as in loops using `for`.

3.10.5 Miscellany

Included here are some useful commands which have not so far been described and some less frequently used commands which come in handy from time to time.

`attributes(object)`

Gives information on the object which is often to help you work out what you've done.

`cbind(datasetA)` and `rbind(datasetB)`

Joins up two datasets by rows or columns. For example:

```
temp<-TG_data_for_day_1[,1] [1:5]
cbind(temp,TG_data_for_day_1[1:5,1:3]  )
```

```
##       temp  variety year ORIGIN
## 1   AARDEN   AARDEN 2003    DEU
## 2 AARDVARK AARDVARK 1997    GBR
## 3    ABELE    ABELE 1970    GBR
## 4      ABO      ABO 1977    FRA
## 5   ACCESS   ACCESS 1998    GBR
```

We have created a subset of the data with two copies of the variety names. I've limited this to the first 5 rows to save paper. There is no particular reason we would want to do this, it's just as a demonstration.

Next, we create two copies of the first three columns of the data using rbind:

```
temp<-TG_data_for_day_1[,1:3]
temp2<-rbind(temp,TG_data_for_day_1[,1:3])
head(temp2[order(temp2$variety),])
```

```
##      variety year ORIGIN
## 1     AARDEN 2003    DEU
## 377   AARDEN 2003    DEU
## 2   AARDVARK 1997    GBR
## 378 AARDVARK 1997    GBR
## 3      ABELE 1970    GBR
## 379    ABELE 1970    GBR
```

Note, that in ordering the data we have used `order(temp2$variety)` rather than `order(variety)` because we require the version of "variety"" associated with the new structure temp2, not the version of variety that we attached earlier.

`dim(x)`

Retrieve the dimensions of an array or dataset.

```
dim(TG_data_for_day_1)
```

```
## [1] 376  15
```

```
dim(temp2)
```

```
## [1] 752   3
```

`help(command)`

This opens a new window and provides help on the command. help(lm) for example, will give help on the linear modelling command that we have used extensively. The help is written in a terse and technical style however, which may be hard to understand at first but you get used to it. It is useful to see what options are available with each command - for many of the commands used in this guide more are available than have been described. Often, sufficient of the output from help makes sense to be able to get a command working by trial and error. At the bottom of the output, there are often examples of the command's use: again not always easy to follow. In Rstudio, help is available (in the same unhelpful style) directly from the windows.

`history(x)`

In native R, this opens up a window with a list of the most recently issued x commands. The default number is 25. These can be copied back into the R window and re-executed. The window with the output can be saved from the File menu to keep a record of commands issued during the R session. In RStudio, the history of commands is available directly from a tab in the top right hand window.c

`ls()  or equivalently objects()`

Lists all the variables available in the current R session. This command can also be executed from the menu in native R, selecting first "Misc", then "List objects". In RStudio it is displayed using the Environment tab in the top right hand window.

`matrix(data, nrow = x,ncol =y)`

Defines a matrix and fills it with data. For example:

```
matrix(rnorm(12),nrow=3, ncol=4)
```

```
##              [,1]       [,2]        [,3]       [,4]
## [1,] -0.233995226 -0.3368211 -0.58919496  0.5523650
## [2,] -0.866784024  1.0255816 -0.08874279 -1.7859894
## [3,] -0.009812204 -2.1193455 -0.86261735 -0.5676484
```

The `rnorm()` part has simulated 12 random numbers which have a specified normal distribution.

Note that there is no need to define the column number; it is fixed by the data and by the row number. More commonly, we would want to name the matrix. By default, all data are initially recorded as missing:

```
matrix(nrow=3,ncol=4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   NA   NA   NA   NA
## [2,]   NA   NA   NA   NA
## [3,]   NA   NA   NA   NA
```

`rep(x,y)`

Generates a repeated list:

```
rep(1:3,4)
```

```
##  [1] 1 2 3 1 2 3 1 2 3 1 2 3
```

```
rep(c('apples','pears'),2)
```

```
## [1] "apples" "pears"  "apples" "pears"
```

Remove an object:

```
rm(harvest_index)
```

Remove those objects that are no longer required.

`rm(list=ls(all.names=TRUE))` Would removes all variables, although this can be more easily achieved from the Session tab which offers more options.

Pull row names from a object (and only show first 5):

```
rownames(TG_data_for_day_1)  [1:5]
```

```
## [1] "1" "2" "3" "4" "5"
```

Pull column names:

```
colnames(TG_data_for_day_1)
```

```
##  [1] "variety"     "year"        "ORIGIN"      "Rht2"        "PpdD1"
##  [6] "yield"       "CALLOW_2011" "FRANCE_2010" "FRANCE_2011" "LGE_2010"
## [11] "LGE_2011"    "NIAB_2011"   "FT"          "HT"          "AWNS"
```

You can also write new names:

```
rownames(TG_data_for_day_1)[1:3]<-c("a","b","c")
rownames(TG_data_for_day_1)[1:10]
```

```
##  [1] "a"  "b"  "c"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
```

`seq(x)`

Generates a sequential list.

```
`seq(10)`                produces a list from 1:10.
`seq(10,12,0.25)`       produces a list from 10 to 12 in steps of 1/4
```

`sink(filename)`

Sends output to the given file rather than to the console. `sink()` with no filename will turn off the previous redirection, Useful if you are expecting lots of output. You can issue this before the command and then turn it off when the command has completed.

`source(filename)`

Runs commands from the filename - given as a variable containing the name or as the name itself in quotes.

`split(x,y)`

Partitions a variable or whole dataframe `x` into separate variables or dataframes on the basis of matches to `y`. Give this a go using:

```
split(yield,ORIGIN) [1:2]
```

In my hands, the following convoluted code will split our data into three separate data frames.

```
DEU<-data.frame(split(TG_data_for_day_1,ORIGIN)[1])
FRA<-data.frame(split(TG_data_for_day_1, ORIGIN)[2])
UK<-data.frame(split(TG_data_for_day_1, ORIGIN)[3])
dim(DEU)
```

```
## [1] 89 15
```

```
dim(FRA)
```

```
## [1] 212  15
```

```
dim(UK)
```

```
## [1] 73 15
```

```
dim(TG_data_for_day_1)
```

```
## [1] 376  15
```

The row numbers don't add up because for some countries the origin was NA.

3.10.6 My number one piece of advice

If you have reached this stage of the tutorial, no doubt you have taken on board a lot of information! It's inconceivable to remember all that information. While you can refer back to this tutorial whenever you need to for help, I find the best method is usually just to *'google it'*.

Whether you are a first time user, an intermediate user, or professional software developer, this is the universal solution to most programming/coding problems.

In terms of R issues, you can normally just google your problem in the following format. Let's say we wanted to change the position of a plot title for a figure we were forming for a publication/presentation, I would search for:

"How to change position of plot title in r"

The first hit for me is this[4]. That link shows someone asking my question and some fantastic solutions to solving the problem.

The websites stackoverflow[5] and stackexchange[6] are typically the best options and have helped me a great deal. Normally, the question you are asking has been already addressed on these websites and users have provided more solutions than you need for the question at hand. I would suggest getting to grips with the format and etiquette of these website communities before posting your own questions. The moderators/users like the questions to be asked in a certain way and can be slightly hostile if you don't meet their criteria. Then your question and their slightly rude responses remain on the internet for everyone to see. I'm speaking from past experience here!

There are some good R reddit communities that accept all type of questions (e.g. try this one[7]).

3.10.7 Learn more

Much useful information and documentation is available on the R web site[8], including the R manual "An Introduction to R."

The manuals, including "An introduction to R", are available directly from the Help menu from with R.

The 'Introductory Statistics with R' book (Dalgaard, 2008) is an excellent introduction both to R and to statistical analysis, with many simple examples.

R in a Nutshell (Adler, 2010), by Joseph Adler in the Nutshell series from O'Reilly gives a lot of detail about what and how R does what it does. An excellent reference book, but not for beginners.

3.10.8 Packages

Many are available from the Comprehensive R Archive network (CRAN) web site `http://cran.r-project.org/`. They can be installed directly from this site, or downloaded as zip files and installed later (useful if you need to install onto multiple machines). We shall be using several of these later.

These packages generally come with their own manual, often detailed. That for the R/QTL package, for example, runs to 96 pages. Although the CRAN website is the first

[4] `https://stackoverflow.com/questions/20355410/adjust-plot-title-main-position`
[5] `https://stackoverflow.com/`
[6] `https://stackexchange.com/`
[7] `https://www.reddit.com/r/Rlanguage/`
[8] `http://www.r-project.org/`

place to search for suitable packages, they are also found elsewhere and are often referred to in methodological publications or the methods sections of paper: programming in R is an expanding industry. The more popular packages (R/QTL again) may have books published about them and often course material and lecture notes can be found on the web.

It is worth reiterating that our guide to the syntax and structures used in R has been very superficial. We have mentioned the data frame and little else. Knowledge of other structures - arrays, matrices, lists - should be acquired at some stage. They are explained in "An Introduction to R" but this book is not a page turner.

3.10.9 The Tidyverse

If you are new to R, it is worth putting the effort into learning about this. The website is: `https://www.tidyverse.org/`. The book "R for Data Science" (Wickham and Grolemund, 2016), which is free online from this website is a very good introduction, including to ggplot.

3.10.10 Other resources

For the "Understanding data types and R objects" section in this tutorial, inspiration was taken from `https://datacarpentry.org/R-ecology-lesson/`. This is an excellent resource and worth checking out. Additionally, formal definitions were taken from the book "R in Action: Data Analysis and Graphics with R" (Kabacoff, 2011).

3.11 List of commands, mainly described in this guide.

3.11.0.1 General

`apply` = fast way of looping over rows and columns

`attach` = attaches a dataset to R for subsequent analyses

`attributes` = get information on the attributes of an object

`cbind` = concatenate two tables or vectors by columns

`colnames` = adds column names to a table or reads them from a table

`detach` = attaches a dataset to R for subsequent analyses

`demo` = demonstration a command (not available for most commands)

`dim` = retrieve the dimensions of an object

`factor` = convert a variable or text into a factor

`for` = loop over a a set of commands with different input values

`function` = define a function which can then be run repeatedly

`getwd` = returns the path to the working directory

`help` = returns help on a command

`history` = lists previously issued commands

`install.packages` = select a package to install

`is.na` = returns TRUE if a value is NA.

`length` = returns the number of entries in a variate

`ls` = lists all active data structures and variates

`matrix` = defines a matrix and fills it with data.

`order` = returns the order or a variate for use in a subsequent sort

`quit` = exit R

`rbind` = concatenate two tables or vectors by rows

`read.csv` = reads in data from .csv file

`read.table` = reads in data

`rep` = generates a list with repeating elements

`rm` = delete data structures and variates from R

`rownames` = adds row names to a table or reads them from a table

`searchpaths` = lists attached packages and datasets in the order in which searched.

`seq` = creates a sequence of values

`sink` = writes output to a file

`sort` = sorts data

`source` = reads commands from a file

`split` = splits data on a factor value

`subset` = selects a subset of data for subsequent analysis

`table` = defines a table: used for input into contingency chi sq tests.

3.11.0.2 Graphical

`abline` = add the best fitting straight line to a scattergram

`boxplot` = produce a Box-and-whisker plot

`heatmap` = plots a heatmap and dendogram

`curve` = add a function to a graph

`hist` = plot a histogram

`image` = colours-in a matrix according to its values.

`pairs` = plot multiple scattergrams in a matrix format

`par` = set graphical parameters

`pdf` = writes graphical output to a pdf file

`plot` = produce a scattergram

3.11.0.3 Statistics

`anova` = return an anova table from a linear model

`apply` = applies a function to rows and/or columns of a matrix

`chisq.test` = carry out a contingency chi-squared test

`colMeans` = returns means across columns of a dataset

`colSums` = returns totals across columns of a dataset

`fisher.test` = carry out a Fisher's exact test

`cor` = returns the correlation coefficient

`fitted` = returns the fitted values from a linear model

`lm` = define and execute a linear model

`mean` = returns the mean of a variate

`median` = returns the median of a variate

`minimum` = returns the minimum of a variate

`maximum` = returns the maximum of a variate

`pchisq` = returns the p-value of a chi-squared statistic

`pnorm` = returns the p-value for a normally distributed variate

`pt` = returns the p-value of a t-test

`qchisq` = returns a chi-squared statistic for a given probability

`pf` = returns the p-value of a F (variance ratio) statistic

`qnorm` = returns a normality distributed variate for a given probability

`quantile` = returns the quantiles of a variate

`qf` = returns a F (variance ratio) statistic for a given probability

`qt` = returns the t-test statistic for a given probability

`sd` = returns the standard deviation of a variate

`sum` = returns the sum of a variate

`summary` = summarise data

`resid` = returns the residuals from a linear model

`rchisq` = returns random numbers from the chisq distribution

`rf` = returns random numbers from the F distribution

`rnorm` = returns random numbers from the normal distribution

`rowMeans` = returns means across rows of a dataset

`rowSums` = returns totals across rows of a dataset

`runif` = returns random numbers from the uniform distribution

`rt` = returns random numbers from the t distribution

`t.test` = one and two sample t-test

`var` = returns the mean of a variate

`var.test` = compare two variances by an F test

Chapter 4

Trial Design.

4.1 Randomised complete block designs

Firstly, we shall use Excel to create a randomised complete block design for 10 varieties in 6 replicates. Start a new workbook. You will need three column headings: "replicate", "variety code", and "random no". First, we need to enter the trial design:

In the "replicate" column, enter 1 in the first 10 rows, 2 in the second 10 rows .... 6 in rows 52-61.

In the "variety" column, enter 1...10 in the first 10 rows, 1...10 in the second 10 rows.....1 to 10 in rows 52-61.

Next, we want to randomise the position of the varieties within replicates; each replicate must end up still containing every variety once only.

In the "random no" column, enter "=rand()" in each of the 60 rows below the header. Sort on replicate, then on random number within replicate.

Job done!

4.2 Trial design in R

We will use the R package `blocksdesign`(Edmondson., 2021) to create partially replicated and other designs. This package originated as a stand-alone program and website, a mirror of which was hosted at NIAB, but the developer, Rodney Edmondson, now only maintains the R version.

To start this tutorial, open a new RStudio screen and start a new script window. If you don't have the package already in your library you can install it by running:

```
install.packages("blocksdesign")
```

Regardless of if you've got the package installed in your library already, you will still need to load it to your current working session. You can do this with:

```
library(blocksdesign)
```

I always start my scripts with a list of library commands, stating the packages I will want to use in each analysis.

The command blocks is used to create designs:

```
blocks(treatments, replicates, blocks = NULL, searches = NULL, seed = NULL, jumps =
1)
```

- `treatments` is the number of varieties
- `replicates` is the number of replicates
- `blocks` defines the number of (incomplete) row blocks

The other options can be ignored for now, and omitted. `blocks` can be ignored too, for simple designs:

```
blocks(treatments=10, replicates = 6)
```

```
## $Replication
##    Treatments Freq
## 1           1    6
## 2           2    6
## 3           3    6
## 4           4    6
## 5           5    6
## 6           6    6
## 7           7    6
## 8           8    6
## 9           9    6
## 10         10    6
##
## $Blocks_model
##   Level Blocks D-Efficiency A-Efficiency A-Bound
## 1     1      1            1            1       1
##
## $Design
##    Level_1 plots treatments
## 1       B1     1          3
## 2       B1     2          9
```

```
## 3       B1     3          5
## 4       B1     4          5
## 5       B1     5          1
## 6       B1     6          3
## 7       B1     7          5
## 8       B1     8          1
## 9       B1     9         10
## 10      B1    10          7
## 11      B1    11          4
## 12      B1    12          8
## 13      B1    13          6
## 14      B1    14          7
## 15      B1    15          4
## 16      B1    16          1
## 17      B1    17          7
## 18      B1    18          9
## 19      B1    19          6
## 20      B1    20          5
## 21      B1    21          9
## 22      B1    22          1
## 23      B1    23          9
## 24      B1    24          1
## 25      B1    25          2
## 26      B1    26          3
## 27      B1    27          3
## 28      B1    28          7
## 29      B1    29          8
## 30      B1    30          4
## 31      B1    31          2
## 32      B1    32          8
## 33      B1    33          7
## 34      B1    34          2
## 35      B1    35          5
## 36      B1    36         10
## 37      B1    37          4
## 38      B1    38          8
## 39      B1    39         10
## 40      B1    40          2
## 41      B1    41          3
## 42      B1    42          9
## 43      B1    43          4
## 44      B1    44          2
## 45      B1    45          6
## 46      B1    46          2
## 47      B1    47         10
## 48      B1    48         10
```

```
## 49      B1    49          7
## 50      B1    50          5
## 51      B1    51          8
## 52      B1    52          4
## 53      B1    53          3
## 54      B1    54          9
## 55      B1    55          6
## 56      B1    56          6
## 57      B1    57         10
## 58      B1    58          8
## 59      B1    59          6
## 60      B1    60          1
##
## $Plan
##   Level_1 Blocks.Plots: 1 2 3 4 5 6 7 8  9 10 11 12 13 14 15 16 17 18 19 20 21
## 1      B1               3 9 5 5 1 3 5 1 10  7  4  8  6  7  4  1  7  9  6  5  9
##   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
## 1  1  9  1  2  3  3  7  8  4  2  8  7  2  5 10  4  8 10  2  3  9  4  2  6  2 10
##   48 49 50 51 52 53 54 55 56 57 58 59 60
## 1 10  7  5  8  4  3  9  6  6 10  8  6  1
##
## $seed
## NULL
##
## $searches
## [1] 84
##
## $jumps
## [1] 1
```

This will produce a version of the randomised complete block design design we produced above. That was much faster than the excel approach! It's better for us to save the result. We will save it as `RCB.save`:

```
RCB.save<- blocks(treatments=10, replicates = 6)
```

The package saves the results in a list object. You can determine what type of object you've saved by running:

```
class(RCB.save)
```

```
## [1] "list"
```

In this case, you can use `$` to access the useful output you saw printed when we ran the command before saving.

1 Try running these on your own computer:

```
RCB.save$Treatments
```

```
RCB.save$Design
```

```
RCB.save$Plan
```

4.2.1 Partially replicated designs

P-rep (partially replicated) designs in which not all entries are replicated, and augmented designs in which a small number of control varieties may be highly replicated but candidate entries are not, are useful in the early stages of plant breeding when only limited seed is available. We still require some replication both to provide an estimate of error and to give some local control of fertility effects through blocking and/or spatial analysis. The `blocksdesign` package can create these.

```
p.rep.1<- blocks(treatments=c(2,8), replicates = c(10,1))
```

The above will produce a design with 2 varieties in 10 replicates and 8 in 1. Note that the order in treatments and replicates must agree. Although this works, you should find it has produced a completely randomised design with no blocking. That is not too bad in this case, as the number of entries is so low, but it is not usually what we usually require.

```
p.rep.2<- blocks(treatments=c(2,8), replicates = c(10,1), blocks=2)
```

If we run the above we now have two blocks.

A cautionary note: many people (including me sometimes) use 'blocks' and 'replicates' interchangeably. But this only true for an randomised complete block design. In other cases it can lead to confusion when, as here, there are variable replicate numbers so that none of the block structures corresponds to 'replicate blocks' or 'complete blocks'.

2 What do you find about the allocation of the two control varieties over the blocks? As the design is small you can just look and count.

If we wanted to count control number across block in larger trial we could run:

```
table(p.rep.2$Design$Level_1, p.rep.2$Design$treatments)
```

```
##
##      1 2 3 4 5 6 7 8 9 10
##   B1 5 5 0 0 1 1 1 0 0  1
##   B2 5 5 1 1 0 0 0 1 1  0
```

This uses the function `table`. Perhaps we think that a block size of 10 is too great, but that 5 would be better.

```
p.rep.3<- blocks(treatments=c(2,8), replicates = c(10,1),blocks=4)
```

Running the above will achieve a block size of 5.

3 Either use `table` or inspect the `design` to examine the allocation of the control varieties over blocks. Does it look sensible?

If we run:

```
p.rep.4<- blocks(treatments=c(2,8), replicates = c(10,1),blocks=c(2,2))
p.rep.4$Plan
```

```
##   Level_1 Level_2 Blocks.Plots:  1 2 3 4 5 6 7
## 1      B1   B1.B1                1 2 6 2 1 1 7
## 2      B1   B1.B2                3 1 2 2 5 1 2
## 3      B2   B2.B1                9 1 1 2 2 8 1
## 4      B2   B2.B2               10 2 2 2 1 1 4
```

We have introduced an additional constraint. There are two levels of blocking: one with two blocks, each of size 10, and then one with two blocks of size five nested within each of the larger blocks. In this case, there is no difference in design between p.rep.3 and p.rep.4, but this is not usually the case.

The first level of blocking is defining complete replicates and the second incomplete blocks within replicates. The second level of blocking is labelled 1, 2 within each of the first levels. This is confusing as there are now two separate blocks both labeled 1 (and 2). It is also a potential cause of error when we come to analyse the data - we'll look at that tomorrow.

The `blocks` command is great for blocking in one dimension, which is very often all we want to do. The facility to nest blocks within one dimension has advantages: it removes much of the angst about selecting block size since you can have two or three nested blocks with little loss of precision compared to choosing the best size in the first case and a potential gain in precision compared to using the wrong block size.

4.2.2 Two-dimensional designs

`blocks` will not construct two-dimensional designs however. For this, we need to use the more flexible command `design`. We shall use this to create a classical two-dimensional model: the balanced lattice square design has a treatment number which is a square (9, 16, 25 etc.) and each treatment occurs once with every other treatment in both a row block and a column block. The block size, k, is the square root of the number of treatments. There are k+1 complete replicates and there are k blocks in each replicate. These are powerful designs, controlling error variation in two dimensions. They are also very restrictive in acceptable variety and replicate numbers.

We shall create the design for 9 varieties in 4 replicates. `design` works on the complete block structure and list of `treatments` and then allocates the `treatments` to the `blocks` optimally. This is very flexible but you need to create the list of blocks and treatments, I created the blocks structure in Excel. Read it in from the file "lattice_9.csv".

For the next steps to work you will either need to change your RStudio working directory to the folder containing the downloaded data files for Chapter 4 (which you should have downloaded to your computer from the course website). Or move the datafiles to your current working directory outside of R.

Read in the block structure from the existing file:

```
lattice.9<-read.csv("lattice_9.csv",header=T,colClasses = "factor")
```

In each replicate there are three row blocks and three column blocks.

The `colClasses = "factor"` option ensures all the columns are treated as factors rather than numbers. It is possible to create this in R with a few short commands; see examples under `help(design)` but in this case it is as easy to create the design in Excel.

We also need a factor for nine varieties replicated four times:

```
var.code<-factor(rep(1:9,4))
```

This is slight overkill in this case: there is no need to provide the replicate number as it is equal for all varieties and R just repeats the list, but it introduces the easy extension for unequal replicate numbers. For example:

```
factor(c(rep(1:7,4),rep(8,8)))
```

```
##  [1] 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 1 2 3 4 5 6 7 8 8 8 8 8 8 8 8
## Levels: 1 2 3 4 5 6 7 8
```

The above command defines a treatment structure for seven varieties in 4 replicates and one in 8 replicates.

Returning to the 3 x 3 lattice, to produce the design:

```
latt.sq<-design(treatments=var.code,blocks=lattice.9)
```

That takes slightly longer to run than the `blocks` command. The syntax is clear. Check the output makes sense.

We want to check if each variety pair occurs once in each row-block and once in each column-block. We can use `interaction` to give each of these blocks a unique code over replicates and `table` to tabulate variety codes with blocks. We shall use a new command `crossprod` to work out the number of concurrences of each variety with every other in these blocks. [In matrix terminology, crossprod pre-multiplies a matrix by its transpose.]

```
crossprod(table(interaction(latt.sq$Design$reps,latt.sq$Design$rows),
latt.sq$Design$treatments))
```

```
##
##     1 2 3 4 5 6 7 8 9
##   1 4 1 1 1 1 1 1 1 1
##   2 1 4 1 1 1 1 1 1 1
##   3 1 1 4 1 1 1 1 1 1
##   4 1 1 1 4 1 1 1 1 1
##   5 1 1 1 1 4 1 1 1 1
```

```
##   6 1 1 1 1 1 4 1 1 1
##   7 1 1 1 1 1 1 4 1 1
##   8 1 1 1 1 1 1 1 4 1
##   9 1 1 1 1 1 1 1 1 4
```

Row blocks look good. What about column blocks?

```
crossprod(table(interaction(latt.sq$Design$reps,latt.sq$Design$cols),
latt.sq$Design$treatments))
```

```
##
##     1 2 3 4 5 6 7 8 9
##   1 4 2 1 1 2 1 0 1 0
##   2 2 4 0 1 1 1 1 1 1
##   3 1 0 4 1 1 1 1 1 2
##   4 1 1 1 4 1 0 2 1 1
##   5 2 1 1 1 4 1 1 0 1
##   6 1 1 1 0 1 4 1 1 2
##   7 0 1 1 2 1 1 4 2 0
##   8 1 1 1 1 1 0 1 2 4 1
##   9 0 1 2 1 1 2 0 1 4
```

blocksdesign searches for the best design. We can make it search harder by increasing the number of `searches`:

4 Increase search number to 1000 in the design command. You can do this by adding `searches = 1000`. Does that improve the design?

Increasing search number will mean the function takes longer to run. It can be hard to know if R is still running something and you can think your computer has frozen.

When something is running, a red stop sign will appear in the toolbar of the console window (top righthand side). If you press this it will stop the run. It's also a very usefull indicator to know if something is running still.

4.2.2.1 A larger balanced lattice square

Read in the following file from your working directory:

```
lattice.25<-read.csv("lattice_25.csv",header=T,colClasses = "factor")
```

This file contains a design for 25 varieties in six replicates. Each replicate is arranged with 5 row blocks and 5 column blocks.

Define the `treatments`:

```
var.code.25<-factor(rep(1:25,6))
```

The target is that each pair of varieties should occur together once in a row block and once in a column block. (With a block size of 5, a variety can occur with 4 other varieties in a block. There are 6 replicates of each variety so there are 6 x 4 = 24 concurrences: hopefully one for every variety pair). As accuracy of comparison between varieties should be greatest when varieties occur in the same block, all pairs of varieties should be compared with equal precision if we achieve the design objective.

```
latt.sq.25<- design(var.code.25,lattice.25)
```

Efficiencies (or `efficiency` factors) can be examined as:

```
latt.sq$Treatments_model
```

```
##   Treatment.model Model.DF D.Efficiency
## 1    ~ treatments        8            1
```

```
latt.sq.25$Blocks_model
```

```
##        First_order effects  D.Effic  A.Effic       Second_order effects D.Effic
## 1           (reps)       5 1.000000 1.000000           (reps)^2       5      NA
## 2      (reps+rows)      29 0.830539 0.827728      (reps+rows)^2      29      NA
## 3 (reps+rows+cols)      53 0.659989 0.653184 (reps+rows+cols)^2     149      NA
##   A.Effic
## 1      NA
## 2      NA
## 3      NA
```

`D-efficiency` and `A-efficiency` are measures of the average precision with which the parameters in the model (for us the variety effects) are estimated. More information can be found on this here[1].

These are hard to interpret. `A-efficiency` is related to the average of the variance of the treatment estimates and is usually what we are interested in.

5 Easier is to examine the design as before. Edit the command for the 3 x 3 lattice to look at the balance of variety concurrences in row and column blocks. Have we reached the optimum design?

If you think we should improve out attempt:

```
latt.sq.25$searches
```

```
## [1] 34
```

That shows that the computer algorithm had 34 attempts at improving the design.

Overriding this:

```
latt.sq.25<- design(var.code.25,lattice.25,searches = 1000)
```

6 Inspect the output. How did we do by adding more searches?

We have a very good design, particularly if we are mainly concerned in controlling variation in one dimension. But if you need a balanced design for 25 varieties in 6 replicates, with two dimensional blocks, you are better off referring to a published catalogue of optimal designs; though you are in danger or reverting to "Procrustean design" (Mead, 1994). Or run `blocksdesign` for many more iterations; I eventually succeeded in recreating the balanced lattice.

4.2.2.2 A partially replicated design

As a final example, we shall return to p-rep and augmented designs: assume we have 36 experimental varieties, but only 12 of them have enough seed for 2 replicates. We want to test all 36. We also have a control variety which we want to test in 12 replications.

[1] `http://homepage.divms.uiowa.edu/~gwoodwor/AdvancedDesign/KuhfeldTobiasGarratt.pdf`

We therefore have (12 x 2) + (24 x 1) + (12 x 1) = 60 plots in total. We shall create a nested design with two main blocks of size 30, and 6 sub blocks of size 5. We are going to use the `design` function for this. Therefore, we need to form our `treatments`.

7 Have a go.

Hint = I've filled in some of the options below. You have to substitute the correct numbers or parameters for x, y and z:

```
aug.treats<-factor(c(rep(1:12,x),rep(y:36,1),rep(37,z)))
```

Check your answer. (e.g. is it the right length). If you are ever doing this on your own and struggle, just do it in Excel and then read it in. I've already created the blocks structure in excel, so you can read it straight in:

```
aug.blocks<-read.csv("aug_blocks.csv",header=T,colClasses = "factor")
```

8 Now run the `design` command: `aug.expt<-design(a,b)` # You substitute for a and b

Extract and examine the block structures again. How does that look? You should find there is a control occurs in every block. What is the distribution of 2-rep varieties over the sub blocks?

```
rowMeans(table(interaction(aug.expt$Design$main_blocks,aug.expt$Design$sub_blocks),aug.expt$Design$treatment
```

```
##  1.1  2.1 1.10 2.10 1.11 2.11 1.12 2.12  1.2  2.2  1.3  2.3  1.4  2.4  1.5  2.5
##    2    0    0    2    0    2    0    2    2    0    2    0    2    0    2    0
##  1.6  2.6  1.7  2.7  1.8  2.8  1.9  2.9
##    2    0    0    2    0    2    0    2
```

The combinations of main-blocks and sub-blocks that have none of the 2-rep varieties don't exist in the trial. For the ones that do, the 2-rep varieties are spread out well.

It is always worth checking the distribution of entries of blocks appears sensible; there is a risk of error in entering the design information and also a risk that the software you are using has not found the allocation of varieties to blocks that you expected.

You could produce a similar design by hand; you could start with a design for 12 varieties in 2 replicates with 6 sub-blocks of size 2 in each replicate. Then add the control to each sub block. Then add two unreplicated entries to each sub block. This should produce a very similar design. Alternatively, you could produce an alpha design for 30 entries in two replicates and then allocate varieties to codes to end up with the same even distribution.

4.2.2.3 An augmented paritially replicated design: complicated

The vignette for `blocksdesign` suggests that if you have unreplicated entries, it may be better to create a design for the replicated entries first, then fill in the unreplicated entries by hand. For example, I tried to create a design with 4 controls in 10 replicates, 55 candidates in 2 replicates and 250 entries in 1 replicate, with two big blocks of size 400 and 20 nested blocks of size 10 in each big block. The target design would have a single control in each of the 40 small blocks – slightly increasing the accuracy of the comparison between controls and candidates at the expense of reducing the accuracy among the four controls.

Running:

```
aug.p.rep<-blocks(treatments = c(4,55,250),replicates = c(10,2,1),blocks=c(2,20),searches=1000)
```

Produced a design in which the four controls concurred once or twice in the small blocks and in which two controls concurred with the two replicate entries 17 times and the other two controls concurred 18 times.

However, re-running the design as:

```
aug.p.rep.2<-blocks(treatments = c(4,55),replicates = c(10,2),blocks=c(2,20),searches=1000)
```

Created a design in which the controls never concurred, but concurred 28, 27, 28 and 29 times with the 2-rep candidates. 30 blocks had 4 entries and 10 had 3. Allocating the 1 replicate entries round up all blocks to size ten would give the desired design. (Entries within blocks would require re-randomising too.) This design will have slightly reduced the accuracy of comparisons among the four controls and increased the accuracy among candidates and between candidates and the controls. There is nothing wrong with the first deign, however, though the second is aesthetically more pleasing.

For a multi-location trials series, you can treat each location as a block, and then have nested blocks within locations as before. The locations need not be complete blocks – not all varieties are tested at all locations. This is becoming common in large plant breeding programmes. An advantage of this approach is that genomic prediction methods (discussed next week) can be used to predict the performance of the varieties in the missing locations.

4.3 Farm scale experiments

On farm experiments, using farm equipment, are experiencing a revival, partly promoted by better farming equipment with on-harvester yield assessment, GPS, remote sensing and so on. The principles of good experimental design, the three R's, still apply however. Generally, the experimental unit is large: a fields or half a field, but the

number of available experimental units available on a single farm is low. Experimental units within a farm can be treated as a single block. The size of the blocks may vary from farm to farm, however.

As an example, consider the imaginary design in this table:

Table 4.1: Farm Blocks

farm	fields
1	1
1	1
1	2
2	1
2	1
2	2
2	2
2	3
2	3
3	1
3	1
3	2
4	1
4	1
4	2
4	2
5	1
5	1
5	2
5	2
6	1
6	1
6	2
7	1
7	1
7	2
7	2
8	1
8	1
8	2
8	2
8	3
9	1
9	1

9	2
9	2
9	3
10	1
10	1
10	2
10	2

There are 10 farms, with between two and three fields available per farm. If a field is big enough, it is split in half – two 'plots' per field. This number could be larger for bigger fields, but I've kept a maximum of two. There are 41 plots in total: 18 fields with 2 plots and 5 with only 1.

Following the same process as before to create the design:

Suppose we have six varieties to test. Five of these can be replicated seven times and one six times.

```
farm.var<-factor(c(rep(1:5,7),rep(6,6)))
farm.expt<-design(treatments=farm.var,blocks=farm.blocks,searches=1000)
farm.expt$Design
```

```
##    farm fields treatments
## 1     1      1          3
## 2     1      1          2
## 3     1      2          4
## 4    10      1          1
## 5    10      1          4
## 6    10      2          2
## 7    10      2          5
## 8     2      1          3
## 9     2      1          4
## 10    2      2          1
## 11    2      2          5
## 12    2      3          2
## 13    2      3          6
## 14    3      1          5
## 15    3      1          4
## 16    3      2          3
## 17    4      1          6
## 18    4      1          2
## 19    4      2          3
## 20    4      2          5
```

```
## 21    5      1          5
## 22    5      1          3
## 23    5      2          1
## 24    5      2          6
## 25    6      1          1
## 26    6      1          3
## 27    6      2          2
## 28    7      1          4
## 29    7      1          6
## 30    7      2          1
## 31    7      2          3
## 32    8      1          1
## 33    8      1          2
## 34    8      2          6
## 35    8      2          4
## 36    8      3          5
## 37    9      1          6
## 38    9      1          5
## 39    9      2          4
## 40    9      2          2
## 41    9      3          1
```

If you want to have a go at this yourself, there is a csv file called "farm_trials_design.csv" in your data folder. Or you can inspect the output below and take our word for it.

Let's take a look at the results:

```
table(farm.expt$Design$treatments,farm.expt$Design$farm)
```

```
##
##     1 10 2 3 4 5 6 7 8 9
##   1 0  1 1 0 0 1 1 1 1 1
##   2 1  1 1 0 1 0 1 0 1 1
##   3 1  0 1 1 1 1 1 1 0 0
##   4 1  1 1 1 0 0 0 1 1 1
##   5 0  1 1 1 1 1 0 0 1 1
##   6 0  0 1 0 1 1 0 1 1 1
```

```
crossprod(table(farm.expt$Design$farm,farm.expt$Design$treatments))
```

```
##
##     1 2 3 4 5 6
##   1 7 5 4 5 5 5
##   2 5 7 4 5 5 4
##   3 4 4 7 4 4 4
##   4 5 5 4 7 5 4
##   5 5 5 4 5 7 5
##   6 5 4 4 4 5 6
```

The allocation of varieties across farms looks reasonable. What about the allocation within fields?

```
crossprod(table(interaction(farm.expt$Design$farm,farm.expt$Design$fields),farm.expt$Design$treatments))
```

```
##
##     1 2 3 4 5 6
##   1 7 1 2 1 1 1
##   2 1 7 1 1 1 2
##   3 2 1 7 1 2 0
##   4 1 1 1 7 1 2
##   5 1 1 2 1 7 1
##   6 1 2 0 2 1 6
```

9 What do you think? I increased searches to 1,000 because one of my test runs did not do so well. It is quite likely that plot sizes will vary from farm to farm. What effect would this have? Will it invalidate the design?

4.4 Alpha designs.

4.4.1 Creation of augmented and p-rep designs from 2-replicate alpha designs

Alpha designs are forms of incomplete block designs which have proven popular since their introduction in the late 1970's. They remain the standard design for NL and RL trials in the UK. They are resolvable incomplete block designs. "Resolvable" means

that the incomplete blocks can be grouped into complete replicates. This is not necessary from the statistical point of view - non-resolvable incomplete block designs can be very effective - but does make management and scoring of the trial easier. The incomplete blocks give better control of field effects. This is particularly true of very large experiments, as often used in the early stages of testing in breeding programmes. The computer programs to create alpha designs no longer seem to be freely available. A catalogue of designs for number of varieties up to 100 has been published, and can be accessed from within GenStat (and I think within EDGAR[2]). For larger number of varieties, the packages Gendex[3] or CycDesigN[4] will create the designs, but these are not free, though you may be able to download a test copy. The R package agricolae (de Mendiburu, 2020) will also produce designs. However, designs for only two replicates are easy to create. These also lend themselves to simple creation of partially replicated designs. For the very large numbers of varieties found in breeders' trials, it is unlikely that designs with greater than two replicates would be required.

Patterson and Williams (1976) presented simple methods of generating alpha designs in two, three and four replicates, with some constraints. The method for two replicates is that followed described below. The designs are not necessarily optimal but are good and will allow large numbers of varieties. I used to use these extensively.

Alpha designs are not balanced. This is in contrast to many classical incomplete block designs with numbers of varieties which are a complete square (25, 36 and so on). Lack of balance, in this sense, means that not all possible pairs of varieties occur together within an incomplete block. Alpha designs are described as 0,1 designs if pairs of varieties concur once or never. They are described as 0,1,2 designs if, in addition, some pairs of varieties concur twice. 0,1 designs are preferred since these give more even estimation of differences between varieties.

The procedure below will create a 0,1 design for two replicates for any number of varieties v, for which $v = sk$ where k is the block size (number of entries per block) and s is the number of blocks per replicate. Thus, one could produce a design for 200 varieties in s=20 blocks per replicate, each with k=10 entries. There is an additional constraint: the number of entries per block must be less than or equal to the square root of the number of varieties. These constraints are not generally restrictive for large breeders' trials with block sizes of around 10; a reasonable starting point for large trials. The method below does not create the best possible design: rearrangements of which pairs of varieties concur may make improvements, but it will be pretty close.

The small example below is for 12 varieties in block sizes of 3.

First write down the variety codes for the first replicate as s rows of k entries:

```
1 5 9
2 6 10
3 7 11
```

[2] http://www.edgarweb.org.uk/

[3] http://designcomputing.net/gendex/alpha/

[4] https://www.vsni.co.uk/software/cycdesign/

```
4 8 12
```

Thus, the third block in the first replicate contains entries 3, 7 and 11.

Alpha designs are cyclical designs. The meaning of the term will be apparent in the creation of the second replicate. Imagine the columns in the first replicate are numbers on a loop. We leave the first column alone but nudge the next column by one position, the third column by two positions, the fourth (if we had one) by three and so on. This gives:

```
1 6 11
2 7 12
3 8 9
4 5 10
```

You can check that over the eight blocks in the two replicates varieties occur together once (e.g. 1 and 6) or never (e.g. 6 and 7).

You must randomise the order of the blocks within replicates, then the order of the entries within blocks. This is then a perfectly reasonable incomplete block design. You must also allocate the varieties to the code numbers used in the design at random.

There are some restraints that can be put on this randomisation process however, which can be useful. Note that varieties occurring within the same column never occur in the same block. If we have both control and candidate varieties but our greatest interest is in the candidates, then if we allocate the controls (up to four varieties in the example) to code numbers 1 to 4 (at random) then they never concur. Differences between the controls are then measured with less precision, but consequently the precision of comparisons between controls and candidates is increased.

Equally, suppose we had only one control, but we wished this to have greater replication, then this could be allocated to up to four code numbers (numbers 1 to 4) to give up to eight replicates. Or we could create a design with two controls in four replicates by allocating each control to two of the codes numbers 1 to 4.

Now suppose we have some candidate varieties for which we have seed for only a single replicate. For two such candidates, we allocate both to the same code number, but one goes into replicate 1 and the second into replicate 2. This acts to reduce the variation in comparisons among all the candidates. For successive pairs of un-replicated candidates, we should continue with this allocation procedure by working down the columns of the design matrix. Thus, six un-replicated candidates could be allocated to entries 5, 6 and 7. In an extreme case, we might have a single control in eight replicates and 16 un-replicated candidates. From the design above, this would give:

```
1 5a 9a
1 6a 10a
1 7a 11a
```

```
1 8a 12a

1 6b 11b
1 7b 12b
1 8b 9b
1 5b 10b
```

Here, code 1 is the control, 5a and 5b are completely different candidates, as are 6a and 6b etc.

This design should be randomised: first blocks within replicates then position within blocks. An example randomisation gave me:

```
10a 1  6a
1  11a 7a
12a 1  8a
1  5a  9a

7b 12b 1
5b 10b 1
1  11b 6b
9b  1  8b
```

For this example, we didn't need to go through the alpha-design procedure: the only alternative is to arrange the controls systematically. Note we should allocate our candidates to the codes at random: they are almost certainly grouped in some sort of order, by cross or by breeder perhaps, and we should disrupt this patterns to avoid risk of bias.

With larger numbers of candidates and larger block sizes, other arrangements are possible. For example, with block sizes of 12 (requiring a minimum of 144 entries for a 0,1 design), then there will be 12 columns in the design matrix. All entries in the first column could be allocated to one control variety and all entries in the next column to another. This would ensure that each block contained each control once. The candidates could then be allocated to the remaining codes in one or two replicates as described above. A similar end point could be obtained by designing an alpha design with a block size of 10 and then adding the two controls to each block independently of the design. Some caution is required. With no control varieties, if only a small number of candidates are present in two replicates, then the incomplete block analysis may fail: insufficient replicated entries may concur to allow estimation of all block effects. In such a case, analysis as a randomised complete block or by spatial analysis would still be possible.

Regardless of the number of candidates and controls, the candidates must be allocated at random to the candidate codes and the controls at random to the control codes. Then blocks must be randomised within replicates and plot positions within blocks.

4.4.2 Final message

Four important reminders:

1. Always save your design and back it up. Keep notes: don't assume you will remember how you created the design months later, when you have some data to analyse.

2. `blocksdesign` and other software, will carry out the randomisation for you. However, most software does not allocate varieties to the code numbers used in the design at random. You should do this yourself if there is any systematic pattern to the list of varieties: for example varieties are listed by parentage. This could, for example, result in the difference between varieties with the same parents being estimated with greater parentage than those with different parents, which may not be what you want. If in doubt **randomise**.

3. It is perfectly acceptable to use the same design for multiple experiments, but not the same randomisation. Without additional randomisation, you may end up, again, with the difference between some pairs of varieties being estimated more precisely than others. Moreover, the analysis will not take account of this: it will happen without you knowing.

4. Please: always randomise the first replicate. I don't get around much anymore, but many breeders and official testing authorities used to lay out the first replicate in variety order. This makes it easy of visitors and breeders to inspect varieties, but invalidates the design for the same reasons given in (2) and (3) above. Pity poor variety number one. It will always be in one corner of the first replicate; nearest the gate, and more likely to get trampled on and run-over than the others. It was probably drilled first and will be harvested first; when human and machine errors are more likely. If you want demonstration plots, plant or sow them in a separate block.

Chapter 5

Trial analysis (sugar beet example).

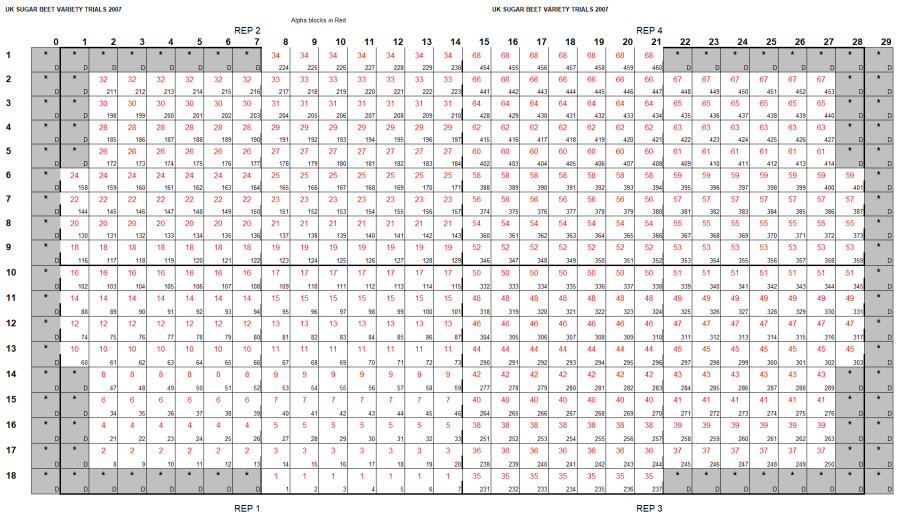
5.1 The layout of the sugar beet trial

These data are from an NL/RL sugar beet trial grown in 2007, designed as an alpha design. We shall analyse this in several different ways, using it to introduce REML as a routine algorithm for the analysis of this sort of data and also to look at the improvements over randomised complete blocks that incomplete blocks and spatial analysis can bring. I've scrambled the names of varieties to avoid anyone's embarrassment. This is a variable site, chosen to illustrate the advantages of more sophisticated trial design.

The design of the trial is as follows:

```
DESIGN TYPE    INCOMPLETE BLOCK
PLOTS          460
REPLICATES     4
BLOCKS/REP     17
VARIETIES      115
PLOTS/VARIETY  4
```

A typical sugar beet plot is in three rows wide, 0.5m apart and is 12m long, with 10m harvested – the two discarded metres are the alleyways between the plots used by tractors for spraying. I make this trial to be 28 plots = 42m wide and 18 plots = 216m long.



For a better impression of the above image: there is a file called "sugar_beet_trial R.xls" in the data folder for this chapter. Inspect the field plan book.

The incomplete blocks run along the rows of the spreadsheet. Note that to accommodate the variety number and the shape of the field, the block sizes are unequal and that there are discard plots within the trial area.

1 Can anyone come up with a better design to fit this area? What about the shape and size of the replicates? Numbers of entries per block? Do you regard the discard plots as a waste of space, or as a saving in cost?

5.2 Fixed effects, random effects and REML

Fixed effects are generally the things you are interested in such as variety performances, fertilizer treatments etc. Random effects are often things that you are not interested in that get in the way of getting good estimates of the fixed effects such as fertility effects; and therefore also include factors in the design to control for fertility effects such as replicates and incomplete. Residual errors from any analysis are always random effects. Over multiple trials, the sites and year effects can be viewed as random. There is ambiguity: sometimes varieties can be viewed as a random sample from a population of possible varieties. The most pure examples of this would be a set of RIL's from an F2 or clones from a cross between two outbred parents. All data are analysed by fitting a model; even a t-test fits a model. A dataset in which some effects are fixed and some effects are random is called a mixed model. REML, residual (or restricted) estimation by maximum likelihood, is an algorithm for analysing mixed models. Variety trials, either individually or in sets, are now routinely analysed by mixed models. This is because some aspect of them - blocks, reps, sites, years and

interactions of these with varieties - are often treated as random effects. Increasingly, varieties are treated as random too. There are advantages and disadvantages to this. The decision depends on the objectives of the experiment.

5.2.1 Preparing the data

The data are saved in the Chapter 5 data folder sorted as columns within rows. The CSV file is called: *"sugar_beet_data.csv"*

2 You should now have picked up on how to read a csv file into R and save it as an object. Do this now for the "sugar_beet_data.csv" and save it as: `beet.data`.

The discard plots have been left out of the file as they had missing data. However, if they had been measured for each trait, it would have improved our fertility correction. Note there are some missing data included as NA: measurements which failed.

We'll work with four of the traits which were recorded:

1. Number of roots measured before harvest – can have an obvious effect on yield in a row crop like beet. Closely correlated with emergence too.
2. Sugar content – the proportion of the fresh weight of a root which is sugar.
3. Sugar yield.
4. Total impurities – a measure of the chemical crud in the beet which affects the efficiency with which the factory can extract the sugar from the root.

I've selected these because they are the most important characters and can also show different patterns of variability within a trial and among varieties.

There is no need to analyse all the characters. This year well concentrate on *sugar content*.

Let's take a look at our column headers in the data object `beet.data` that you've created above:

```
colnames(beet.data)
```

```
##  [1] "rep"           "column"        "row"           "plot"
##  [5] "block"         "variety"       "roots"         "sugar.content"
##  [9] "yield"         "impurities"
```

If you don't get the same result on your PC, you may have left out the required: `headers=T` argument in the `read.csv` command. The first 6 columns are experimental design information and therefore need to be treated as factors. We can inspect how R is currently treating them using:

```
str(beet.data)
```

```
## 'data.frame':    460 obs. of  10 variables:
##  $ rep          : int  1 2 3 4 1 2 3 4 1 2 ...
##  $ column       : int  9 4 17 21 14 4 15 19 9 1 ...
##  $ row          : int  14 4 14 8 10 3 15 7 18 9 ...
##  $ plot         : int  54 187 279 366 115 200 264 378 2 116 ...
##  $ block        : int  9 28 42 54 17 30 40 56 1 18 ...
##  $ variety      : int  469 469 469 469 468 468 468 468 467 467 ...
##  $ roots        : int  160 153 157 156 151 160 157 159 154 145 ...
##  $ sugar.content: num  19.1 18.9 19.4 19.2 18.1 ...
##  $ yield        : num  12.6 10.6 12.3 12.4 14.6 ...
##  $ impurities   : num  2.39 2.38 2.37 2.37 2.29 ...
```

That's informative. The trait data (roots, sugar.content, yield and impurities) are treated as either numeric or integers. That's how we want R to treat the phenotypes. However, the experimental design columns (first 6) are treated as integers. We need to convert these to factors. Use the command: `factor()`.

You can do this for each individual column, referring to it by name [eg `factor(beet.data$rep)`] or by index [e.g. `factor(beet.data[,1])`]:

Or put it into a loop if you wish to speed things up:

```
for (i in 1:6){
   beet.data[,i]<-factor(beet.data[,i])
}
```

If you want help understanding how the loop works you can ask a demonstrator or refer to Chapter 3.

We haven't attached the data before carrying out these conversions. We could do so and carry out the conversions to factors on the attached dataset, but we want to work on a subset of the data and it is easy to be confused about what version of the data we are working on.

As we saw in the R introduction chapter, some analysis in R requires NA to be removed in the response variable. We could do that using:

```
beet.for.anal<-subset(beet.data,!is.na(sugar.content))
```

However, I don't think we need to worry about that here; the functions we are using have in built handling of NA.

There are some points to consider before we start:

- *Watch out for negative variance components* – if you get them, drop that term from the analysis. This is true for all mixed model analyses - not just for variety trials.
- *"Informative missingness"*. Usually we should check that missing data are not associated with a particular variety. Not so much in variety trials, but in case-control analysis of disease risk, this can result in major problems.

5.2.2 Randomised complete block design

You should get the same result as from an ANOVA, as with REML. Try both.

Note that there are some missing data, so the values in the analysis of variance table will vary slightly depending on the order in which terms are fitted. You should fit `varieties` last.

Using `lm`, the standard method for fitting linear models in R:

```
RCB<-lm(sugar.content~rep+variety , data=beet.data)
anova(RCB)
```

```
## Analysis of Variance Table
##
## Response: sugar.content
##            Df Sum Sq Mean Sq F value    Pr(>F)
## rep         3  4.111  1.3702  36.837 < 2.2e-16 ***
## variety   114 62.928  0.5520  14.840 < 2.2e-16 ***
## Residuals 332 12.349  0.0372
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Want to stop writing `data = beet.data`? You could use `attach()`. However, I find it can lead to downstream issue when you start subsetting data.

We could have carried this out in one step:

```
anova(lm(sugar.content~rep+variety,data=beet.data))
```

It makes more sense to save the output first as there is other information we want to extract. In particular we would like to see the line means.

Run this on your computer: `summary(RCB)`

As you will see, this gives more information that we require, but it is worth knowing the command is there.

```
RCB$coefficients
```

gives the "coefficients" without the other information, Note, however, that we are missing an entry for rep 1. You may also observe that we are missing an entry for variety 1.

R, rather than providing simple means, as you would get if you analysed the data by hand (or e.g. by GenStat) expresses results as a deviation from an "Intercept". The intercept is the mean for rep 1, variety 1. You can adjust for this by adding the "Intercept" to all other values. As it is usually the differences between varieties we are interested in, this doesn't matter, though is not very satisfying. However, there are some packages to do this for you. One example is *emmeans* (Lenth, 2020) which will do this job, and more. Another example is *predictmeans* (Luo et al., 2020) which runs faster in my experience, so we'll use that for this tutorial. You will need to install the package by running:

```
install.packages('predictmeans')
```

Then load it via:

```
library(predictmeans)
```

This package only has several functions but it provides some detailed summaries and diagnostics for a large range of different models. The key function is: `predictmeans`.

```
RCB.vars<-predictmeans(RCB,"variety",plot=F)
```

That's saved a list object called `RCB.vars` which contains the functions output. There's a lot of information there, but for now we are just after the line means in a sensible format, which we can access using:

```
RCB.vars$`Predicted Means` [1:5]
```

```
## variety
##       1      55     105     150     155
## 17.9850 18.6025 17.7475 18.3750 18.2625
```

Note I've added `[1:5]` to save paper. Print on your screens without `[1:5]` included. If we access another part of the function output, we can print the first ten standard error of the means:

```
RCB.vars$`Standard Error of Means` [1:10]
```

```
## variety
##       1      55     105     150     155     157     166     189     194     204
## 0.09643 0.09643 0.09643 0.09643 0.09643 0.09643 0.11148 0.09643 0.13668 0.09643
```

3 Note that the standard errors are not all identical. Why is this?

GenStat and ASReml (and other packages?) also provide standard errors of differences between variety means, and this is often the information used in comparing different statistical analysis methods of analysis.

4 Why is the standard error of differences between mean preferred?

The function has already calculated this for us, we just need to access it from our saved object:

```
RCB.vars$`Standard Error of Differences`
```

```
##   Max.SED   Min.SED  Aveg.SED
## 0.1765377 0.1363749 0.1384401
```

5.2.3 Randomised compelete block design - with mixed models

Next, we are going to use the package 'lme4' (Bates et al., 2020). Using the techniques you've mastered, install and load this package to your R environment. This is typically the package that R users would use to fit and analyze linear mixed models.

```
RCB.lmer<-lmer(sugar.content~variety+(1|rep), data=beet.data)
```

Print the output of `RCB.lmer` to your own computers and inspect the output. The `(1|rep)` structure fits replicates as a random effect.

5: It is impossible to do this using `lm`. Here it should make little difference. Why is this?

There are a number of useful commands that we can use to inspect our fitted model:

`anova(RCB.lmer)` will give you a test for significance of the fixed effects in the model (here varieties). These will be very similar to those from `lm`. The tiny difference arises from the treatment of missing data. (With replicates treated as random, there is some information about varieties in between replicates comparisons.)

`ranef(RCB.lmer)` will give you the random effects for replicates.

`fixef(RCB.lmer)` will give you the fixed effects for varieties. Note that variety 1 is missing again. Its effect, as a deviation from the intercept is zero).

`summary(RCB.lmer)` gives a lot of output, mainly consisting of correlations among the estimates of the fixed effects. At the top are the estimates of variance components and a list of fixed effects and their standard errors. Note: the standard deviation listed alongside the variance component is just the square root of the variance component. It is not a measure of precision.

As before, the fixed effects are all expressed as deviations from variety 1. An easy way to see the fitted means is to alter the way the model parameters are fitted:

```
test<-lmer(sugar.content~variety+(1|rep)-1,  data=beet.data)
```

Beware – this alters the result from the anova, since this now includes one extra df testing for the difference of the mean from zero.

`summary(test)` will now give all variety means and their standard errors. Again, there are no standard errors of differences. Alternative summaries of the efficiency of the analysis are to look the F ratio or p-value for the variety effects, or use Akaike's Information Criterion.

As before, we can again use `predictmeans` to extract our fitted means and standard errors.

```
RCB.lmer.vars<-predictmeans(RCB,"variety",plot=F)
```

You can then inspect components of the saved list `RCB.lmer.vars`, such as `Predicted Means` and `Standard Error of Means`. Results are similar to those we had before. To get the average standard error of variety differences, if desired, we can edit and re run the previous commands:

```
RCB.lmer.vars$`Standard Error of Differences`
```

```
##   Max.SED   Min.SED  Aveg.SED
## 0.1765377 0.1363749 0.1384401
```

We can check on the fit of the model. Residuals and fitted values are available as: `resid(RCB.lmer)` and `fitted(RCB.lmer)`. We should plot several elements of the model to inspect the fit and check the assumptions. Such as the distribution of the residuals, residuals against fitted values and observed against fitted values. Thankfully, `predictmeans` has a function that takes care of this for us:

```
residplot(RCB.lmer)
```

All these plots look acceptable to me.

5.2.4 Plotting residuals against field structure.

It is always worth checking the pattern of the residuals in the field layout. There is a method for doing this in R which can be quite resourceful. It requires row and column to be set to numeric. If you remember we changed these to factors at the start of the tutorial, we can check the current type (or mode) of a column using `class`:

```
class(beet.data$column)
```

```
## [1] "factor"
```

```
class(beet.data$row)
```

```
## [1] "factor"
```

Let's keep these columns as factors but save new columns with row and column as numeric, this is easy in R:

```
beet.data$c <- as.numeric(beet.data$column)
beet.data$r <- as.numeric(beet.data$row)
```

Something that is not as simple is our NA pattern in the data. We need to extract the residuals from our model and then add a column to our dataset, so the plotting function can use our layout and residuals at the same time. This is problematic because we have NAs in our phenotype, which were ignored in the model. This means we have less residuals than we have plots:

```
length(resid(RCB.lmer))
```

```
## [1] 450
```

```
length(beet.data$plot)
```

```
## [1] 460
```

There's a 10 plot difference, which is explained by the plots with NA for sugar.content:

```
sum(is.na(beet.data$sugar.content))
```

```
## [1] 10
```

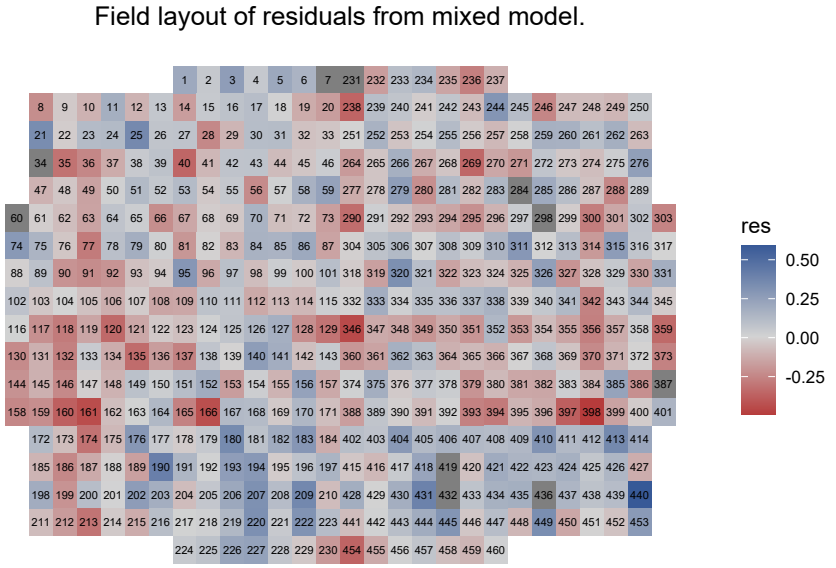
This is a classic R problem, as it stands R won't let us merge columns of different length (which is a good thing). We could have taken the NA plots out of the data beforehand - but this can mean multiple data sets if we have multiple traits. Here's a neater solution:

```
#identify which positions contain none NA phenotypes and save as vector called sel
sel<-which(!is.na(beet.data$sugar.content))
#create column with all entries marked as NA in main data
beet.data$res<-NA
#extract residuals from our model and save in the positions of none NA data points
beet.data$res[sel] <- resid(RCB.lmer)
```

I've used the # symbol to break down what each step does. In short - we now have residuals + those 10 NA plots in the correct place in our data and we can plot the residuals against the field layout. You will need to download and load the R package: desplot (Wright, 2020).

The command we then run is `desplot`:

```
desplot(beet.data, res ~ c + r, text=plot, cex=0.5,gg=T,
        main="Field layout of residuals from mixed model.")
```



That's worked well. The command has several arguments (see `help(desplot)`), the ones we've used were:

- `beet.data` simply the name of our data
- `res ~ c + r` a formula that tells the function to plot residuals against columns + rows
- `text=plot` this labels each plot, which is very helpful (note the image is flipped compared to our first inspection of the trial layout)
- `gg=T` plots via `ggplot2` which speeds up the plotting function

You can see patches of high and low residuals. The residuals can show patterns of field fertility that are not controlled by the experimental design. You can see here such patches of high and low residuals. This is not too surprising as this is a large trial and the only control of field fertility effects in the analysis is through replicates. As these

are large, variation within replicates is not so well controlled. That is why we use incomplete block designs.

You can use this plotting method to plot out other elements of the data against the field structure:

1. We could have plotted our actual raw trait values and inspect obvious issues in the data.
2. We could use it plot the NA structure of the data (NA plots appear to be marked as gray). This would help us look for meaningful NA patterns.
3. We could use it to visualise the field structure and trial layout, you can add blocks and reps ect to the image and create figures for publications or presentations.
   The package for spatial analysis we shall use later will produce similar plots within R.

5.2.5 Alpha design (include block structure)

We shall now fit the alpha design block structure. We can compare standard errors and look at the fertility patterns again. Will the alpha design offer any improvements?

To include blocks, fit as random effects using `lmer`. Add blocks to the previous run of `lmer` as `(1|block)`.

In the experimental design, each block has a unique code. What would happen if blocks were coded 1..17 in each replicate rather than 1..68 over the four replicates? The coding I have used is safer. If the coding is 1..17 in each replicate then you must fit blocks as:`(1|rep/block)`. The `/` nests the term following it within the term preceding it, so that in this case 4 x 17 block effects are correctly fitted.

Here's how I ran the analysis:

```
alpha <-lmer(sugar.content~variety+(1|rep)+(1|block), data= beet.data)
anova(alpha)
```

```
## Analysis of Variance Table
##         npar Sum Sq Mean Sq F value
## variety  114 56.735 0.49767  18.851
```

```
alpha.lmer.vars<-predictmeans(alpha ,"variety",plot=F)
alpha.lmer.vars$`Standard Error of Differences`
```

```
##   Max.SED   Min.SED  Aveg.SED
## 0.1599940 0.1206777 0.1245988
```

This gives me an average standard error of a variety difference of 0.1246, a reduction of about 10% compared to the RCB design. The average variance of a difference from the alpha design divided by that from the RCB design is termed efficiency and is 123%.

5.2.6 Rows and column analysis

As a final refinement, model the random effects as

```
(1|rep/row)+(1|rep:column)
```

What block effects are we fitting. Do these make sense?

If you compare the variance components using the two different methods, in this case you will find they are very similar. In this case, I find the average standard error of the variety means is 0.1087824. Did you get the same?

As an alternative, to compare the different analyses, and in the interests of speed, we can tabulate the F statistic from the analysis of variance:

```
anova(RCB.lmer)
```

```
## Analysis of Variance Table
##         npar Sum Sq Mean Sq F value
## variety  114 62.935 0.55206  14.842
```

For the other designs I get:

- RCB = 14.842
- Alpha design = 18.851
- Row and column design 24.442

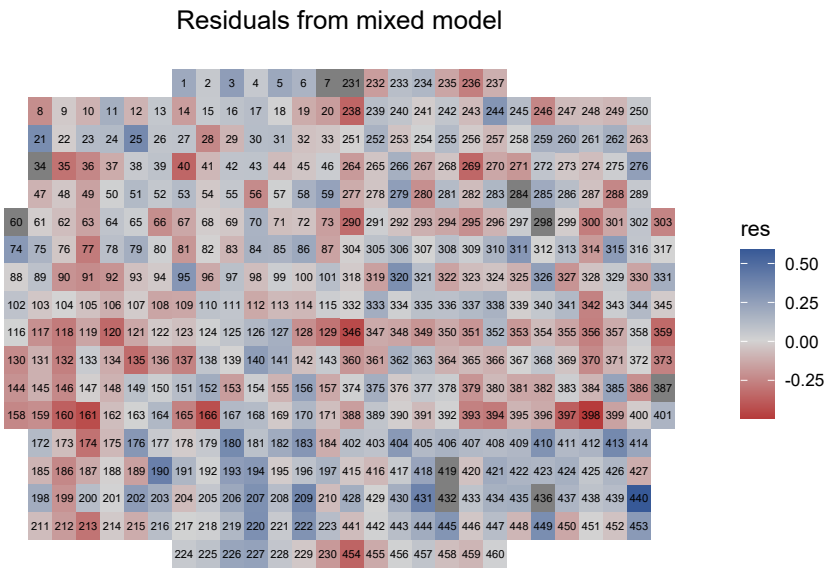
6: Can you think of any additional models? What do the results say about the original alpha blocking for this trial?

The process of changing the trial design after the analysis is called post-blocking. For studies of alternative designs: to suggest improvements to the blocking structures used in the future, it is acceptable. However, it should not be used routinely to estimate variety means as the standard errors, if not the effects themselves, will be biased by continually searching to find the better fitting model. You should stick to the analysis appropriate to the experimental design.

5.3 Spatial analysis

Field fertility effects are not typically refined to the blocks, reps, rows and columns of your trial design and you may need a more sophisticated method to capture a spatial effect in the field. We saw earlier in the plot of our residuals in the rows and columns of the trial layout that there were clear non random high and low areas in our field associated with the phenotype:

```
desplot(beet.data, res ~ c + r, text=plot, cex=0.5,gg=T, main="Residuals from mixed model")
```



In that model we just included replicate and clearly missed patterns in our data. If we were to plot the residuals after fitting block, rows and column, hopefully we would account for what was missed. However, in some cases the trial design components still don't adjust appropriately for the noise in the data and we might be able to find a better model by running a spatial analysis.

Autoregressive models of order 1 (AR1: the standard methods of spatial analysis of yield trials) are available in GenStat or ASReml and guides to their use in these packages are available. ASReml is available in an R version, so it is easy to start using it if you know R. Unfortunately, there is no free version. It may be possible to fit spatial models with lmer too, though it is beyond my understanding and I have yet to find anyone who knows how to.

We shall try a slightly different approach using the free R package from Wageningen (WUR); SpATS (Rodríguez-Álvarez et al., 2017). This models fertility effects using a two dimensional spline. A one-dimensional spline is a bendy curve, so a two-dimensional spline is a sheet, bent to fit fertility patterns running over the field. This is conceptually simple, but the statistical modeling is difficult. You will need to install the package to your R and load it:

```
install.packages("SpATS")
```

```
library(SpATS)
```

SpATS requires rows and columns to be present as both numeric and factors. Moreover, they must both be part of the dataframe. If you remember, we have already met these requirements:

```
class(beet.data$row)
```

```
## [1] "factor"
```

```
class(beet.data$r)
```

```
## [1] "numeric"
```

We need to specify the number of knots in our spatial fit. I believe this controls how flexible our bendy sheet is. Here we specify half the number of rows and columns in our design.

```
# Specify the number of segments used in fitting the spline
nrow <- max(beet.data$r)
ncol <- max(beet.data$c)
nseg.row <- nrow/2
nseg.col <- ncol/2
```

Now fit the model using the main function of SpATS, which is conveniently called `SpATS`. Options are spread over several lines for ease of reading:

```
SpATS.out <- SpATS(response = "sugar.content",
    genotype = "variety",
    spatial = ~PSANOVA(c,r, nseg = c(nseg.col, nseg.row)),
    genotype.as.random = FALSE,
    random = ~ row + column,
    data = beet.data,
    control = list(monitoring=0))
```

That looks complicated. So let's break it down step by step:

- `response = "sugar.content"` is our phenotype, for some reason it has be included in speech marks.
- `genotype = "variety"` the name of our genotype factor

- `spatial = ~PSANOVA(c,r, nseg = c(nseg.col, nseg.row))` this is a formula for defining our spatial P-Spline model. `PSANOVA` means you are running a P-spline ANOVA, where the smoothing consists of five components each dependent on a single smoothing parameter (we'll get on to that next).
- `genotype.as.random = FALSE` simply whether we express genotype as random or fixed
- `random = ~ row + column` here we specify random factors of our model in a formula. We could have added block and rep here. We can add fixed factors too in the same fashion.
- `data = beet.data` our dataset.
- `control = list(monitoring=0)` setting monitoring to `0` here prevents a lot of unwanted output being printed during the fitting. We could control other elements of the fitting via the same means.

We can inspect the model with the following:

```
summary(SpATS.out,which = "variance")
```

```
##
## Spatial analysis of trials with splines
##
## Response:                   sugar.content
## Genotypes (as fixed):       variety
## Spatial:                    ~PSANOVA(c, r, nseg = c(nseg.col, nseg.row))
## Random:                     ~row + column
##
##
## Number of observations:        450
```

```
## Number of missing data:        10
## Effective dimension:           156.91
## Deviance:                      -741.432
##
## Variance components:
##                Variance            SD     log10(lambda)
## row           6.944e-03     8.333e-02            0.44462
## column        5.932e-03     7.702e-02            0.51301
## f(c)          3.856e-04     1.964e-02            1.70007
## f(r)          9.192e-03     9.587e-02            0.32282
## f(c):r        3.107e-04     1.763e-02            1.79390
## c:f(r)        2.775e-04     1.666e-02            1.84291
## f(c):f(r)     3.380e-03     5.814e-02            0.75725
##
## Residual      1.933e-02     1.390e-01
```

Here we can examine the variance components from the model, there are 8 in total:

- `row` and `colunm` treated as random effects
- the residual variance

Then the 5 smooth effects of the spatial trend:

- `f(c)` and `f(r)` are main smooth effects
- `f(c):r` and `c:f(r)` are smooth varying coefficient terms
- `f(c):f(r)` is the smooth-by-smooth interaction between the row and column trend

The interpretation of this part can be useful for looking at how much variation has not been accounted for in our model; shown by the residual variance. The column `log10(lambda)` shows the log10 of the proportion of residual variance to each model component. The lower the value, the greater proportion of the model term variance compared to the residual variance.

Let's take a look at the dimensions of the model which are informative of the complexity of the spatial fit.

```
summary(SpATS.out,which = "dimensions")
```

```
##
## Spatial analysis of trials with splines
##
## Response:                  sugar.content
```

```
## Genotypes (as fixed):       variety
## Spatial:                    ~PSANOVA(c, r, nseg = c(nseg.col, nseg.row))
## Random:                     ~row + column
##
##
## Number of observations:        450
## Number of missing data:        10
## Effective dimension:           156.91
## Deviance:                      -741.432
##
## Dimensions:
##               Effective     Model     Nominal     Ratio     Type
## variety           114.0       114         114      1.00        F
## Intercept           1.0         1           1      1.00        F
## row                13.0        18          16      0.81        R
## column             19.6        28          26      0.75        R
## c                   1.0         1           1      1.00        S
## r                   1.0         1           1      1.00        S
## rc                  1.0         1           1      1.00        S
## f(c)                0.5        15          15      0.03        S
## f(r)                0.9        10          10      0.09        S
## f(c):r              0.6        15          15      0.04        S
## c:f(r)              0.2        10          10      0.02        S
## f(c):f(r)           4.2       150         150      0.03        S
##
## Total             156.9       364         360      0.44
## Residual          293.1
## Nobs                450
##
## Type codes: F 'Fixed'    R 'Random'    S 'Smooth/Semiparametric'
```

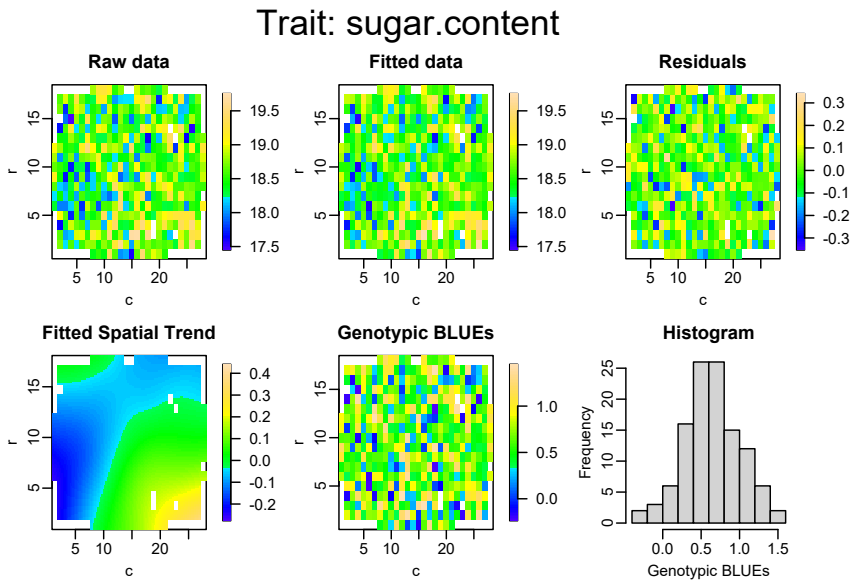
The table is a little complicated. We can see all the components included in the model. We can see what 'Type' each component is, either 'Fixed', 'Random' or a 'Smooth' component. The other key columns show: 'Effective' which can be considered the effective degrees of freedom of that component, 'Model' is the number of parameters of that component,'Nominal' is the same as 'Model' except for random factors which have -1 and 'ratio' is the proportion of effective and nominal dimension.

There are 8 components to the spatial trend, the 5 smooth components discussed above and the linear effects of row and column and their linear interaction.

The effective dimensions (ED) are really informative for the 5 smooth components. The higher the ED of these components (and thus the higher the value in the 'ratio' column) indicates that more of the spatial trend has been captured by that component. It's an indicator that the fitted surface of that component is more complex than the others and has more spatial patterns.

In our trial you can see that out of the smooth components, the smooth-by-smooth interaction between the row and column f(c):f(r) had the highest ED. This is typically informative of a more complicated spatial effect, not limited to rows and columns. We can interpret this for ourselves with the excellent `plot()` function of `SpATS`:

```
plot(SpATS.out)
```



These plots are really useful for a more straight forward interpretation of our spatial model. The 'Raw data' plot is the phenotype plotted out in the row and column structure of our trial. That pattern looks familiar. We saw there was an area of the field associated with lower phenotype values in our residual plot earlier. I would call the worse part of this area between columns 0-10 and rows 4-10. Now if we look at the residual plot on the top right, we should check for any remaining patterns. It looks like our model has been much better at clearing up the residual noise. What is left appears to be mostly random.

We can then compare our residual plot to the Fitted Spatial Trend. This is the easiest way of assessing how strong our spatial trend is (or how well the model has captured it). Compare the axis values of each plot. The spatial trend is of the same magnitude as the residuals. That's typically a sign of a strong spatial effect that has been accounted for by the model. If the fitted spatial trend were a few orders of magnitude lower than the raw data and residual values, we'd know that we had an unidimensional trend.

You can see in the Fitted Spatial Trend plot that there is strong and complex trend across the trial. This explains why the effective dimensions for the more complicated

smoothing component (the smooth-by-smooth interaction) were highest. We can also clearly see the region of the field that had low phenotypic values (in dark blue) and we now know that the spatial effect associated with the trial partially contributed to that issue. We've accounted for that in our model now.

Line means are accessed from our SpATS model using:

```
predict.SpATS(SpATS.out,which= "variety")
```

`predict.SpATS` works to predict fixed and random effect. Standard errors of effects are reported, and these can be averaged, but unfortunately `predictmeans` does not work with the output so we cannot currently get the standard errors of differences. For the time being, the best we can do is compare average standard errors (rather than standard errors of differences).

```
mean(predict.SpATS(SpATS.out,which= "variety")[,7])
```

I get 0.07984206 compared to 0.09437533 for the row and column design, which is an impressive improvement in this case. We would expect that, as the row and column design would not of captured the complexity of spatial trend that we found through SpATs.

SpATs is a relatively new package. Therefore, the literature and guidance surrounding its use and interpretation is limited. There is an associated paper with the package (Rodríguez-Álvarez et al., 2016), it's informative but not easy reading. The paper Velazco et al. (2017) has some really nice examples and it clear to follow. I've used definitions and explanations from both for forming this section.

Chapter 6

Cross sites analysis in R.

We shall analyse the data from the TG association mapping panel to examine methods for analysing data across sites. For this we shall use the `lmer` package and (optionally) the `emmeans` package. Running `emmeans` here can add a lot to the run-time.

It would be better to use ASReml or GenStat, or other commercial packages to analyse these data, but `lmer` in R is free and readily available.

We shall compare variety effects from various types of analyses – BLUE / BLUP / one stage / two stage / weighted / and see what sort of a difference it makes?

6.1 BLUEs, BLUPs and one stage analysis

To start:

1. Load the `lme4` package to your R.
2. We could use `predictmeans` to extract our predicted means. However, for just that purpose, it can take a while to run. Therefore, install and load the package `emmeans` (Lenth, 2020) instead, which runs faster for this method.
3. Change your working directory to the location of today's data files or copy today's data files to your current working directory.
4. Read in the csv file "TG_all_data_for_stage_1.csv" and save as an object called TG

We need to convert our experimental design columns to factors, to do by running the following:

```
for (i in 1:5){
  TG[,i]<-factor(TG[,i])
}
```

Now inspect the data with:

```
summary(TG)
```

```
hist(TG$YLD_ADJ)
```

```
hist(log(TG$YLD_ADJ))
```

```
boxplot(TG$YLD~TG$SITE)
```

Any comments on the data?

The conventional way to analyse these data would be to first analyse each site (there are six here) separately, then take the line means from each site and analyse these in a second analysis fitting terms for varieties and sites with the interaction (varieties x sites) used as error. This approach, with minor modifications is still common. If there are no missing variety-site combinations, the estimates of average variety effects (the BLUEs) are identical to simply taking the average across sites. There is nothing wrong with it, but we ought to be able to do better. It is described as a two-stage analysis.

Our data is of individual plot measurements at all sites. We can mimic simple analysis by:

```
TG.2.stage.mimic<-lm(YLD_ADJ~VAR+SITE+SITE:VAR+SITE:REP,data=TG)
```

1 Why do we have the term SITE:REP rather than just REP? Have a look at the analysis of variance table. Does it make sense? Are the F tests carried out correctly?

We shall now repeat this analysis using `lmer`. Which terms should be fixed and which random?

I fitted the model below. For simplicity and time, I've ignored the row and column structure of the experimental designs. This may take a little time to run.

```
TG.res.1<-lmer(YLD_ADJ~(1|SITE)+(1|SITE:REP)+(1|SITE:VAR)+VAR,data=TG)
```

Inspect the results:

```
summary(TG.res.1)
```

To inspect only the variance components we can use:

`VarCorr(TG.res.1)` on its own prints the standard deviations. Whereas, `print(VarCorr(TG.res.1),comp=c("Variance")` will give you variance:

```
print(VarCorr(TG.res.1),comp=c("Variance"))
```

```
##  Groups   Name        Variance
##  SITE:VAR (Intercept)  28.042
##  SITE:REP (Intercept)  13.391
##  SITE     (Intercept) 379.378
##  Residual              62.049
```

```
anova (TG.res.1)
```

```
## Analysis of Variance Table
##     npar Sum Sq Mean Sq F value
## VAR  385 108474  281.75  4.5407
```

Gives us a test for the significance of the variety term.

The authors of `lme4` have purposely excluded P values from these significance tests. Most people however do like to look at P values. There are several ways to obtain a P value with this test. The easiest is to download another package called `lmerTest` and load it to your R. Then next time when you run `lmer()` followed by `anova()` you will see an estimated P value.

Or you could look up the p-value via multiplying the F statistic by the degrees of freedom and treat the result as a chi-sq test statistic with the same number of df. This is perfectly acceptable for large experiments, as here.

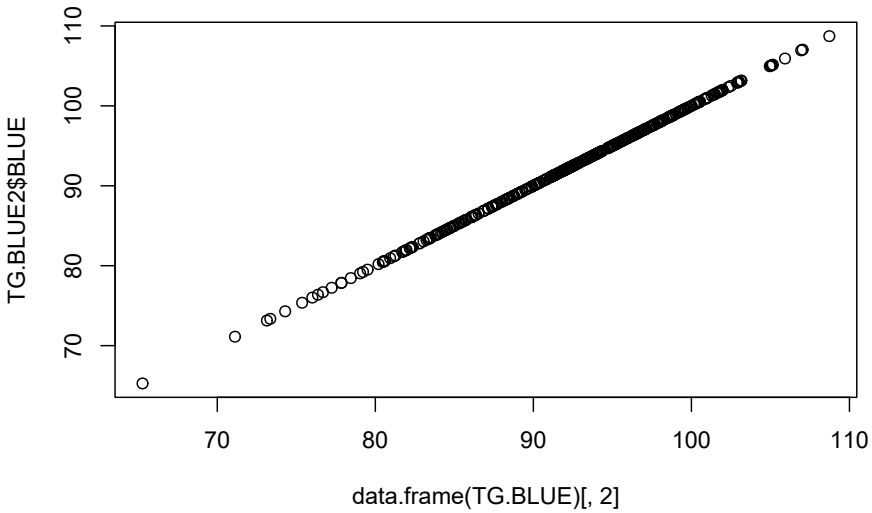
To get out the variety means we use this function from `emmeans` (make sure you have this package loaded).

```
TG.BLUE<-emmeans(TG.res.1, ~ VAR)
```

This gives a warning: you are advised to alter the options to emmeans: `emm_options(pbkrtest.limit = 4688)` but run time is then increased greatly, and in this case it makes no difference to the estimate of the BLUEs, so we shall ignore the warning.

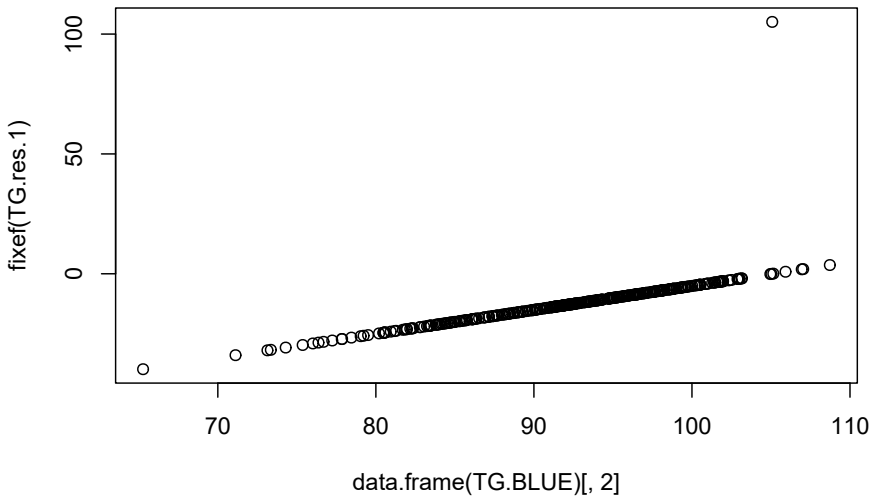
We could have done this using `predictmeans` and not run into these warnings, however, it takes about 5 minutes to run. Let's check that the BLUEs we can get directly from the `lmer` output agree with those from `emmeans`. There's a slightly messy workaround for extracting the BLUEs directly from lme4, remember you have to add the first variety to the rest of the data:

```
#extract fixed effects and add intercept to each
TG.BLUE2<-as.data.frame(fixef(TG.res.1) + fixef(TG.res.1)[1])
#change name of column to BLUE
names(TG.BLUE2)<-("BLUE")
#the intercept (first variety) is x2 what it should be, so fix
TG.BLUE2[1,1]<-TG.BLUE2[1,1]/2
#plot data against emmeans method
plot(data.frame(TG.BLUE)[,2],TG.BLUE2$BLUE)
```



Looks convincing! Both methods produce the same results. In passing we can check that the means from `emmeans` agree with simply getting the fixed effects from lm.

```
plot(data.frame(TG.BLUE)[,2],fixef(TG.res.1))
```



Comments?

We want to save the BLUEs for comparison with others. We can do this using the `write.table` function:

```
write.table(TG.BLUE,"TG.BLUE.txt")
```

This saves a text file to your working directory, open it now and copy and paste it into excel for comparison later.

In principle, we could get the standard errors of differences of comparisons between variety pairs in the same way as for the sugar beet trial we analysed, I would recommend going back to `predictmeans` for this.

This is a one-stage analysis: we have taken the raw plot data and produced an across-sites BLUEs in a single analysis. We made some simplifications:

- We have ignored row and column effects within each site.
- We have assumed a constant error variance at each site. This is the most important omission.
- What have we assumed about replicate effects over the whole analysis?
- We have lumped countries and years together into a single term: sites.

Next, we shall repeat the analysis, but treat the varieties as random effects to estimate BLUPs. Paste the BLUPs into Excel, and record the variance components again

```
TG.res.2<-lmer(YLD_ADJ~(1|SITE)+(1|SITE:REP)+(1|SITE:VAR)+(1|VAR),data=TG)
print(VarCorr(TG.res.2),comp=c("Variance"))
```

```
##  Groups   Name        Variance
##  SITE:VAR (Intercept)  28.066
##  VAR      (Intercept)  35.551
##  SITE:REP (Intercept)  13.395
##  SITE     (Intercept) 379.817
##  Residual              62.025
```

To get just the random effects for variety rather than a long list of all effects:

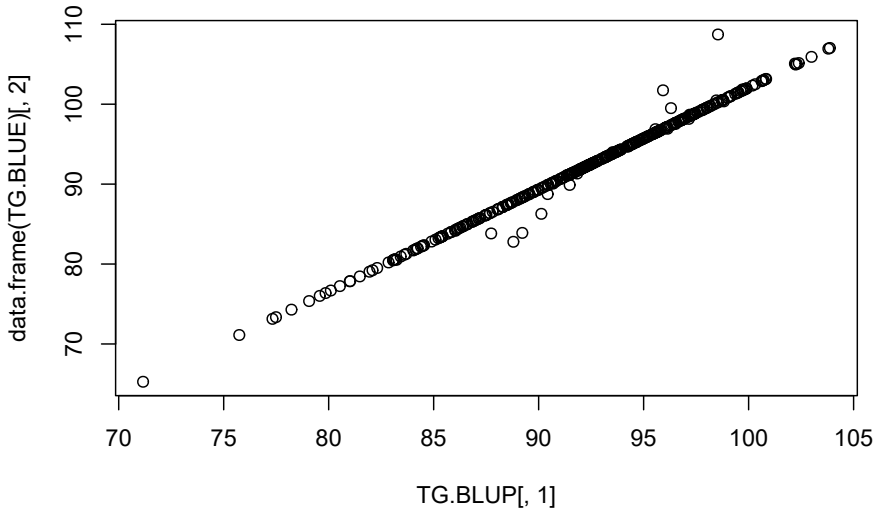
```
TG.BLUP<-ranef(TG.res.2)$VAR+fixef(TG.res.2)
```

How do the BLUPs compare to the BLUEs? Look at the correlations and plot:

```
cor(TG.BLUP[,1],data.frame(TG.BLUE)[,2])
```

```
## [1] 0.9953824
```

```
plot(TG.BLUP[,1],data.frame(TG.BLUE)[,2])
```



To test the significance of random effects in mixed models, we use a likelihood ratio test (LRT). This is a very commonly used method in statistics and quantitative genetics. The difference between 2 x the log-likelihood of two models is approximately distributed as a chi-squared statistic with df (for the test) equal to the difference in df between the two models. This method is not appropriate for testing the significance of fixed effects in mixed models however. To get the log-likelihood of the model you should have just fitted:

```
logLik(TG.res.2)
```

```
## 'log Lik.' -17385.84 (df=6)
```

```
logLik(lmer(YLD_ADJ~(1|SITE)+(1|SITE:REP)+(1|VAR),data=TG))
```

```
## 'log Lik.' -17499.43 (df=5)
```

The likelihoods can be difficult to interpret. Higher values (i.e. less negative values) indicate a better fit to the data. The original mode therefore gave a better fit but used up 1

df to do this. This is always the case: adding parameters gives a better fit (a smaller error variance in linear modeling) but each parameter requires 1 df. In mixed modeling, each variance component has 1df – to estimate the variance.

The LRT is therefore 2 x (-17385.84 - -17499.43) with (6 - 5) df.

`pchisq()`

will return probabilities associated with chi-squared tests. There are two slight complications, Firstly pchisq returns the cumulative probability distribution: the left hand side of the distribution. You require 1 minus this for the significance test. Secondly, variance components are always positive, never negative (though their estimates can be). For this reason, the chi-sq test gives a false impression. We would always dismiss a negative component as non-significant. To adjust, for this asymmetry we halve the probability that you conventionally look up. So a p-value of 0.24 would be treated as 0.12. This isn't as esoteric as you may think: the same correction is often required in linkage analysis. In this case, there was never any doubt that this interaction was going to be significant.

6.1.1 One stage analysis with weights

So far, we have treated each observation as if it was known with equal precision. This isn't true. Some of the trials were better than others. The WT column in the data is a weight for each observation. It is defined as 1/variance of the observation. This is a very common weight to use. As an estimate I have used the error term from an earlier analysis of each trial.

SITE	Ve
CALLOW_2011	72.63
FRANCE_2010	60.81
FRANCE_2011	23.92
LGE_2010	12.64
LGE_2011	25.41
NIAB_2010	14.72

These were estimated from a full analysis which included row and column terms (just as in the sugar beet example) so it isn't a perfect match for the model we are using. It serves to illustrate the method. We simply tell `lmer` to include the weights:

```
TG.res.3<-lmer(YLD_ADJ~(1|SITE)+(1|SITE:REP)+(1|SITE:VAR)+VAR,weights = WT,data=TG)
```

`emmeans` will provide the BLUEs to compare results with the previous unweighted analysis.

2 Compare the standard errors from the two analyses as well as the BLUEs themselves. What do you find?

Finally, we shall export the BLUPs for each site. As an alternative we could make the sites x varieties effect fixed and estimate BLUEs for each site, but in practice this would give us the same result as if we analysed each site separately in a two-stage analysis. There are two possible advantages to using BLUPs. Firstly we automatically get an estimate for every variety, including those with missing data. Secondly, these estimates will be shrunk towards the overall sites BLUE depending on the magnitude of the sites x varieties interaction. This seems reasonable: if there is no interaction then whatever site we want to select for, we should just select on the overall mean yet if the interactions are very large we should select on the estimated effect from the single site analysis. Treating the interaction as random, we are in effect borrowing information from the other sites depending on how high the correlation is between site means. (This is a very similar approach to treating variety means at each site as separate traits and selecting on one trait (i.e. site) while incorporating information based on the correlation with other traits (i.e. sites). We shall return to correlated characters later in the course.)

We just need to export the results from the analysis we have already carried out:

```
ranef(TG.res.1)$"SITE:VAR"
```

This is similar to the command we used to export the variety means, except by trial and error, I found I needed to include the quotation mark to prevent R interpreting ':' as some form of separator.

The output is quite long so this time we shall export to a file. Make sure you know what your working directory is, and change it required.

```
getwd()

write.csv(ranef(TG.res.1)$"SITE:VAR","site.var.csv")
```

`write.csv` is a version of `write.table` and writes csv files (so they can be read directly into Excel).

After this, there is some tedious formatting of the results using vlookup since not all varieties are present in all sites. We require a column for each site, but we can't simply cut and paste because not all varieties are present at all sites. I've done this for you. My results are in 'results TG R.xlsx' in the Chapter 5 data folder downloaded from the website. We want to compare the BLUPs we've estimated for each site with the means we can estimate from the site by site analyses. I have previously carried out the analysis for each site and results are also in the spreadsheet. To compare means we also need to add the average effect for each variety (from emmeans) onto the BLUP. I've also done this. Strictly, we should add on the average site (not variety) effect too, but as we are only interested in the correlation between sites this doesn't matter.

I've done much of the tedious work for you here. It is the results that are important to think about. Correlating my shrunk estimates of variety effects with the individual site analysis I get:

```
CALLOW_2011 0.93
FRANCE_2010 0.94
FRANCE_2011 0.95
LGE_2010    0.87
LGE_2011    0.94
NIAB_2010   0.83
```

What do you think? What has happened to the correlation between sites:

On site specific BLUEs

	CALL11	FR10	FR11	LGE10	LG11	NIAB10
CALLOW_2011	1.00	0.39	0.46	0.23	0.57	0.52
FRANCE_2010		1.00	0.59	0.54	0.47	0.41
FRANCE_2011			1.00	0.51	0.48	0.40
LGE_2010				1.00	0.37	0.32
LGE_2011					1.00	0.53
NIAB_2010						1.00

Average 0.45

On shrunk estimates:

	CALL11	FR10	FR11	LGE10	LG11	NIAB10
CALLOW_2011	1.00	0.74	0.75	0.71	0.80	0.79
FRANCE_2010		1.00	0.83	0.82	0.79	0.79
FRANCE_2011			1.00	0.87	0.79	0.79
LGE_2010				1.00	0.79	0.79
LGE_2011					1.00	0.84
NIAB_2010						1.00

Average 0.79

Discuss!

Compare the results here with the variance components we recorded earlier.

3 You are a grower and must select seed for the coming season. Assume, fortuitously, that one of the trials was located on your farm. Will you select on variety means across all sites, the variety mean on your farm, the shrunk estimate of yield for your farm?

As far as I am aware, the only country in which shrunk estimates are routinely provided to growers is Australia. In the UK, BLUEs across sites and years, and at individual trial sites, are published.

There is a method for getting an estimate of the expected correlation in variety mean between sites (unshrunk) from the variance components. I shall describe it, but we shall return to this later in the course. Here, I estimate 0.38. The minimum and maximum observed values are 0.23 and 0.59 and the mean 0.45, so this estimate seems reasonable.

Most across-environment analyses, including the UK national and recommended list trials for example, do less analysis than we have done so far: they simply treat varieties as fixed, and estimate the BLUEs (though they won't call them that). This will usually be done in a two-stage analysis, and there will be no weighting. Although the overall means vary little in our example, the predictions at each site can vary a lot. In addition, in many official trials, and breeders' trials, the variance associated with varieties can be much smaller than in this genetically very variable GWAS panel. There is usually a much greater lack of balance between varieties over sites and years, so the difference between methods could be greater than we see in this example.

6.2 Weighted two-stage analysis

A comprehensive one-stage analysis, as implemented in Australia, will include the full experimental design at each site, with separate error variances for each site, analyse the data at site level with an AR1 x AR1 spatial model, incorporate information on relationships between varieties, and implement a more sophisticated G x E analysis than we have used here. Computation is in ASReml. That is beyond both the scope of this course and beyond me. Hans-Peter Piepho advocates use of the two-stage analysis, but weighting by the precision of the variety means, or better by the full variance-covariance matrix of variety effects. This has the advantage of making the analysis of individual sites much more flexible. Genetic relationships between varieties can also be incorporated. This too is beyond me. We shall carry out a simplified version:

Approximate weights are 1/(variance a variety mean). These could be obtained from the output of emmeans for each site. Emmeans reports the standard error of the variety mean, so the weight would be $1/se^2$. We could analyse each site in turn here, fitting row and column effects in addition to replicates, or carrying out a spatial analysis. However we shall approximate the weights as r/(error variance) where we shall use the same error variance reported earlier.

Read in the csv file: 'site_means_for_stage_2.csv', you should be able to say what each of the following commands does:

```
#to run the below line, you will need to remove: data/
TG2<-read.csv("data/site_means_for_stage_2.csv",header=T)
```

```
TG2$VAR<-factor(TG2$VAR)
TG2$SITE<-factor(TG2$SITE)
TG2.res.1<-lmer(YLD_ADJ_av~(1|SITE)+VAR,weights = WT,data = TG2)
TG2.BLUE<-emmeans(TG2.res.1,~VAR)
```

4 Export the data and compare with the BLUEs from the previous analyses. Or plot / correlate with previous results we have saved. You should be able to edit the commands above to repeat these analyses with / without weighting and with varieties as a fixed or random term.

6.3 AMMI

The additive main effect, multiplicative interaction model has become the standard method of examining genotype by environment interactions. It has some limitations, however. It cannot handle missing data and it assumes that the G x E terms, varieties x sites in our example, are fixed effects. There are random effects alternatives (more complicated), but we shall stick to the standard model.

We shall illustrate the approach using basic R commands. There are dedicated packages, both using R and standalone, but the basic approach will, I hope, help with understanding what is going on, at the expense of learning a few more R commands (but you can copy from here).

The csv file 'AMMI.csv' has variety means from the individual site analyses for the TG data. I've deleted varieties with missing data. Most of these were only present at a single site so would not contribute to the pattern of G x E. For varieties with only a single missing observation, I could have inserted the predicted effect for that site: mean + variety effect + site effect (i.e. assuming no interaction term at that site.)

I've also added countries of origin – deleting those for which country was unknown. The final data set has 371 varieties with three countries of origin, tested at six sites: two in each country of origin (in different years). We are interested in examining whether, on average, varieties tend to perform best at home, rather than playing away.

Read the data in and save as an object called: `TG.AMMI`

To simplify the analysis and inspection of results, we shall work on the performance of varieties averaged over their origin, rather than on the individual variety performance. As we are working on GxE, we also want deviations from the trial means and from the country of origin means. We are therefore aiming for a matrix of 3 rows and 6 columns. We could do these calculations in Excel, and in truth, like most of you I would have found that faster than working out the code below in R, but I've done it, and can copy it if I need to do a similar exercise again.

Before we produce the required deviations, let's see if any differences between countries of origin appear genuine.

1. How can we test for this? (Answer is on the next page if you want to cheat, but have a think first).
2. Should we treat differences between varieties within countries as fixed or random effects?
3. Carry out the test for each trial site in turn. What do you make of the results?

```
anova(lm(TG.AMMI[,3]~COUNTRY,data=TG.AMMI))
```

```
## Analysis of Variance Table
##
## Response: TG.AMMI[, 3]
##            Df Sum Sq Mean Sq F value        Pr(>F)
## COUNTRY     2   4478 2239.01  18.387 0.00000002449 ***
## Residuals 368  44813  121.77
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It is worth looking at variance components too – use `lmer` to split variance into between countries and within. The countries are a fixed effect rather than a random sample from a population – and they are definitely not exchangeable. However, the between country variance can be used as a simple means to quantify the relative magnitude of differences between and within countries, but we mustn't get carried away in our interpretation: to infer how what we might find if we tested things in Hungary for example.

What is your interpretation of the variance components?

We shall now get a matrix of G x E terms: deviations from row and column averages. There are various ways of getting at this. This is one:

We shall use the command aggregate to group and average the data by country of origin.

```
country.means<-aggregate(TG.AMMI[,-1:-2],list(TG.AMMI$COUNTRY),mean)
country.means
```

```
##   Group.1 CALLOW_2011 FRANCE_2010 FRANCE_2011 LGE_2010 LGE_2011 NIAB_2010
## 1     DEU    98.27816    124.4816    90.46138 86.33333 82.68701  68.45862
## 2     FRA    96.65825    125.2517    93.64142 86.40730 78.70014  67.00697
## 3     GBR   105.71096    125.7438    96.34849 83.18342 84.21452  70.53014
```

I can talk you through this command. Do you understand the `[,-1:-2]` reference? `list` is required to tell R that the term in the brackets is to be treated as a list – one of the R internal structures. Without it, you get an error. This is an annoying quirk of R: it is very fussy about the type of structure it is working on, but often the fix is very simple, as here, yet still takes some trial and error to identify.

```
cm.rm<- rowMeans(country.means[,-1])
cm.cm<- colMeans(country.means[,-1])
cm.mean<-mean(as.matrix(country.means[,-1]))
```

It is reasonably obvious what these commands are doing, together with another example of a requirement to change the data type to get a command to run.

Now that we have the marginal means, we can work out the GxE terms: in effect the error term from a simple linear model. Can you see what this is? Writing the model down may help if it isn't obvious.

The following bit of code - employing two loops, one nested inside the other, works out the deviations.

```
cm.GE<-country.means
 for (i in 2:7) {
   for (j in 1:3){
     cm.GE[j,i]<-
     cm.GE[j,i]+ cm.mean - cm.rm[j]- cm.cm[(i-1)]
     }
 }
```

Let's inspect cm.GE:

```
cm.GE
```

```
##   Group.1 CALLOW_2011 FRANCE_2010 FRANCE_2011  LGE_2010   LGE_2011   NIAB_2010
## 1     DEU   -1.271135  -0.0109323   -2.355892  1.691808  1.4862798  0.45987222
## 2     FRA   -2.385319   1.2648473    1.329880  2.271503 -1.9948595 -0.48605159
## 3     GBR    3.656454  -1.2539150    1.026012 -3.963310  0.5085797  0.02617936
```

This has been fiddly and we want to check that the GxE terms are correct. There is a simple way of doing this. Do you know what it is? Answer is coming up.

```
colMeans(cm.GE[,-1])
```

```
rowMeans(cm.GE[,-1])
```

We now have the GxE terms for country in the data.table `cm.GE`. This is a matrix of 3 rows (countries) x 6 columns (sites).

We have just fitted a simple linear model to a table. We had row effects, column effects, and an overall mean and from these we make an estimate of the true value of each cell in the table. These are often referred to as the fitted values. The deviations from the fitted values (which are our GxE terms in our data) sum to zero over rows and to zero over columns. If we squared these deviations and added them up, that would be the error sum of squares, or residual sum of squares in an analysis of variance.

Rather than fitting a linear model, we can fit a multiplicative model. In this, the row effects and column effects are estimated in a different way, and are often called the row and column eigenvalues, or the principle components. The estimate of the true value of each cell in the table is made by multiplying the row effect and the column effect together. The deviations from the fitted values can be squared and added up and treated as a residual sum of squares as before. However, the row and column means are no longer zero. If we wanted, though it would be daft, we could use the row and column means of these residuals to estimate the residuals themselves. Rather than do this though, we can fit another multiplicative model, with a second set of row vectors and column vectors. The error sum of squares will be smaller than before, and the deviations would still not sum to zero. We could iterate in this manner as many times as we wished, but eventually we would get a perfect fit to the data matrix. In practice, we often find that two or three iterations are all that is required to get a very good fit (a very small error sum of squares). This process of iteration and getting closer and closer to a perfect fit is called Singular Value Decomposition (SVD). It is very very closely related to what we do in Principle Component Analysis or Principle Co-ordinate Analysis.

So what? Well suppose you had a matrix of 20 sites and 1000 varieties – 20 000 entries. It is very difficult to identify or interpret patterns in the data. But two multiplicative iterations of the type described about would reduce the data to 20 x 2 eigenvectors for sites and 1000 x 2 eigenvectors for varieties. We are now trying to interpret 2040 rather than 20,000 data points, which is still quite a task but potentially easier. The interpretation is helped, however, because we can plot the data in something called a biplot. This is just a scattergram of the eigenvectors from the first and second iterations. It is called a biplot not because it is a plot in two dimensions, but because it plots both row (variety) and column (site) effects together, and these often lead to easier interpretation. We'll see this once we produce our own.

The second complication is that the solution to the multiplicative model is not unique. We could multiply every row eigenvalue, by any constant and divide the corresponding column eigenvalue by the same constant and we would get exactly the same prediction. The standard approach is to scale the sum of squares of row eigenvalues to 1 and independently scale the SS of the column eigenvalues to 1. When these two eigenvectors are multiplied together to estimate the values in the table, they then need rescaling. This rescaling factor is called the eigenvalue. That seems complicated, but the eigenvalue has the nice property that the bigger it is, the more variation is accounted for, so it is a measure of importance in its own right.

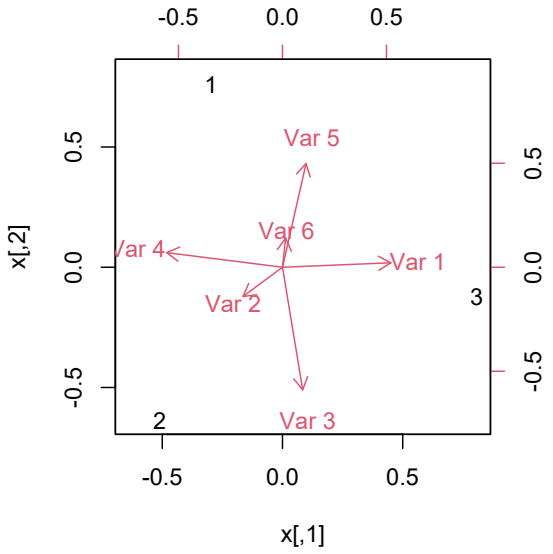
Back to the data. To carry out the SVD:

```
svd.cm.GE<-svd(cm.GE[,-1])
```

Inspect. We can discuss it.

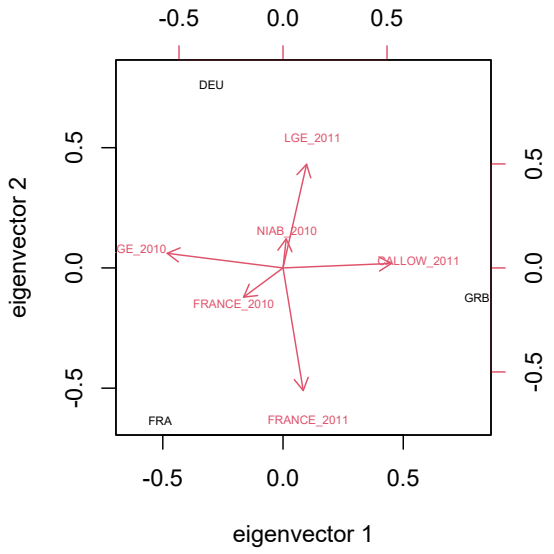
We could produce the biplot, as it's just a scattergram, using the standard plot command, but there is a command biplot which helps interpretation:

```
biplot(svd.cm.GE$u,svd.cm.GE$v)
```



That is a bit ugly though, so we'll add some labels:

```
biplot(svd.cm.GE$u,svd.cm.GE$v,xlabs = c("DEU","FRA","GRB"),ylabs=colnames(cm.GE[-1]),cex=0.5,xlab="eigenvec
```



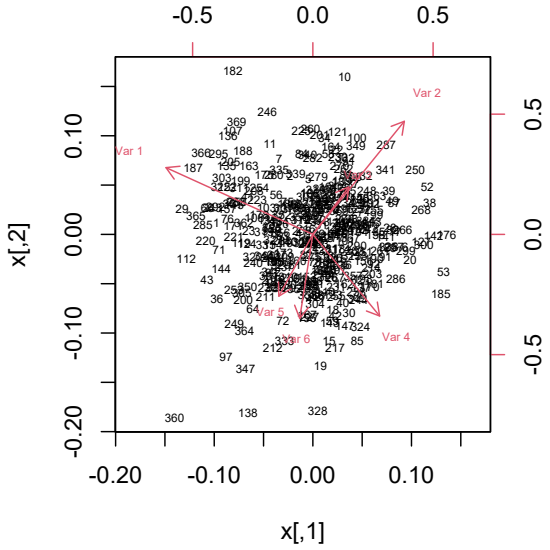
Zoom the plot and we'll discuss it: is there any visible evidence for country specific adaptation? To my mind, this has worked very well. I wish I had done this earlier as I would have included it in the publication.

Finally, with the TG data, we could have produced a biplot on the variety x sites matrix, without first summarising over countries. Following the same commands as above run:

```
TG.AMMI.rowMeans<-rowMeans(TG.AMMI[,3:8])
TG.AMMI.colMeans<-colMeans(TG.AMMI[,3:8])
TG.AMMI.mean<-mean(as.matrix(TG.AMMI[,3:8]))

TG.AMMI.GE<-TG.AMMI
 for (i in 3:8) {
   for (j in 1:371){
     TG.AMMI.GE[j,i]<-
     TG.AMMI.GE[j,i]+ TG.AMMI.mean - TG.AMMI.rowMeans[j]- TG.AMMI.colMeans[(i-2)]
     }
 }

colMeans(TG.AMMI.GE[,3:8])
rowMeans(TG.AMMI.GE[,3:8])
```

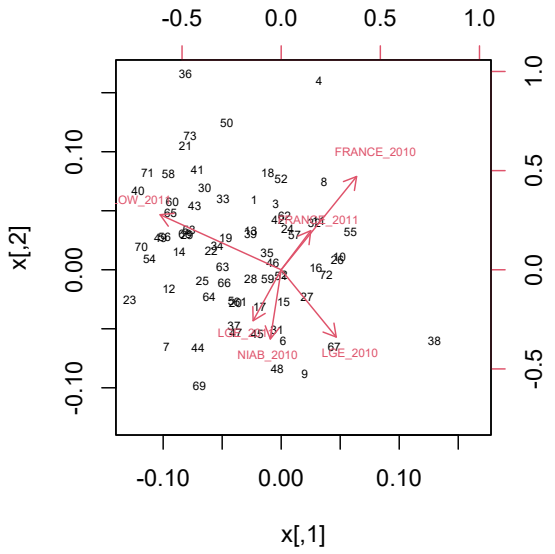
Then we can run svd and plot the results:

```
svd.TG.AMMI.GE<-svd(TG.AMMI.GE[,3:8])
biplot(svd.TG.AMMI.GE$u,svd.TG.AMMI.GE$v,cex=0.5)
```



Can you make sense of that? It is a little easier if we produce plots for each country of origin in turn. For example for the UK:

```
biplot(svd.TG.AMMI.GE$u[TG.AMMI$COUNTRY=="GBR",],svd.TG.AMMI.GE$v,cex=0.5,ylabs=colnames(TG.AMMI[,-1:-2]))
```


-0.5
0.0
0.5
1.0
0.10
0.00
-0.10
x[,2]
x[,1]
FRANCE_2010
FRANCE_2011
NIAB_2010
LGE_2010

Any better?

Chapter 7

Population genetics.

7.1 Hardy-Weinberg equilibrium and population structure

Hardy-Weinberg equilibrium frequencies and population subdivision (Fst)

In the absence of the action of evolutionary forces (selection, mutation, migration, non-random mating, genetic drift), genotypic frequencies in a population follow directly from the individual allele frequencies in the population. This is called "Hardy-Weinberg equilibrium". However, in most populations and in particular in breeding populations, the conditions under which HW hold are not very realistic. Nevertheless, when averaged over multiple markers, the HW test provides a simple means of assessing inbreeding, selfing, or population mixing in genetic samples. In particular, it is useful for detecting population sub-structure within a population.

To illustrate this, we will use a subset of a Teosinte (*Zea mays* ssp. *parviglumis*) data set published by Weber et al. (2007). Teosinte is the wild, outcrossing ancestral species from which maize was domesticated. Details of the entire data set can be found in the paper. Here we will use data of 592 SNP markers from 544 genotypes. The genotypes can be classified in four groups based on the geographical origin (States in Mexico).

In the tables below are the SNP genotype frequencies of two markers are given (for all individuals in the data set).

Use the information in the table to answer the following questions:

1

a: Based on the information in the table, give the allele frequencies, the observed (Ho) and expected (He) heterozygocity, and the fixation index (F) for each marker.

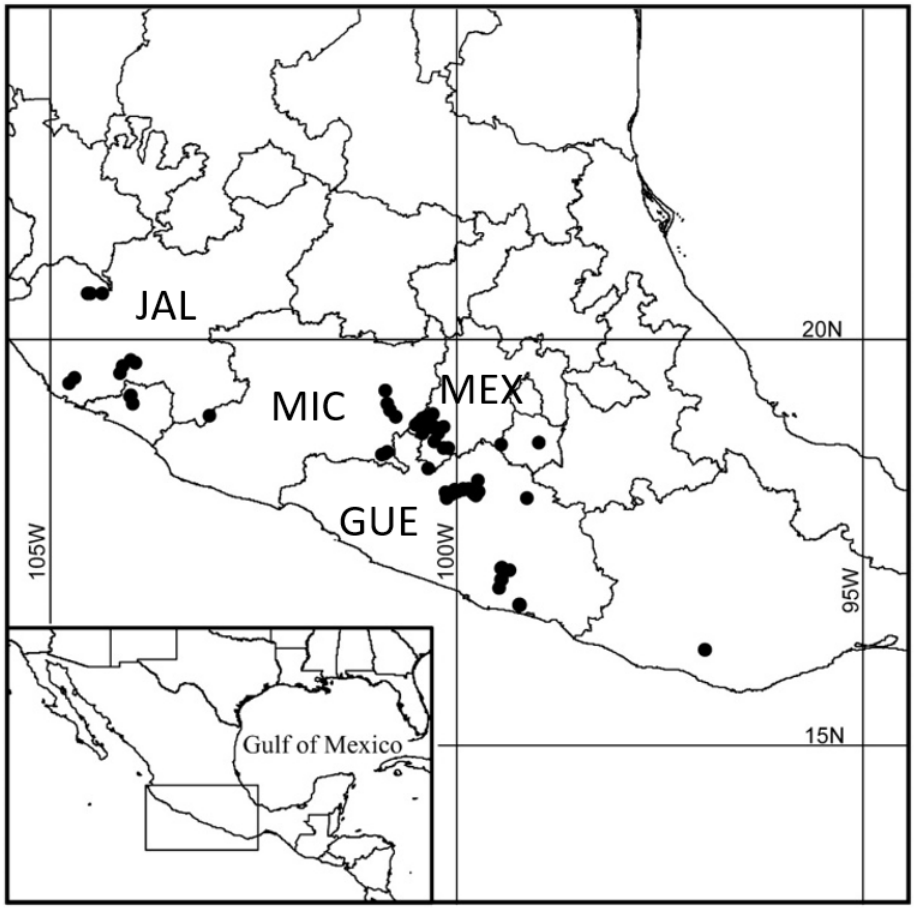


Figure 7.1: Teosinte distribution over the states of Mexico

Table 7.1: SNP genotype frequencies for two markers.

PZA00219_6	N.individuals	PZA03001_18	N_individuals
A/A	163	A/A	190
A/G	158	A/C	207
G/G	213	C/C	145

Table 7.2: SNP genotype frequencies by Mexican state for a single marker

PZA00219_6	Guerrero	Jalisco	Mexico	Michoacan
A/A	30	79	18	36
A/G	61	5	29	63
G/G	134	9	31	39

b: What do the results in (a) suggest in terms of population structure? Explain.

c: Test whether the population is in HW equilibrium based on these two markers. (to do this you will need to do a Chi-square test by hand).

The table below gives the same information for SNP PZA00219_6 but separated by group (defined by the Mexican state in which they were collected).

Use this table to address the following questions:

2

a: Give the Ho, He, and F for each population for marker PZA00219_6.

b: Test whether the groups are in HW equilibrium for this marker.

c: Compare the results with those in the quiz block above. How do you explain the differences?

Your conclusions so far above are based on only one/two loci. We will now look across all loci using a set of three related R packages: `adegenet` (Jombart, 2008) which is a general purpose genetic data analysis package, and the population genetic packages `pegas` (Paradis et al., 2020) and `hierfstat` (Goudet and Jombart, 2020) which use the `adegenet` data format. For a non-R alternative you could also use Arlequin[1]. However, use of Arlequin takes some practice, to get the hang of the input formats.

You will need to install the packages `adegenet`, `pegas` and `hierfstat` to you own R environment.

Throughout this tutorial we start to use longer and more complicated scripts. You can copy and paste these scripts directly from the tutorial book (as we've been doing prevoisly). Alternativly, if you are becoming tired of copying and pasting, we've added the full scripts the Chapter 7 data folder on the course website. See: "teosinte_HardyWeinberg_script.R", "teosinte_pca_script.R", "AG_script.R" and "LD_Script.R", if requried.

[1] `http://cmpg.unibe.ch/software/arlequin3/`

Set your home directory to a folder containing today's data files. For the "genind" data format used in `adegenet`, it is easier to read in the genotype data and population classification of individuals from separate input files. Take a look at the datafiles – "teosinte_genos_R.txt" and "teosinte_pops_R.txt" – to see the format required. Then read in the data files "teosinte_genos_R.txt" and "teosinte_pops_R.txt", these data files are found on the course website.

```
# Load required libraries. adegenet and its associated packages are good for population genetic analysis in
library(adegenet)
library(pegas)
library(hierfstat)
# read in the data for genotypes and population codes separately.
#you will need to delete "data/" if you copy this script
genos<-read.table("data/teosinte_genos_R.txt", header=T)
pops<-read.table("data/teosinte_pops_R.txt", header=T)
```

When loaded, convert to the "genind" format, add in the population information, and check that you have inputted the data correctly:

```
# convert to genind data format
teosinte<- df2genind(genos,ploidy=2,sep="/")
# add in the population information
teosinte@pop<-as.factor(pops$pop)
#check that the data is as expected
teosinte
```

```
## /// GENIND OBJECT /////////
##
##  // 544 individuals; 592 loci; 1,184 alleles; size: 2.8 Mb
##
##  // Basic content
##    @tab:  544 x 1184 matrix of allele counts
##    @loc.n.all: number of alleles per locus (range: 2-2)
##    @loc.fac: locus factor for the 1184 columns of @tab
##    @all.names: list of allele names for each locus
##    @ploidy: ploidy of each individual  (range: 2-2)
##    @type:  codom
##    @call: df2genind(X = genos, sep = "/", ploidy = 2)
##
##  // Optional content
##    @pop: population of each individual (group size range: 80-227)
```

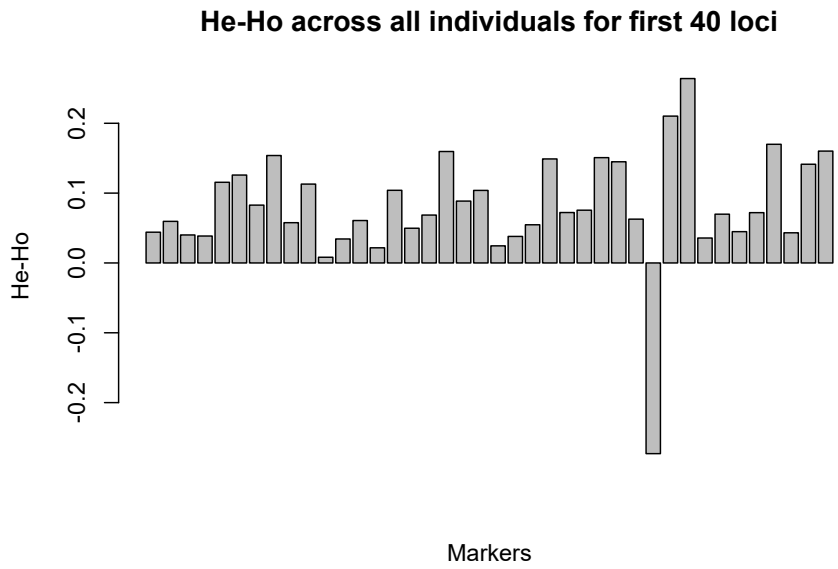
First we make a summary of the data, which includes the observed and expected heterozygosities for every marker, assuming that all individuals belong to a single population.

```
#make a population genetic summary of the data set considering it as a single population
Teo_Summary<-summary(teosinte)
```

Inspect `Teo_Summary` on your own computer.

We can calculate the "fixation index" as you did earlier in the exercise, and also plot Ho-He for a subset of markers to see if there are any patterns. What do you observe?

```
# plot Ho-He for the first 40 markers
barplot(Teo_Summary$Hexp[1:40]-Teo_Summary$Hobs[1:40],xaxt='n',main="He-Ho across all individuals for first 40 loci", yla
```




```
# calculate F values
Fvalues<-1-(Teo_Summary$Hobs/Teo_Summary$Hexp)
```

Now we can test agreement with HW expectations at each locus.

```
#test for Hardy-Weinberg equilibrium at each marker separately
hwt.all<-hw.test(teosinte, B=0)
```

Inspect the first 40 markers on your own computer by running: `hwt.all[1:40,1:3]`

We can count how many of the 592 HW tests we have done are significant at the 5% level. (In R, the `sum` function will count the number of times a condition is `TRUE` when used as in the script here).

```
# count the number of markers that significantly deviate from HW equilibrium with P<0.05
sum(hwt.all[,3]<0.05)
```

```
## [1] 559
```

3 Are more results significant than we might expect by chance?

Next, calculate the average Ho, He and F across all markers:

```
# Calculate average (se) of Ho, He, F across all markers

se <- function(x) sqrt(var(x)/length(x))
Ho_ave<-round(mean(Teo_Summary$Hobs),4)
Ho_se<-round(se(Teo_Summary$Hobs),4)
He_ave<-round(mean(Teo_Summary$Hexp),4)
He_se<-round(se(Teo_Summary$Hexp),4)
F_ave<-round(mean(Fvalues),4)
F_se<-round(se(Fvalues),4)

Average_table<-data.frame(NA,nrow=5,ncol=3)
Average_table[1,1]<-paste(Ho_ave, "(", Ho_se, ")")
Average_table[1,2]<-paste(He_ave, "(", He_se, ")")
Average_table[1,3]<-paste(F_ave, "(", F_se, ")")
colnames(Average_table)<-c("Ho", "He", "F")
rownames(Average_table)[1]<-c("All")
Average_table
```

```
##                    Ho                 He                  F
## All 0.2871 ( 0.004 ) 0.3747 ( 0.0043 ) 0.2327 ( 0.0059 )
```

Then we can test the hypothesis that Ho-He is significant across all markers using a paired t-test, firstly checking if the Ho and He variances are equal. Is the result as expected?

```
bartlett.test(list(Teo_Summary$Hexp, Teo_Summary$Hobs))
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  list(Teo_Summary$Hexp, Teo_Summary$Hobs)
## Bartlett's K-squared = 3.1346, df = 1, p-value = 0.07665
```

Test the hypothesis that Hobs>Hexp across all markers:

```
t.test(Teo_Summary$Hexp, Teo_Summary$Hobs, pair=T, var.equal=TRUE, alter="greater")
```

```
##
##  Paired t-test
##
## data:  Teo_Summary$Hexp and Teo_Summary$Hobs
## t = 34.023, df = 591, p-value < 2.2e-16
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  0.08336362        Inf
## sample estimates:
## mean of the differences
##              0.08760561
```

We will now rerun these analyses for each state separately. The script shows how to do this for Mexico state (MEX).

```
#separate into "populations" by state:

teosinte_bypop<-seppop(teosinte)
teo.GUE<-summary(teosinte_bypop$GUE)
teo.JAL<-summary(teosinte_bypop$JAL)
teo.MEX<-summary(teosinte_bypop$MEX)
teo.MIC<-summary(teosinte_bypop$MIC)

# repeat above analysis for MEX only
```

Table 7.3: Ho, He and F for Mexican population and states.

	Ho	He	F
Entire population	0.287 (0.004)	0.375 (0.004)	0.233 (0.006)
Guerrero	0.298 (0.004)	0.375 (0.004)	0.203 (0.007)
Jalisco	0.221 (0.005)	0.326 (0.006)	0.295 (0.009)
Mexico	0.312 (0.005)	0.362 (0.005)	0.137 (0.008)
Michoacan	0.300 (0.005)	0.357 (0.005)	0.157 (0.007)

```
barplot(teo.MEX$Hexp[1:40]-teo.MEX$Hobs[1:40], xaxt='n', main="He-Ho in MEX for first 40 loci ", ylab= "He-H

Fvalues<-1-(teo.MEX$Hobs/teo.MEX$Hexp)

hwt.MEX<-hw.test(teosinte_bypop$MEX, B=0)
hwt.MEX[1:40,1:3]
sum(hwt.MEX[,3]<0.05)

Ho_ave.MEX<-round(mean(teo.MEX$Hobs),4)
Ho_se.MEX<-round(se(teo.MEX$Hobs),4)
He_ave.MEX<-round(mean(teo.MEX$Hexp),4)
He_se.MEX<-round(se(teo.MEX$Hexp),4)
F_ave.MEX<-round(mean(Fvalues),4)
F_se.MEX<-round(se(Fvalues),4)

Average_table[2,1]<-paste(Ho_ave.MEX, "(", Ho_se.MEX, ")")
Average_table[2,2]<-paste(He_ave.MEX, "(", He_se.MEX, ")")
Average_table[2,3]<-paste(F_ave.MEX, "(", F_se.MEX, ")")
rownames(Average_table)[2]<-c("MEX")
Average_table

bartlett.test(list(teo.MEX$Hexp, teo.MEX$Hobs))
t.test(teo.MEX$Hexp, teo.MEX$Hobs, pair=T, var.equal=TRUE, alter="greater")

# finally calculate F statistics across all markers (this may take a while)
fstat(teosinte)
```

For other populations, I have compiled the data in the exercise below.

So for doing the same for all states, we get the following values for Ho, He and F.

4 Inspect the above table and give your conclusions: how do individual states compare to the entire population?

Finally, we can calculate F statistics, considering each state as a population. This takes a few minutes so we provide the values here. Averaged over all loci , these are shown below this paragraph. F_{IS} is a measure of within population inbreeding, F_{IT} is a measure of total inbreeding and F_{ST} is a measure of population divergence. Note that $1 - F_{IT} = (1 - F_{IS})(1 - F_{ST})$, as it should be.

$F_{IS} = 0.205$

$F_{IT} = 0.246$

$F_{ST} = 0.052$

5 What can you conclude from these statistics?

By now you may have some opinion on the importance of the population structure in this data set. You may also have noticed some differences among the sub-populations. We will further investigate this using principal component analysis (PCA) of the entire molecular marker matrix using R. PCA is a statistical technique which summarizes the variation observed across all markers into a smaller number of underlying component variables. Each principal component accounts for a decreasing amount of variation in the dataset, e.g. PC1 accounts for the highest amount of variability in the original data as possible, and each subsequent PC explains the highest amount of variance possible in the remainder of the data. As a result, PCA finds the number of effective dimensions in the data defined by the significant PCA axes. If two major groups are present one dimension (PC1) might be enough to describe it. If three, then an extra axis might be necessary, and so on and so forth.

The data for this analysis are in a file called "teosinte_imputed_R.txt". I have *imputed* missing data in this data set – you will learn more about this later in the course (for the previous analysis, missing data were left as "NA"). We will use some simple R code to conduct a PCA on the data.

Let us load the data and required package `ggfortify`, you will need to install this if you don't have it already.

```
library(ggfortify)
#load data (delete 'data/' if you copy this!)
teosintepca<-read.table("data/teosinte_imputed_R.txt", header=T)
```

First we fit the PCA and observe (and plot) the percentage of variance explained by each of the first 10 PCs.

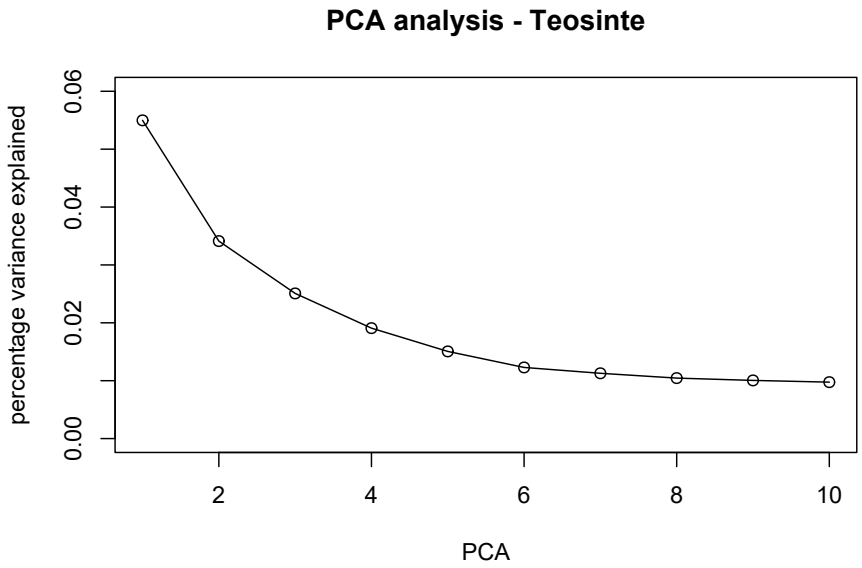
```
#define pca input (markers only, not state name)
pca.input<-teosintepca[c(1:592)]
#run PCA
pca.output<-prcomp(pca.input)
```

List the first 10 PCAs on your computer by running:

```
summary(pca.output)$importance[,1:10]
```

We can also plot the proportion of variance explained by first 10 PCAs by running:

```
plot(summary(pca.output)$importance[2,1:10], main="PCA analysis - Teosinte", ylab="percentage variance expla
lines(summary(pca.output)$importance[2,1:10])
```



Inspect these outputs to comment on the following question:

6 How much of the variation in this data set is explained by the first 10 PCs? What does this suggest in terms of the underlying population structure?

Now, we will plot the first two PCs against each other, colouring the points by geography (state).

```
#plot first 2 PCAs
autoplot(pca.output, data=teosintepca, colour='State')
```

Take a look at this plot and address the questions below:

7

a: Describe what you see in the plot. Is there (some) agreement with the classification purely on the geographical origin?

b: Based on the scatter plot you have just made, what seems to be odd about the group "Jalisco"? How does your observation here correspond with your findings in the fixation index for this sate?

We could have also plotted the alternative PCs (1 and 3 here). Try running this on your laptop.

```
pca_extra=pca.output
pca_extra$x=pca_extra$x[,c(1,3)]
colnames(pca_extra$x)=c("PC1","PC2")
pca_extra$rotation=pca_extra$rotation[,c(1,3)]
colnames(pca_extra$rotation)=c("PC1","PC2")
autoplot(pca_extra, data=teosintepca,colour='State')+labs(y="PC3")
```

Table 7.4: Ho, He and F for Mexican population and sub-divided states.

	Ho	He	F
Entire population	0.287 (0.004)	0.375 (0.004)	0.233 (0.006)
Guerrero	0.298 (0.004)	0.375 (0.004)	0.203 (0.007)
Jalisco 1	0.199 (0.007)	0.245 (0.007)	0.161 (0.011)
Jalisco 2	0.242 (0.006)	0.329 (0.006)	0.244 (0.010)
Mexico 1	0.318 (0.005)	0.358 (0.005)	0.110 (0.008)
Mexico 2	0.249 (0.009)	0.245 (0.008)	-0.025 (0.015)
Michoacan 1	0.301 (0.005)	0.355 (0.005)	0.151 (0.007)
Michoacan 2	0.293 (0.009)	0.306 (0.007)	0.020 (0.018)

Given what we have observed with the PCA investigation, it seems that state boundaries may not be the best way of grouping populations. Looking at the output from the PCA, it seems 3 states (Jalisco, Mexico and Michoacan) may have at least 2 groups of distinct individuals in them. A closer look at the data reveals that in all 3 states, these individuals represent distinct natural populations. If we sub-divide the samples of each of these 3 states into 2 geographic groups based on the PCA data, we now have 7 geographic groupings. We can see what difference this makes to our population statistics by recalculating the Ho, He and F values, and the F statistics using the methods from part c above. Feel free to do this (replace the original pop file with the file "teosinte_pops_subdivided_R.txt") and explore the data. You will obtain the following summary.

F_{IS}, F_{IT}, F_{ST} averaged over all loci:

F_{IS} = 0.185

F_{IT} = 0.251

F_{ST} = 0.081

8 What conclusions can you draw from this exercise?

The reassignment of populations based on the PCA here seems a bit ad hoc. Data exploration of this kind is very important in population genetics but there are more powerful methods for exploring how many "true" populations there are in a geographic data set, as well as how to assign individuals of unknown provenance to these populations. We will explore these methods more this afternoon.

7.2 Other ways of analysing population structure

7.2.1 Distance Trees

We have examined population structure in natural populations of Teosinte, an outbreeding species. For contrast, and to see what plant breeders are more likely to encounter, we now investigate population structure in a breeding population of a self-pollinated species (barley). The data to be used is from a UK association mapping study of barley (AGOUEB[2], and includes 1500 mapped SNPs for 621 genotypes (varieties). We'll run our analysis on the barley AGOUEB data set, using SNPs from across the whole genome but thinned so that markers are >2cM away from each other. This prevents results being dominated by patterns of diversity from particular chromosome regions (especially the centromere – lots of markers but very little recombination. Thinning in this manner has become standard practice. It is easily forgotten but is important. In this case, 307 markers remain to estimate relationships between the genotypes. Note that markers were scored as 0, 1 or "NA", because we are dealing with inbred lines, so there are no heterozygotes. However, for one of the approaches below (bootstrapping) no missing data is allowed. Therefore, I have imputed the missing data. The imputed values should be clear as they have values intermediate between 0 and 1. Data are in "AG_for_R.txt," already formatted.

We are going to scale the data so that each marker has mean 0 and standard deviation 1. That way they all have equal weight in the analysis.

```
#load barley data, take out /data if you copy this
AG_data<-read.table("data/AG_for_R.txt", header=T)

#scale data so all markers have mean=0, sd=1
AG_scaled<-as.data.frame(scale(AG_data))
```

Check that this has worked by running the following on your computer:

```
#check scaling worked
mean(AG_scaled$GM1,na.rm=T)
sd(AG_scaled$GM1,na.rm=T)
```

Then calculate a distance matrix (distance between all lines). Here we are using a simple "Euclidean" measure, based on geometric distance; other options are available.

```
#calculate a distance matrix
dist_mat<-dist(AG_scaled, method='euclidean')
```

[2] http://agoueb.org

We can then use this distance matrix in two different clustering approaches. Firstly, *Neighbor joining*. A basic method of agglomerative clustering method where individuals are grouped into progressively higher order clusters. Bootstrapping is a method of obtaining confidence in each node:

```
#use neighbor joining with bootstrapping
par(cex=0.7) #changes font size so you can see more labels
tree<-nj(dist_mat)
boot.tree<-boot.phylo(tree,AG_scaled,FUN =function(x) nj(dist(x)),B=100)
```

```
## Running bootstraps:       100 / 100
## Calculating bootstrap values... done.
```

Now we can plot the tree:

```
plot(tree)
nodelabels(boot.tree,frame="none")
```

9 Use the zoom function to look more closely at the plot. How many higher-level groups can reliably be distinguished in this network?

Our second option is Principal Coordinate Analysis. For this dataset, PCoA is probably more informative than a tree. PCoA is related to, but different from Principal Component Analysis; PCoA analysis is based on distances between pairs of individuals.

Install and load the package called `labdsv` for principal co-ordinate analysis (there are many alternatives available).

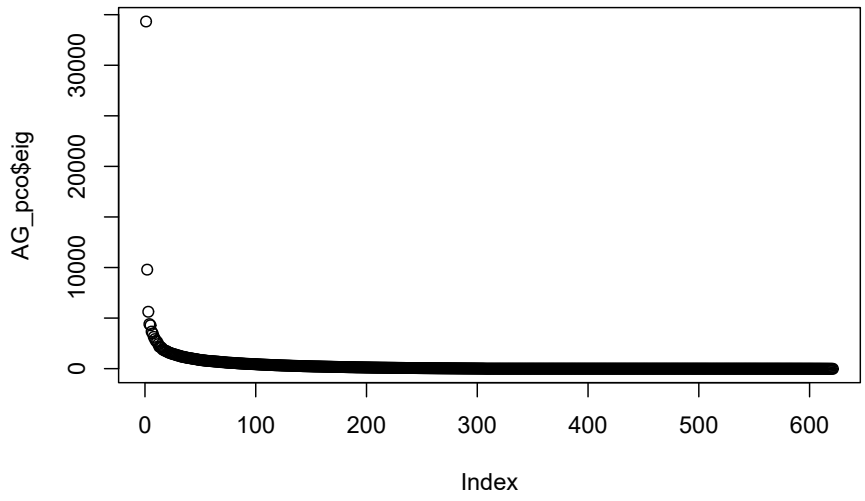
```
library(labdsv)
```

Carry out principal co-ordinate analysis:

```
AG_pco<-pco(dist_mat,k=10)
```

Plot the eigenvalues:

```
plot(AG_pco$eig)
```



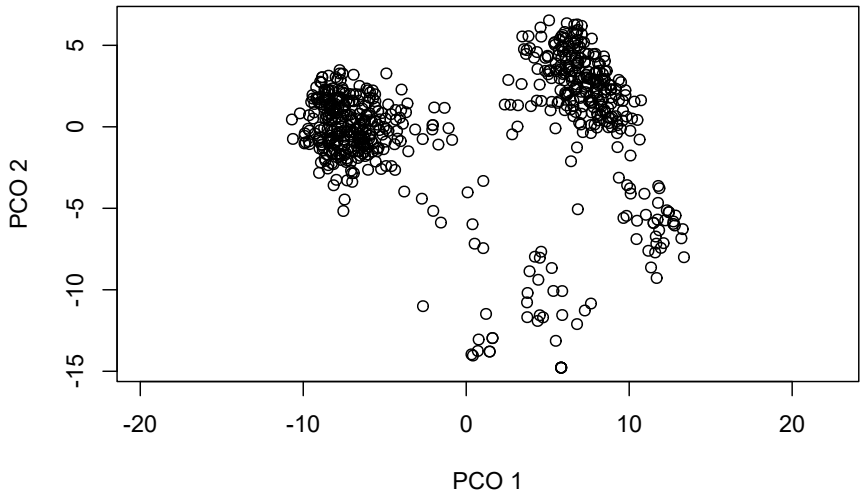
Tabulate percentage variation explained by top 10 eigenvalues:

```
pcntvar<-AG_pco$eig/sum(AG_pco$eig)
pcntvar[1:10]
```

```
##  [1] 0.18035911 0.05144023 0.02956470 0.02314040 0.02253805 0.01925869
##  [7] 0.01824109 0.01648530 0.01535072 0.01433016
```

Plot the first two pco axes:

```
plot(AG_pco)
```



You can also plot different pairs of pco axes using something like `plot(AG_pco$points[,1],AG_pco$points[,3])`.

Use your inspection of the PCoA results to answer the following questions:

10

a: Examine the Eigenvalues for the first 10 axes in the text output. What does this tell you about the structure in this data set?

b: Examine the graph for Axes 1 vs 2 but also axes 1 vs 3 and 2 vs 3. Compare your findings for the 2 approaches (NJ tree vs PCoA).

c: How many "natural" groupings do you think there are in this data set?

7.3 Structure

This exercise is to show you how to run STRUCTURE. STRUCTURE is easy to run but has its pitfalls. This guide should not be used as a substitute for the STRUCTURE manual, which gives more detailed guidance.

7.3.1 What STRUCTURE does

Imagine a population of a randomly mating diploid. Suppose we have a genome-wide set of markers, not necessarily mapped, but not closely linked to one another. Studying a sample of individuals from a single population, we would expect each marker to be in Hardy-Weinberg equilibrium. We would also expect every pair of markers to be in linkage equilibrium. Now imagine a second population of the same species. This too should be in equilibrium between and within markers. Suppose the populations have been separated for long enough that their allele frequencies have diverged and that mating within each population is at random. Now if we have a single sample drawn from both populations, but we don't know the origin of the individuals in the sample, then we'll find that the population is no longer in HW equilibrium nor in linkage equilibrium. However, by trial and error, we could allocate individuals to populations until we found the division which give HW proportions and no LD within each of the populations. In practice, allocating individuals by trial and error is impossible. STRUCTURE does this allocation for us by use of a model based Bayesian approach. It has proven tremendously successful. The source publication has been cited >2000 times according to Google Scholar. STRUCTURE will cope with more complex scenarios than that described above such as: more than two populations; ploidy levels other than diploid; ancestry of individuals mixed between populations; modest linkage between markers. Latest versions handle dominant markers. Other related programs focus on specific scenarios such as:

- *STRUCTURAMA* (available from same site as STRUCTURE). An alternative to STRUCTURE which includes calculation of the number of populations. Only works for outcrossed diploids.

- *INSTRUCT*. An alternative to STRUCTURE which works for partial selfers – estimates selfing rate simultaneously. It is available as a command line driven stand-alone or as a web based application run remotely from here: `http://cbsuapps.tc.cornell.edu/InStruct.aspx`.

- More recently, *ADMIXTURE*[3] uses the same statistical model as STRUCTURE but a faster optimization algorithm.

[3] `http://software.genetics.ucla.edu/admixture/`

7.3.2 Limitations of STRUCTURE

1. It takes time to run
2. Individuals are assumed to be random mating within populations – problems for mixed mating systems.
3. It is often not easy to decide on the number of sub-populations.
4. It is best at de-convoluting distant splits in population structure rather than more recent co-ancestry
5. Just as for PCoA, tree drawing, and most other methods of displaying genetic relationships, it is important to remember that the answer you get will depend on the distribution of markers. Once again, unlinked, or only loosely linked markers are advised for STRUCTURE. This is important. Remember to thin your markers if their distribution is patchy with clumps of markers in high LD. In our analysis, the markers have already been thinned we shall treat them as unlinked. In practice, this is what most people seem to do.

A note on selfing and ploidy. If species are obligate or near obligate selfers, then they should be analysed as haploids in STRUCTURE. Otherwise STRUCTURE will attempt to assign individuals to groups which are in HW equilibrium, and with a population of near-inbred lines, spurious clusters will result. For the few heterozygous markers that there may be, in something like barley for example, the simplest procedure is to switch these markers off. This is throwing out data, so as the degree of outcrossing increases it becomes more and more costly. The programme INSTRUCT should overcome these limitations; if you have a dataset for a crop with a more mixed mating system, then it is worth a go.

7.3.3 Using Structure

We will now again use the barley data from the AGOUEB project, thinned to 307 markers. The format of the data can be seen in the file "AG for Struct.txt." First we will learn how to use structure. Then, when we have some data, we will run through an exercise of how to interpret the data.

First we need to enter the data and learn how to set up a run. Structure guides you through data input process using a Windows style import wizard. To start the data import window: File, New Project...

Step 1

Enter the project name and select a directory and the input file. This process can be a fiddle – sometimes you have to click once to select, and sometimes twice. Trial and error should see you through, however.

Step 2

Enter information about the data. If you've forgotten or don't know this, clicking on "Show data file format" may help.

Don't forget to enter the "Missing Data Value." If you don't do this, the missing data will be treated as an extra allele. In this data file, the missing is -9 which is also the default.

Ploidy: For a collection of inbred lines enter 1 here to treat the data as haploid. Number of loci – look in the dataset

Step 3

This formats data input for any additional columns or rows in the file.

Select the row of marker names – our file includes this.

We are not using map data – Linkage information can be included in the analysis for loosely linked markers. Map distances between markers can be included at the top of the input file. See the STRUCTURE manual for details. Generally, good results are obtained treating markers as unlinked. In fact for markers which are linked and in high LD ($r^2 > 0.9$ or even less) it is worth deleting one of the pair (our marker thinning here has effectively done this).

Phase information: irrelevant here because we are dealing with a haploid organism! Generally, with outbred polyploid individuals, this box would need ticking if the phase was known. See the manual for further details.

Data file stores data for individuals in a single line: This also does not apply to haploids. With diploids, data may be entered on a single line as:

1 1 2 2 3 3 4 4 for four loci

or as:

```
1 2 3 4
1 2 3 4
```

The second format is the default for STRUCTURE.

Step 4

Individual ID for each individual: we have a unique variety code as the first column of our data, so select this.

Putative population of origin for each individual: With a priori knowledge of the population of origin of each individual, this can be entered and included in the analysis. This can work extremely well if, for example, you have a set of lines of known origin, and a set of unknown or admixed individuals. The data from the known lines is effectively used as a training set to classify the unknown lines. Here, leave it blank. If you had a mix of say, spring and winter wheats, you could use this as prior population information.

USEPOPINFO selection flag: Used in conjunction with the previous column to identify which individuals to use in the training set and which are to be classified without prior information. See the Structure manual for more details. Leave blank here.

Phenotype information: Do not select this.

Other extra columns: We don't have any.

Click on Finish, Check the details are correct, then click Proceed. STRUCTURE will test the format of the data and you will be prompted to correct any errors. If the data input is successful, a spreadsheet-like display of the input data should be returned.

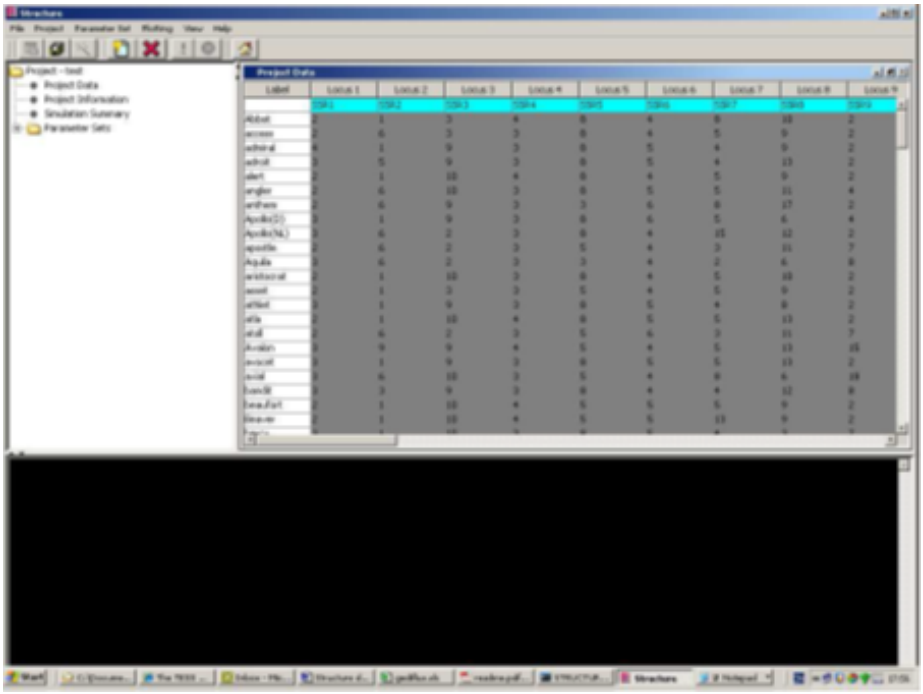


Figure 7.2: Spreadsheet-like display of the input data

Step 5. Parameter Sets

Before running STRUCTURE, we need to supply some input parameters. This is deceptively easy. However, the default parameters are not necessarily suitable for all datasets. See the manual for further details. One approach to selecting sensible parameter sets is to find a good publication working on a similar dataset to your own and copy from there.

Again there is an input wizard to guide you through the process. Select: Parameter Set, New...

Run Length: STRUCTURE uses Monte Carlo Markov Chain (MCMC) methods. To run successfully, the program iterates many times. There is generally an initial "burn-in"

period during which the program settles down, and then a further period in which the program runs and results are generated. The longer these periods are, the more reliable the results. In practice, the numbers selected are determined by the power of your computer and your patience. Generally, select a burn in and a run length of 100,000 at least. In this demonstration, for reasons of speed, select 10,000 for each box. In our experience, long runs are best – up to 106. One can use lower values for initial analysis and then crank up the numbers for the definitive runs.

Ancestry Model: Use the Admixture model. For most crop applications, we expect there will have been some crossing between sub-populations. Otherwise stick with the defaults. See manual for more details

Allele frequency model: Stick with the defaults – correlated allele frequencies. Again, although our subpopulations differ, for crops, we expect they haven't been isolated for so long that there is no correlation in allele frequencies between them. (If you were working on indica and japonica rice, you may have another view.) See manual for more details.

Advanced: "Compute probability of the data" should be ticked – we need this to work out how many subpopulations we may have. In addition, select "Print Q-hat". This writes the membership of each individual in each of the inferred populations to a separate file: useful for subsequent analyses.

Finally, click OK and give your parameter set a name.

Step 6 Running the program

We are now finally ready to run the program. To establish how many cryptic populations we have, STRUCTURE is run multiple times, varying the population number. "K" is the number of populations. To check for stability / repeatability of the STRUCTURE run, it is advisable to replicate each run several times. We shall have only two replicates – for reasons of time - and look at the range K=1-3.

Select: Project, Start a Job. Select the name of your new project. In a session running STRUCTURE you may create several different parameter sets (e.g. with different run times) and you would be prompted with a list here.

Remember, when running STRUCTURE on your own data, you would typically use a longer burn-in, a longer run-time, test more population numbers, and carry out more repeat runs.

Click Start. Structure will take a few minutes to run.

When the runs have finished:

Select: View, Simulation Summary This presents a table of summary information from each run. The most important column here is the fourth: Ln P(D).

Ln P(D) gives the posterior probability of the population number. Hopefully, you will see that this increases (gets less negative) as population number increases, but that values are reasonably close within replicate runs. Ideally, with more time, one would continue to increase K to find the value at which Ln P(D) was maximised. In practice,

this is not always possible: runs are unstable (see below), they take too long, or K can continue to increase to improbable values. Some compromise is required. The manual describes the problems of deciding on an appropriate value of K in more detail. We will discuss this below.

In the left hand side of the screen, select one of your runs. The right hand side screen should now change to display the results for that run. One can select various graphical displays of the results. It is worth exploring and experimenting with these. The "Data plot" options are very useful to check on the stability of the runs. In particular the plot of Log Likelihood against the number of iterations should be seen to stabilise during the burn-in and then fluctuate around a constant value during the run. There should be no trend upwards or downwards during this period (which would indicate a longer burn-in was required). The data plot of Ln P(D) may fluctuate initially, but should settle to a constant value by the end of the run, with no increasing or decreasing trend.

The "Bar plot" shows population membership for each individual in the dataset. This can be sorted by maximum population membership to give a cleaner display (Sort by Q). This plot is often seen in publications using STRUCTURE.

The "Triangle plot" shows group membership for any pair of populations, plus the residual pooled membership for the remaining populations, all in a single graph. Generally, if the dataset has population structure, and STRUCTURE has detected it, these plots will show clusters of individuals in the corners of the triangle (they come from that particular population), with some individuals scattered along the sides and in the body of the triangle (they are admixed between two or more populations). If this pattern is not seen, then you should suspect that there is no population structure, or none has been detected. We know from the values for Ln P(D) that at least three subpopulations are present. However, 10,000 iterations is too few for this dataset, especially once K is increased much beyond 3.

The "Tree plot" shows the relationships between the K populations in a run.

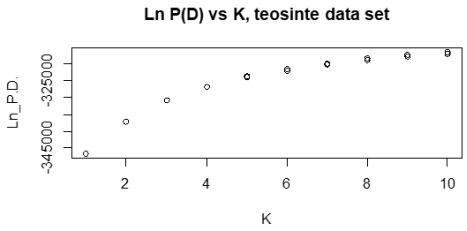
We can also plot the output files in R. View the output files in explorer – xxx_q and xxx_f. They should open in notepad or wordpad. The xxx_q file will also open in excel and read into R. The code below will produce a histogram of the maximum population membership for each variety. (The run below is one of the K=3 runs). Remember to set your directory in R before running the code. These plots are often bimodal and often show that many individuals have strong membership of one or other of the populations, but other individuals have mixed ancestry.

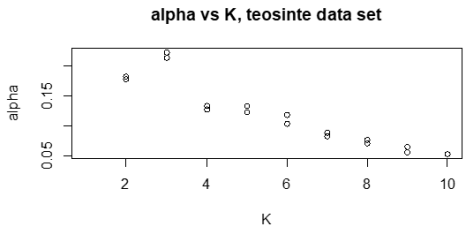
```
struct<-read.table("100000_run_5_q")
attach(struct)
hist(pmax(V2,V3,V4))
```

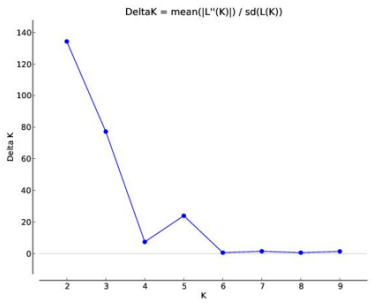
7.3.4 The exercise

I have run both the AGOUEB barely data set and the teosinte data set with 100000 run length for K=1-10 with 2 runs for every K. We are going to compare the two data sets to understand the strengths and limitations of analyses using STRUCTURE for analysing inbred crop species vs outbreeding wild species. The likelihoods and alpha values are plotted against K and the bar plots for K=3, K=5 and K=10 are shown below, together with the histograms of the maximum assignment value to a population for all individuals, for K=3, K=5 and K=10. In addition, using the Structure Harvester website[4], Earl and vonHoldt (2012), I have plotted the Delta K distribution of Evanno et al. (2005). This is a method for determining the most informative value of K.

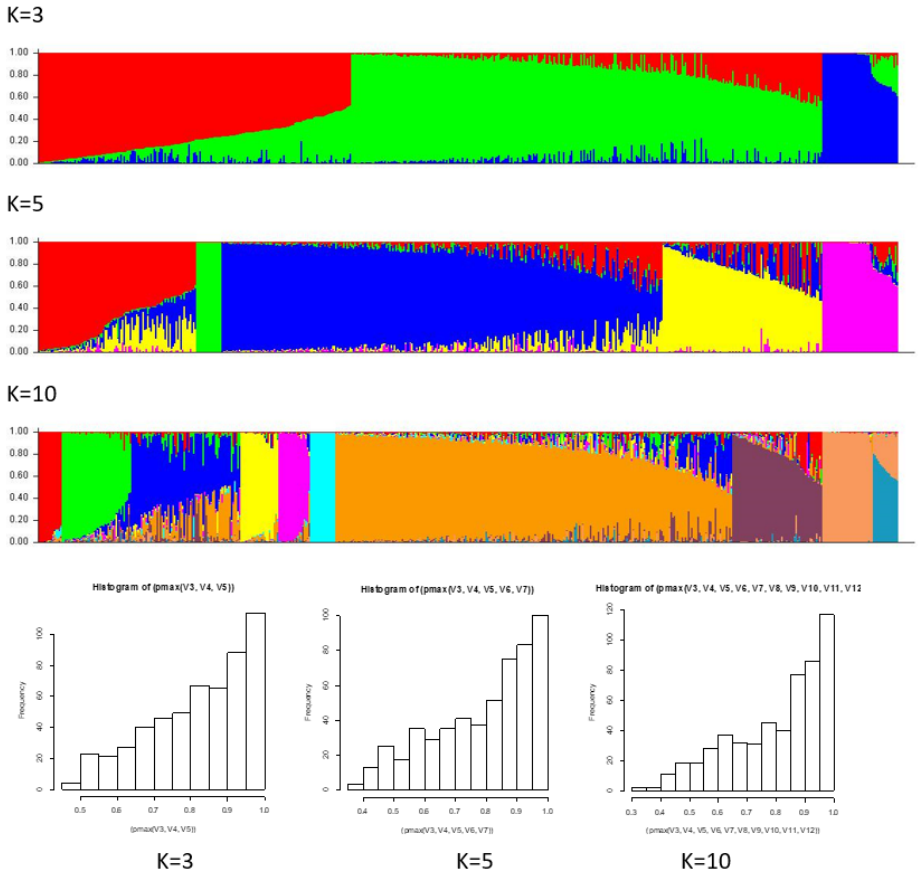
7.3.4.1 Teosinte data set







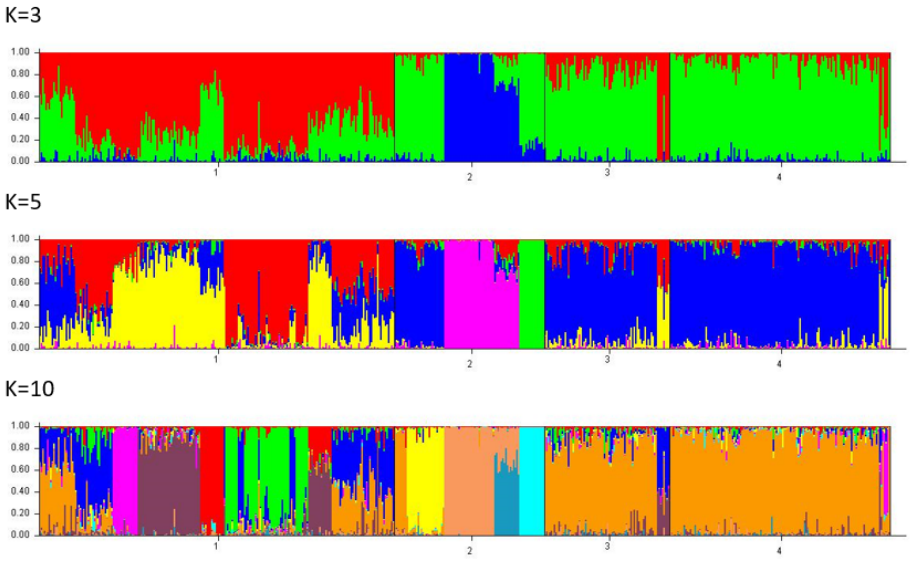
[4] http://taylor0.biology.ucla.edu/structureHarvester



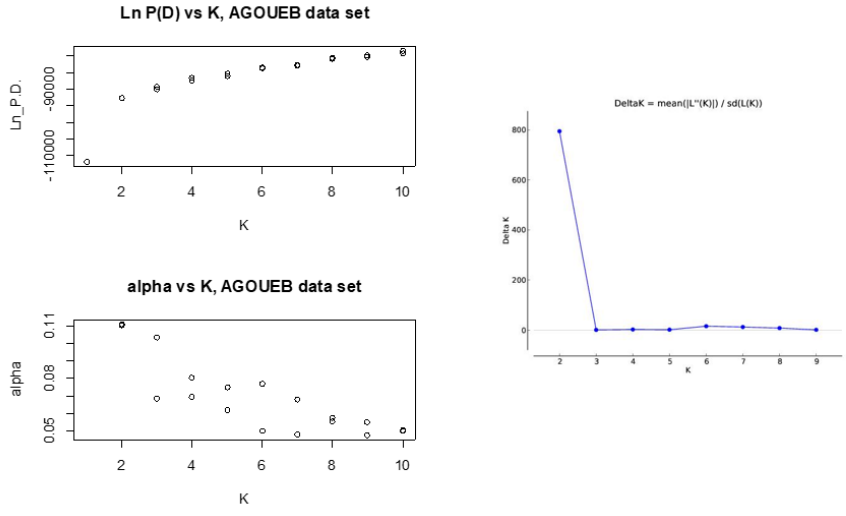
7.3.4.2 Teosinte data set alternative view with individuals ordered by natural populations within states

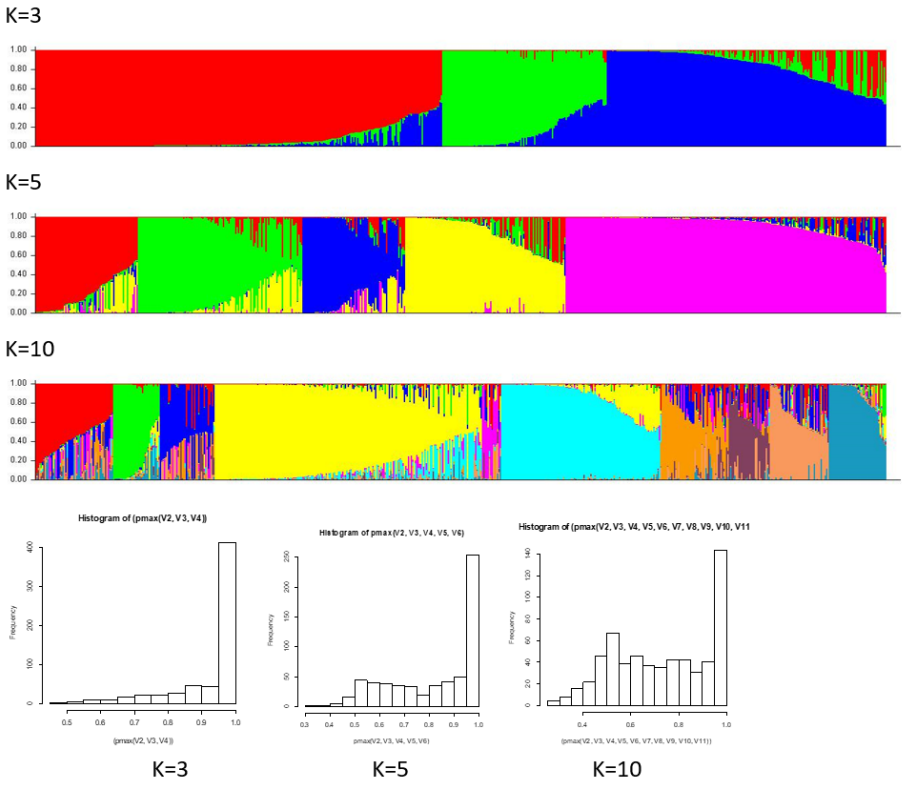
States:

1. Guerrero
2. Jalisco
3. Mexico
4. Michoacan



7.3.4.3 AGOUEB Barely data set





7.3.4.4 Exercise questions (teosinte data)

Looking firstly at the teosinte data:

11

Looking at the scatter plots, what happens to the likelihood, the rate of change of the likelihood and the value of alpha as K increases?

What appears to be happening as we move from K=3 to K=5 and K=10 in the bar plots? Use the histograms to confirm your observations. Note that the bar plots for the teosinte data set are arranged by collection population in 4 groups for the separate states (GUE, JAL, MEX, MIC).

What value of K may be the most appropriate for this data set, taking into account all the information in the figures above? (hint: there is no clear answer – think of how you might try to reach a conclusion)

You have now used several techniques to investigate the population structure of teosinte from field collections. As plant breeders, you may have conducted

this exercise in order to select lines for a crossing programme to improve your maize breeding germplasm (say for drought or disease resistance). Say you are selecting 15 lines, which would you select? Explain the reasons for your choice and which data analyses were most useful to reaching your decision. What other (non-genetic) information might you take into account?

7.3.4.5 Exercise questions (barley data)

Now looking at the barley data:

12

Looking at the scatter plots, what happens to the likelihood, the rate of change of the likelihood and the value of alpha as K increases – how does this differ from the teosinte data set

Looking at your own tree plot for K=3, what would you conclude about the relationship between the 3 inferred populations?

What appears to be happening as we move from K=3 to K=5 and K=10 in the bar plots? Use the histograms to confirm your observations. Is this different than the case for teosinte?

What value of K do you think is most appropriate for this data set, taking into account all the information in the figures above? How does this compare to your results from the PCoA and tree drawing exercise?

Do you think "Structure" is a useful tool for analysing inbred crop datasets? If not, what technique should we use to account for structure in association mapping?

7.4 Linkage disequilibrium analysis

Linkage disequilibrium mapping underlies all marker-trait association studies in crop genetics. Studying the pattern of LD in your population is one of the first things you should do before carrying out association mapping. It will help in deciding the marker density and the population size that you require for a well-designed association mapping study. This will be discussed again later in the course. There are several R packages that will estimate and plot LD but they seem to have quite fast turnover. Here, we will use a package called `LDcorSV`, which can calculate R^2 values accounting for population structure or kinship. We will estimate LD for both the teosinte and barley data sets, with and without accounting for population structure.

7.4.1 Linkage disequilibrium in the teosinte and barley data sets

Start RStudio and load packages; you may need to first install some of these packages. `ggplot2` is a very useful graphics package to get to know in R, and part of the Tidyverse[5]: publication-ready figures can be created very straightforwardly. `RcolorBrewer` is useful for designing your own colour schemes in heat maps. You will need to install the package `LDcorSV`.

We will investigate LD between loci by plotting LD against genetic distance along Teosinte chromosome 1 and barley chromosome 1 from the data sets we used earlier. For convenience I have used the imputed teosinte dataset but `LDcorSV` does accept missing data. The map positions are in "Teosintemap.txt" and "AGmap.txt". In this exercise we will compare LD decay with and without correction for population structure so we will also load two population structure data sets for teosinte: "Teosinte_State.txt" for the Mexican states where the sample were collected, and "Teosinte_structure_k5.txt" which has the population assignments from the Structure analysis with k=5. The columns in "Teosinte_State.txt" and "Teosinte_structure_k5.txt" are the probability of belonging to each population (always 0 or 1 for states). For barley, we will account for population structure using use the results of the PCoA, using the first two principal co-ordinates (3 populations). The number of columns in the population structure data files is 1-the number of populations; the last population can be omitted as the proportion of ancestry across all populations must sum to 1.

The command `LD.measures` from `LDcorSV` calculates all pairwise R^2 values, the option `S=` allows you to incorporate population structure. The output is a table with the format: marker 1 name / marker 2 name / R^2 value. To plot a LD heatmap, we need to rearrange the results into a matrix, and to plot LD decay against genetic distance, we need to calculate the distance between the marker pairs. The scripts for this data manipulation are provided – we will work through them and explain the details in the class.

7.4.1.1 Preparing the analysis

```
#load packages (you will need to install them first if you don't have them)
library("LDcorSV")
library("ggplot2")
library("RColorBrewer")
colnew<-colorRampPalette(c("grey","red"))(n=99)

#Read in the data (after setting working directory), delete "/data" if you copy this script

Teosintemap<-read.table("data/Teosintemap.txt",header=T)
Teosinte<-read.table("data/Teosinte_Chr1.txt",header=T)
```

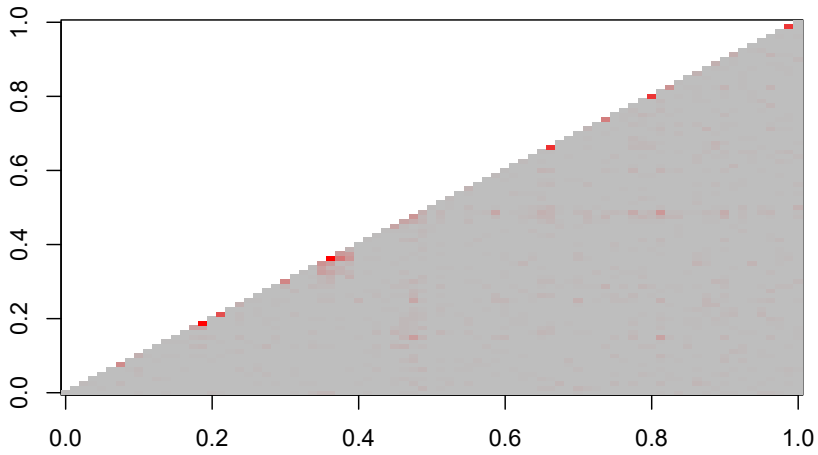
[5] https://www.tidyverse.org/

```
Teosinte_state<-read.table("data/Teosinte_state.txt",header=T)
Teosinte_structurek5<-read.table("data/Teosinte_structure_k5.txt",header=T)
AGmap<-read.table("data/AGmap.txt",header=T)
AG<-read.table("data/AG_Chr1.txt",header=T)
AGpcoa<-read.table("data/AG_Chr1_pcoa.txt",header=T)
```

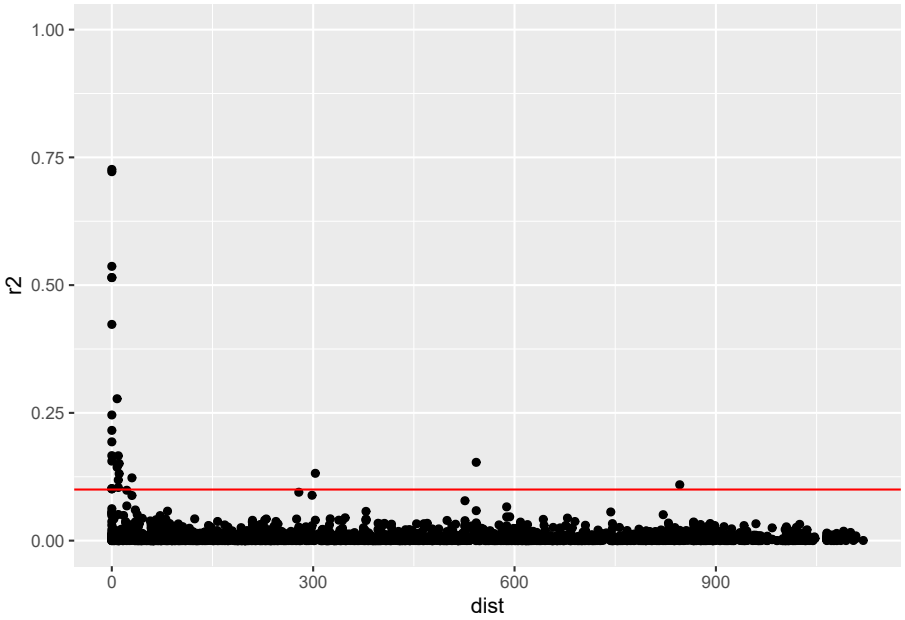
7.4.1.2 Teosinte analysis with no population structure

Run the script for the teosinte analysis with no population structure. You should obtain a heatmap of all pairwise r2 values and a scatter plot of the LD (values) versus the genetic distance:

```
#Teosinte: calculate r^2 without taking account of population structure

Teosinte_No_structure<-LD.Measures(Teosinte,na.presence=TRUE)

#Extract r^2 and the intermarker distances.
Teosinte_No_structure$dist="NA"
nmarkers=82
k<-0
for(i in 1: (nmarkers-1))
{
  for(j in (i+1):nmarkers)
  {
    k<-k+1
    Teosinte_No_structure[k,4]<-abs(Teosintemap[i,3]-Teosintemap[j,3])
  }
}

Teosinte_No_struct_heatmap<-matrix(NA,nrow=nmarkers-1,ncol=nmarkers-1)
k<-0
for(i in 1: (nmarkers-1))
{
  for(j in 1:(nmarkers-1))
  {
    k<-k+1
    ifelse(i<(j+1),Teosinte_No_struct_heatmap[i,j]<-Teosinte_No_structure[k,3], k<-k-1)
  }
}
Teosinte_No_structure$dist<-as.numeric(Teosinte_No_structure$dist)
image(t(Teosinte_No_struct_heatmap),col=colnew)
```




```
Teosinte_No_structure_ordered<-Teosinte_No_structure[order(Teosinte_No_structure$dist),]
ggplot(Teosinte_No_structure_ordered, aes(dist,r2))+
    geom_point()+
    ylim(0,1)+
    geom_smooth(formula=Teosinte_No_structure_ordered$r2~ Teosinte_No_structure_ordered$dist,se=F)+
    geom_hline(aes(yintercept=0.1),col="red")
```



Inspect the results and answer the following questions:

13

a: What is your preliminary conclusion based on the scatter plot. Does LD decay rapidly or slowly as a function of genetic distance between markers in this species?

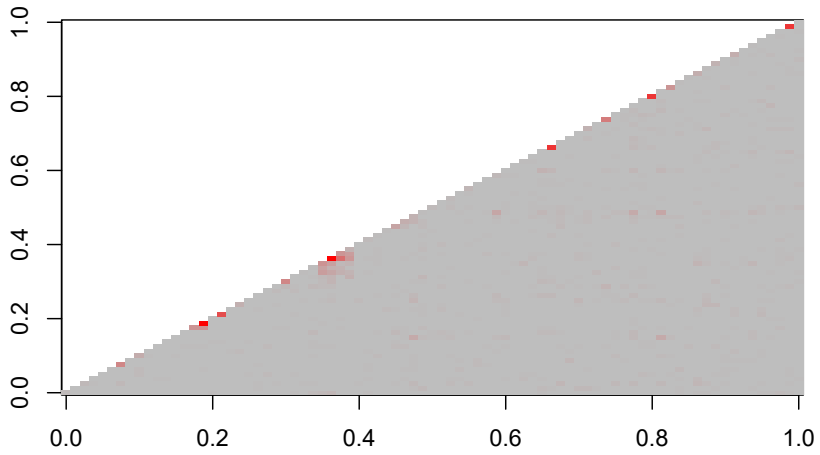
b: Now take a look at the LD matrix heatmap. What extra information is provided by this plot? Can you identify pairs of markers that show relatively high LD despite being quite far apart?

7.4.1.3 Teosinte analysis with population structure (states)

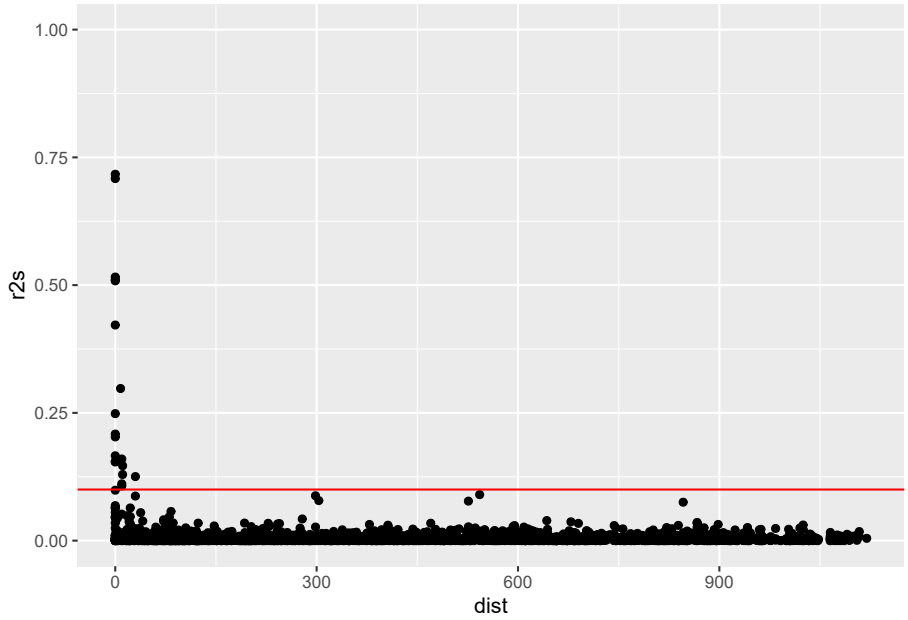
Rerun the teosinte analysis selecting state as a subpopulation grouping (the script is provided below):

```
#Teosinte: calculate r^2 taking account of population structure (states)

Teosinte_structure<-LD.Measures(Teosinte,S=Teosinte_state,na.presence=TRUE)


#Extract r^2 and the intermarker distances.
```

```
Teosinte_structure$dist="NA"
nmarkers=82
k<-0
for(i in 1: (nmarkers-1))
{
  for(j in (i+1):nmarkers)
  {
    k<-k+1
    Teosinte_structure[k,5]<-abs(Teosintemap[i,3]-Teosintemap[j,3])
  }
}

Teosinte_struct_heatmap<-matrix(NA,nrow=nmarkers-1,ncol=nmarkers-1)
k<-0
for(i in 1: (nmarkers-1))
{
  for(j in 1:(nmarkers-1))
  {
    k<-k+1
    ifelse(i<(j+1),Teosinte_struct_heatmap[i,j]<-Teosinte_structure[k,4], k<-k-1)
  }
}
Teosinte_structure$dist<-as.numeric(Teosinte_structure$dist)
image(t(Teosinte_struct_heatmap),col=colnew )
```




```
Teosinte_structure_ordered<-Teosinte_structure[order(Teosinte_structure$dist),]
ggplot(Teosinte_structure_ordered, aes(dist,r2s))+
  geom_point()+
  ylim(0,1)+
  geom_smooth(formula=Teosinte_structure_ordered$r2s~ Teosinte_structure_ordered$dist,se=F)+
  geom_hline(aes(yintercept=0.1),col="red")
```



Now consider the following questions:

14

a: Inspect the scatter plot. What is similar and what is different from the one obtained from the model that does not account for the population structure? Hint: Pay attention to the scale of the Y-axis.

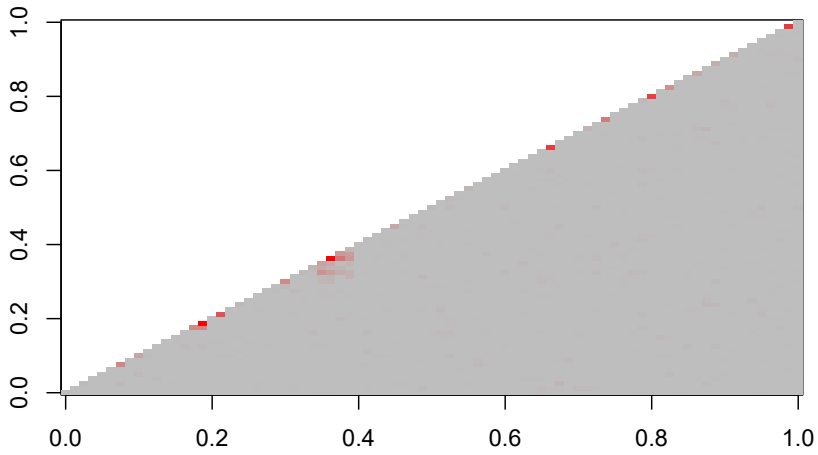
b: Compare the LD matrix plot with that obtained from the model that does not account for population structure.

7.4.1.4 Teosinte analysis with population structure (Structure k=5)

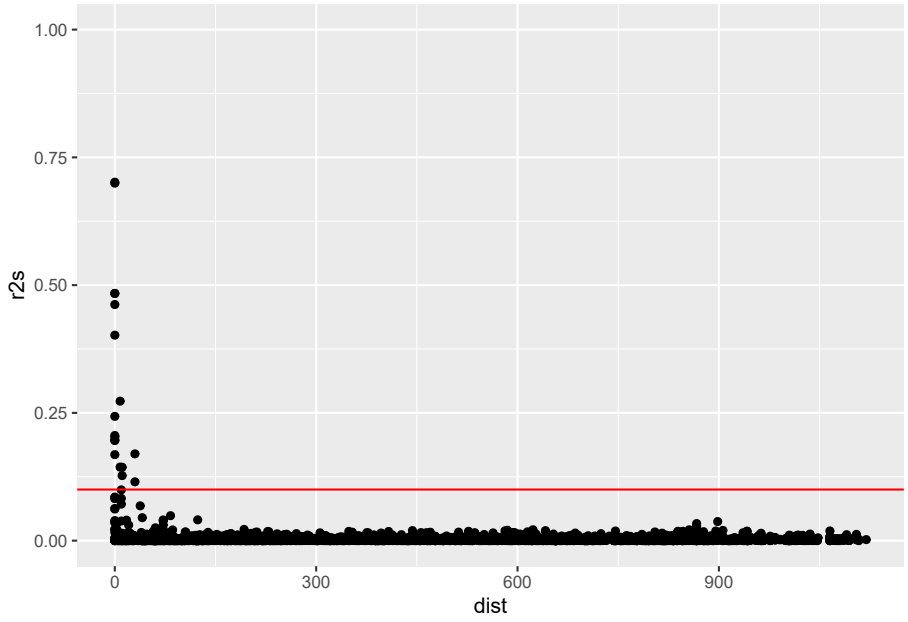
You might have observed some effect from the inclusion of the groups. However, we already know from the Hardy-Weinberg analysis that "state" does not seem to very accurately represent the true population structure in this data set. Rerun the analysis again using the Structure output as the population structure.

```
#Teosinte: calculate r^2 taking account of population structure (Structure k=5)

Teosinte_structurek5<-LD.Measures(Teosinte,S=Teosinte_structurek5,na.presence=TRUE)

#Extract r^2 and the intermarker distances.
```

```
Teosinte_structurek5$dist="NA"
nmarkers=82
k<-0
for(i in 1: (nmarkers-1))
{
  for(j in (i+1):nmarkers)
  {
    k<-k+1
    Teosinte_structurek5[k,5]<-abs(Teosintemap[i,3]-Teosintemap[j,3])
  }
}

Teosinte_structk5_heatmap<-matrix(NA,nrow=nmarkers-1,ncol=nmarkers-1)
k<-0
for(i in 1: (nmarkers-1))
{
  for(j in 1:(nmarkers-1))
  {
    k<-k+1
    ifelse(i<(j+1),Teosinte_structk5_heatmap[i,j]<-Teosinte_structurek5[k,4], k<-k-1)
  }
}
Teosinte_structurek5$dist<-as.numeric(Teosinte_structurek5$dist)
image(t(Teosinte_structk5_heatmap),col=colnew )
```




```
Teosinte_structurek5_ordered<-Teosinte_structurek5[order(Teosinte_structurek5$dist),]
ggplot(Teosinte_structurek5_ordered, aes(dist,r2s))+
  geom_point()+
  ylim(0,1)+
  geom_smooth(formula=Teosinte_structurek5_ordered$r2s~ Teosinte_structurek5_ordered$dist,se=F)+
  geom_hline(aes(yintercept=0.1),col="red")
```



15 Compare the results with those in the earlier parts. What do you conclude?

7.4.1.5 Barley association mapping panel

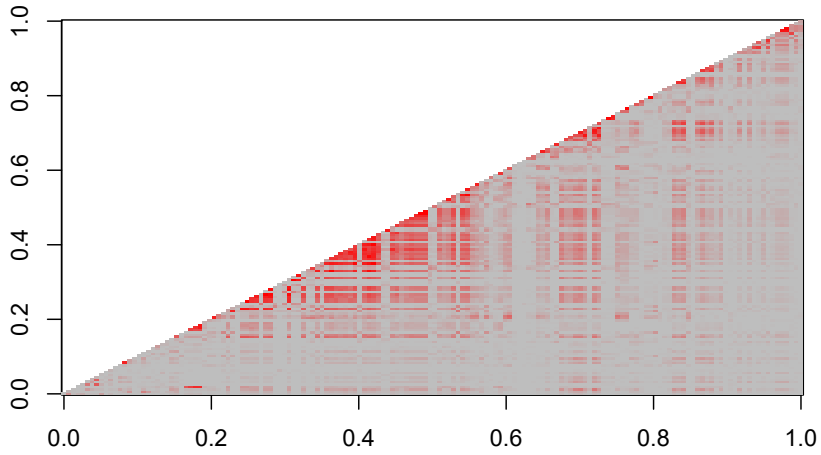
Understanding the breakdown of LD and whether the pattern of LD is affected by population structure are critical components of all association mapping studies such as the barely AGOUEB study.

16 Before looking at any results, what is your expectation of LD decay as function of genetic distance in the barley association mapping panel, compared to teosinte? Think of all the factors that differ between the two data sets and how they may affect LD.

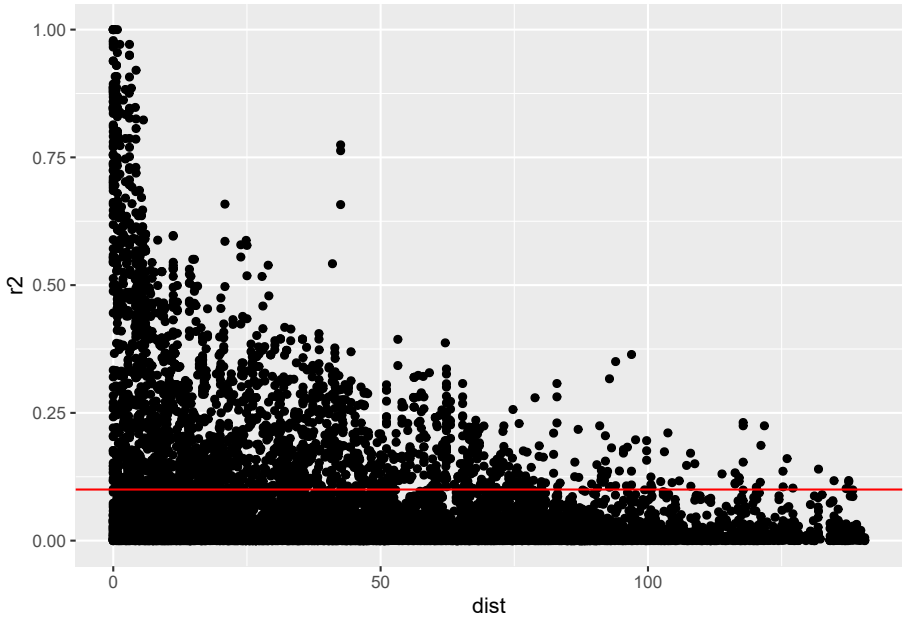
7.4.1.6 Barley association mapping panel (no population structure)

Run the LD analysis for barley chromosome 1 without population structure.

```
#Barley: calculate r^2 without taking account of population structure

AG_No_structure<-LD.Measures(AG,na.presence=TRUE)

#Extract r^2 and the intermarker distances.
AG_No_structure$dist="NA"
nmarkers=159
k<-0
for(i in 1: (nmarkers-1))
{
  for(j in (i+1):nmarkers)
  {
    k<-k+1
    AG_No_structure[k,4]<-abs(AGmap[i,3]-AGmap[j,3])
  }
}

AG_No_struct_heatmap<-matrix(NA,nrow=nmarkers-1,ncol=nmarkers-1)
k<-0
for(i in 1: (nmarkers-1))
{
  for(j in 1:(nmarkers-1))
  {
    k<-k+1
    ifelse(i<(j+1),AG_No_struct_heatmap[i,j]<-AG_No_structure[k,3], k<-k-1)
  }
}
AG_No_structure$dist<-as.numeric(AG_No_structure$dist)
image(t(AG_No_struct_heatmap), col=colnew)
```




```
AG_No_structure_ordered<-AG_No_structure[order(AG_No_structure$dist),]
ggplot(AG_No_structure_ordered, aes(dist,r2))+
  geom_point()+
  ylim(0,1)+
  geom_smooth(formula=AG_No_structure_ordered$r2~ AG_No_structure_ordered$dist,se=F)+
  geom_hline(aes(yintercept=0.1),col="red")
```



17

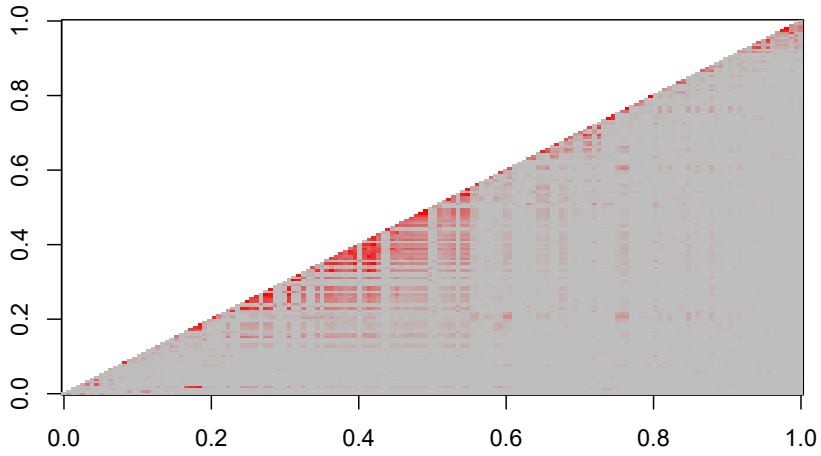
a: Inspect the scatter plot of r2 values versus genetic distance. How rapidly does LD decay in this population?

b: What is your initial interpretation of both plots?

7.4.1.7 Re-estimate LD in the barley data set controlling for population structure.

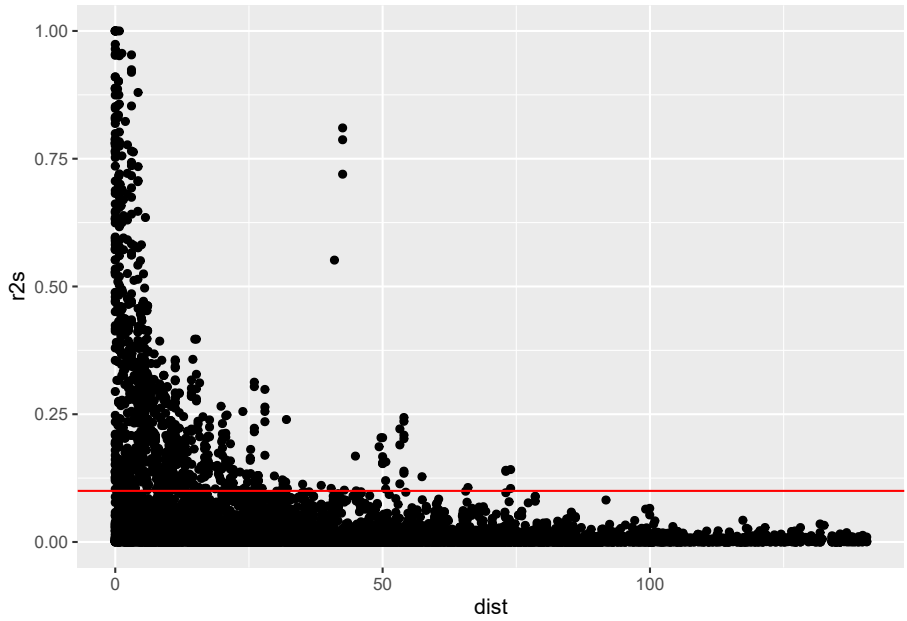
```
#Barley: calculate r^2 taking account of population structure (pcoas 1 and 2)

AG_structure<-LD.Measures(AG,S=AGpcoa,na.presence=TRUE)


#Extract r^2 and the intermarker distances.
AG_structure$dist="NA"
nmarkers=159
k<-0
for(i in 1: (nmarkers-1))
{
  for(j in (i+1):nmarkers)
```

```
  {
    k<-k+1
    AG_structure[k,5]<-abs(AGmap[i,3]-AGmap[j,3])
  }
}

AG_struct_heatmap<-matrix(NA,nrow=nmarkers-1,ncol=nmarkers-1)
k<-0
for(i in 1: (nmarkers-1))
{
  for(j in 1:(nmarkers-1))
  {
    k<-k+1
    ifelse(i<(j+1),AG_struct_heatmap[i,j]<-AG_structure[k,4], k<-k-1)
  }
}
AG_structure$dist<-as.numeric(AG_structure$dist)
image(t(AG_struct_heatmap),col=colnew)
```




```
AG_structure_ordered<-AG_structure[order(AG_structure$dist),]
ggplot(AG_structure_ordered, aes(dist,r2s))+
  geom_point()+
```

```
ylim(0,1)+
geom_smooth(formula=AG_structure_ordered$r2s~ AG_structure_ordered$dist,se=F)+
geom_hline(aes(yintercept=0.1),col="red")
```



18

a: Compare the scatter plot with that obtained from the null model. Does the new plot give a clearer picture of the LD decay than before? If so, why?

b: What is your conclusion regarding LD between pairs of markers in general when comparing the new LD matrix plot with that from the model that does not account for population structure? ?

7.4.1.8 Comparisons accross datasets.

Compare the results observed for barley with those for Teosinte.

19 What might be the explanation for the differences? How might this impact on association mapping studies?

7.5 Recombination and linkage disequilibrium: applications

The following figure is from a paper Gardner et al. (2016) which created a genetic map for wheat using an experimental MAGIC (Multiparent Advanced Generation InterCross) population. The population was created by crossing 8 distinct UK wheat varieties in all combinations over 3 generations, so that each individual has on average 1/8 of its genome from each parent. The lines are then selfed for several generations. This population has an overall high recombination rate compared to standard biparental mapping populations used for QTL detection. The top figure plots the genetic map (based on recombination frequencies) vs the physical map (in base pairs) for chromosome 3B. The genetic map is based on recombination frequencies estimated in the population. The bottom figure shows the recombination rate along the chromosome based on a comparison of the 2 maps. For mapping QTL, we would use the genetic map.

20

a: It is clear that the relationship between the 2 maps is non-linear. Explain why this is so with reference to the recombination rate graph

b: Why does there appear to be a large region of no recombination in the centre of the chromosome?

c: Recombination frequencies were estimated from the linkage disequilibrium generated in this population. What would a graph of LD along the chromosome look like? Which parts of the chromosome have high and low linkage disequilibrium? What implications does this have for QTL mapping in wheat?

d: As well as the centre of the chromosome, there are other shorter regions (e.g. around 610 Mb and 690 Mb) of the chromosome which have low recombination. The markers in these regions are also distinctive in being out of Hardy-Weinberg equilibrium. What could be going on here? (hint: think back to the causes of underdominance)

7.6 Pedigree analysis

Inspect the complex pedigree below.

21 Calculate the inbreeding coefficient of individual N in the following complex pedigree. This is difficult so we give two hints:

a: there are 7 possible paths b: two of the 7 paths have an inbred individual at the top – you must account for the fact that this individual is inbred.

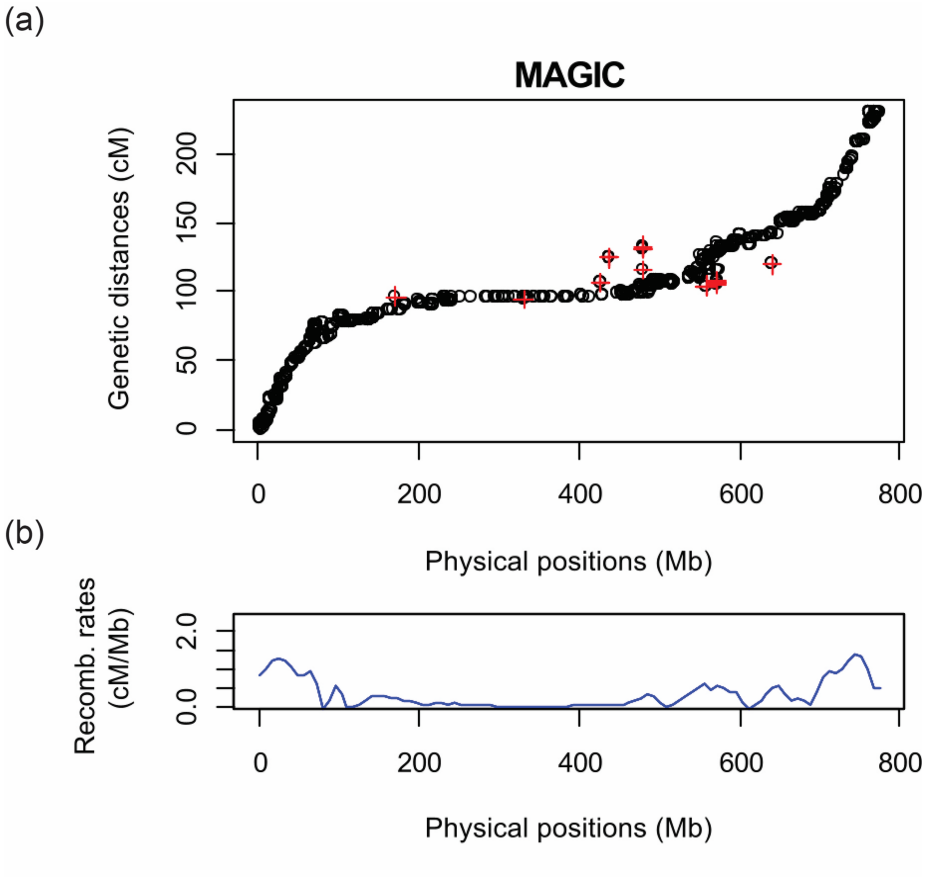


Figure 7.3: MAGIC genetic vs. physical map positions (a). Recombination rate in MAGIC population (b).

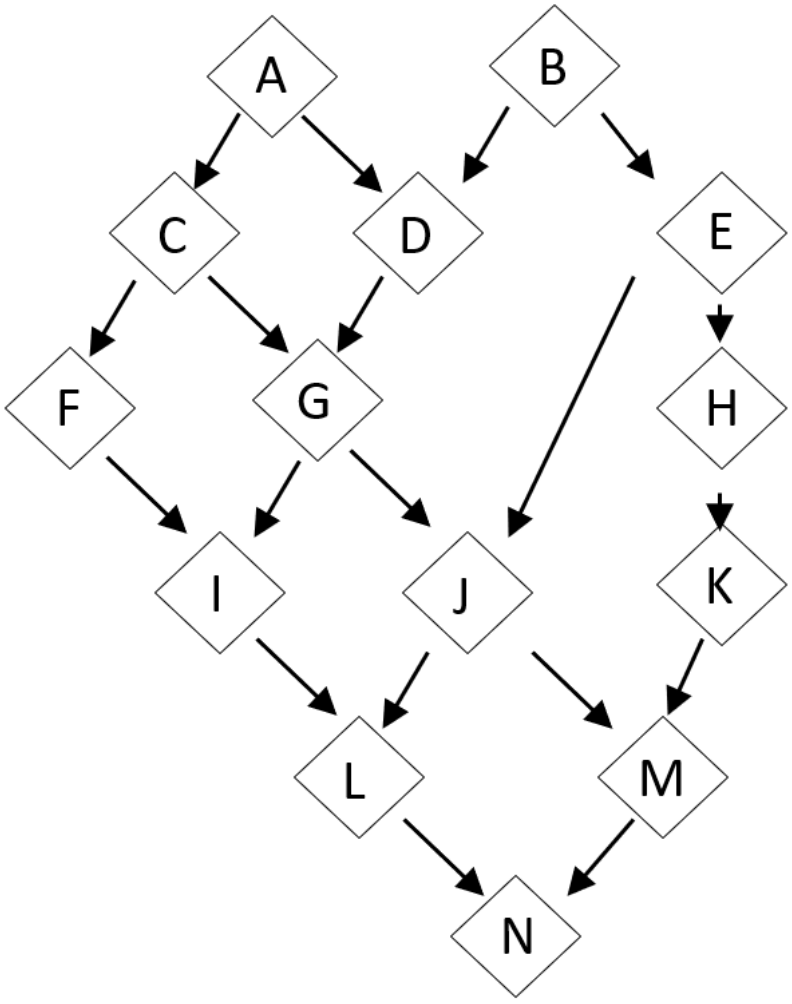


Figure 7.4: Example of a complex pedigree

Chapter 8

Imputation.

8.1 Introduction

Marker data almost always come with missing values. When working with a small number of values, say regressing a phenotype on a single marker, this is not a problem: the records with the missing data are simply dropped from the analysis. However, with very many markers, and in analyses which work on multiple markers simultaneously, this does cause problems. For example, computation of kinship from a marker set with missing data can result in a matrix which does not invert. Many methods using genomic selection also require complete data.

For this purpose, methods have been adapted to fill in the missing values with a best guess, and then analyses proceed as if the data were complete. The validity of using these imputed data as if they were real is questionable: if the imputation is accurate, then there is no problem, but with increasing inaccuracy come enhanced Type I and Type II error rates; significance tests assume you have more data than is true. Also, biased imputation (of heterozygotes for one of the homozygous classes say) could act to increase or reduce power – it is hard to predict which.

There are other uses of imputation. If we knew one marker could be perfectly predicted from a set of others, there would be no requirement to genotype it. Over the whole genome, this can save a lot of genotyping. This approach is described as "haplotype tagging" or SNP tagging. Imputation can also be used to search for genotype errors: where a predicted SNP call disagrees with the observed call this may indicate bad data rather than bad prediction. This is akin to the use of observed double recombinants in linkage analysis as indicators of genotyping error. Finally, in genotyping by sequencing, individuals are sequenced with very low coverage to keep costs down. Very many SNPs are detected but with a massive missing data problem. However, because the SNP density is so high there are always SNPs in high LD with each other that can be used to patch-in the missing data. That's the theory anyway.

There are specific methods for imputation of SNP data. These rely on having a genetic or physical map and explicitly or implicitly model linkage disequilibrium. These are much used in human genetics with very high densities of markers. Marchini and Howie (2010) give a review of methods and software.

For many of the datasets we have available we don't have an adequate genetic map so these methods cannot be applied. In addition, I am uncertain how well they perform with errors in map order or in allocation of markers to linkage groups, or whether the marker density used in many crops is adequate for their successful application. This is an area of active research.

Fortunately, imputation is not restricted to genetic applications so generic methods can also be used. It is some of these which we shall look at today. We'll first describe three possible approaches to imputation.

1. **Mean**: replace the missing marker genotype with the mean allele frequency (or twice the allele frequency for diploids). This isn't strictly imputation. It has the merit of being quick and easy. It is also probably adequate for many purposes where significance testing and/or precise estimation of effects are not necessary. Examples would be in computation of kinship matrices and in genomic prediction.

2. **Singular value decomposition (SVD)**: approximates the missing values from the first n row and column eigenvectors and values. This is not straight forward since SVD requires a complete matrix in the first place, so an initial guess at the missing values, usually the column means, is first made, followed by prediction from SVD, often with iteration until the prediction of the missing values stabilises.

3. **Random Forest**: This method is based on decision trees. A decision tree is a classification or regression system whereby data for an outcome variable are repeatedly split into smaller subsets on the basis of an optimally selected threshold of an optimally selected predictor. At each split different predictors/thresholds are selected. Subsequently, given data on the predictors, by following the tree through these splits, an estimate of the outcome variable is made. A random forest is a resampling method whereby multiple decision trees are built from bootstrapped samples of the data. For each bootstrapped dataset, at each split, a different subset of predictors is sampled at random from which the optimal predictor/threshold for that subset is then selected. The final set of random forest predictions is just the average of the predictions from each tree. The resampling ensures that the random forest predictions are more robust than those from single trees.

For more on these and other contemporary approaches to handling large data sets see James et al. (2013) for more details. A pdf is also available free online but it's worth buying.

8.2 The exercise

The aims of today's exercise are:

1. Compare the different imputation approaches.
2. Examine how accuracy of imputation decreases with decreasing LD among the markers.

We are going to work on a subset of data from a NIAB MAGIC population. We have extracted 4434 markers on 710 lines for which there are no missing data. This means we can artificially delete data, impute these missing values and compare the imputed values with the originals. Many of these markers are in very high LD with each other. We shall therefore delete one marker of each pair with correlation above a given threshold and repeat the delete–impute–compare process to assess how accuracy of imputation decreases with LD.

Below is a description of the work flow:

1. Skim the markers on correlation coefficients of <1, <0.9, <0.8, <0.7, <0.6, <0.5 to give seven datasets with reduced levels of LD among the markers. To speed things up we have already carried out this stage of the exercise (using the "deduplicate" script provided in the Chapter 8 data folder). The files are labeled "magic.full.txt", "magic.lt.1.0.txt", "magic.lt.0.9.txt",etc. These are in the data folder on the course website.
2. For each dataset, select 500 markers at random – so that accuracy of imputation isn't confounded with numbers of markers. We would expect better results with higher marker density.
3. For each dataset, delete 10% of the genotypes at random.
4. Impute the missing data and compare observed with imputed.
5. Tabulate the results for each LD threshold and method of imputation and discuss.

To speed things up, we are going to share the workload. Everyone will run the exercise using only one of the different MAGIC data sets (1.0, 0.9, etc), with 3-4 people assigned to each data set.

The order in which we run the analyses will depend on the time of day: the random forest approach is slow so best run over a break. We shall start with the svd package.

There is a R script the data folder for today: "Imputation_Script_2021.R". You can use this to avoid copying the sections below. Or just copy from this tutorial if you rather.

8.3 Data preparation (for all methods)

Note that the data are coded '0, 1, 2' and that there is a low frequency of heterozygosity among these lines – they are F4:5 families. Read the data in, here shown for the data after skimming on a correlation coefficient of 0.5

```
#delete the "data/" if you copy
magic.data<-read.table("data/magic.lt.0.5.txt")
```

Sample 500 markers using the function sample.

```
sample.500<-magic.data[,sort(sample(1:dim(magic.data)[2],500,replace=FALSE))]
```

Delete 10% of markers at random (may take a while):

```
missing<-sample.500
for (i in 1:710) {
  for (j in 1:500)  {
    if(runif(1) <0.1) missing[i,j]<-NA
  }
}
dim(missing)
```

```
## [1] 710 500
```

We can use the below script to calculate the mean NA % per marker:

```
#count NA per marker
na_count <-sapply(missing, function(y) sum(length(which(is.na(y)))))
#convert to a percentage
na_count <-(na_count/710)*100
#pull mean
mean(na_count)
```

```
## [1] 9.974366
```

On average we have 10% missing data per marker, as we intended. You now have a data set ready to be imputed.

8.4 SVD imputation

In this exercise, we shall use the package `softImpute` which implements a sophisticated version of SVD which seems to work, though I don't wholly understand the method.

The basic command in softImpute is: `softImpute(data.matrix, rank.max = 100).`

`data` is the name of the matrix holding your data.

`rank.max` is the maximum rank of the solution. It must be less than or equal to the smallest of either the number of rows or columns. The larger the value, the longer the programme will take to run but potentially the more accurate the solution. The more structured the data, the lower `rank.max` could be. If `rank.max` is too large we risk overfitting (adding noise) and accuracy of imputation could decrease. We are going to run `softImpute` with 500 markers at a time, so `500` is the maximum possible value. There are other parameters that one can adjust, but this is the minimum we need and seems to work well on this dataset.

There are two other useful commands within the package that we shall use:

```
biScale(data,row.center=F,row.scale=F,col.center=T,col.scale=T)
```

`biScale` standardises rows and columns to means of zero and variances of one. It is similar to the standard R command scale, except it can scale over rows and columns simultaneously. Marker data are generally standardised for use in PCO and kinship matrices so we shall adopt that as the default position here. In the package, the default is to standardise data over both rows and columns. We shall be explicit in defining that we only want columns (ie markers) standardised.

```
complete(data,softImpute.output.file,unscale=T)
```

`complete` takes the original data, and substitutes the imputed data for the missing values. A convenient option is `unscale=T`. This transforms the imputed values from the standardised scale (if you used `biScale`) back to the original. This back transformation may not always be required, but for comparison of original and imputed data it makes things easier.

All the other commands we shall use are standard R for manipulating and displaying the data and results.

Now we can start....

Scale the selected data:

```
library(softImpute) #you will need to install if you have not already
missing.scale<-biScale(missing,row.center=F,row.scale=F,col.center=T,col.scale=T)
```

Impute the missing data – may take a while. Note we have set the maximum rank of the solution to 100.

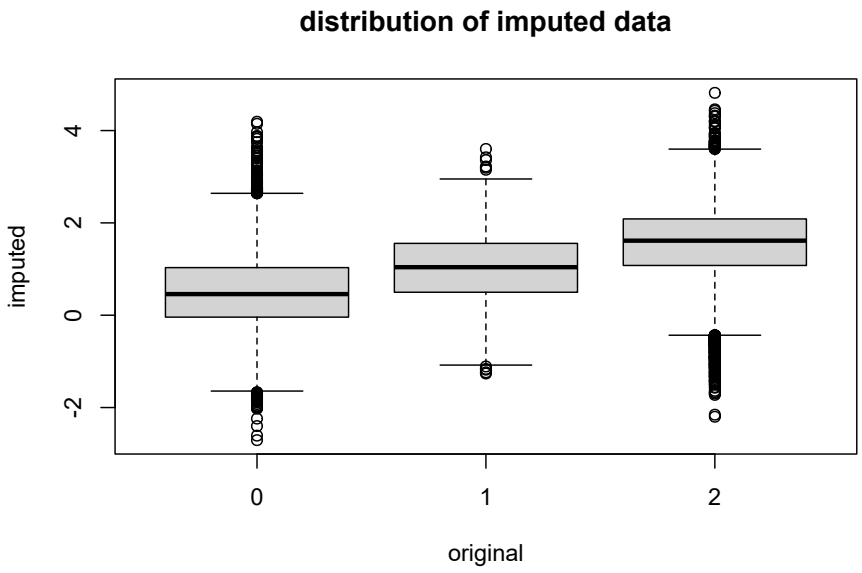
```
missing.scale.imp<-softImpute(missing.scale,rank.max=100,lambda=0)
```

Create a new matrix, including the imputed data, transformed back to the original scale.

```
missing.imp<-complete(missing,missing.scale.imp,unscale=T)
```

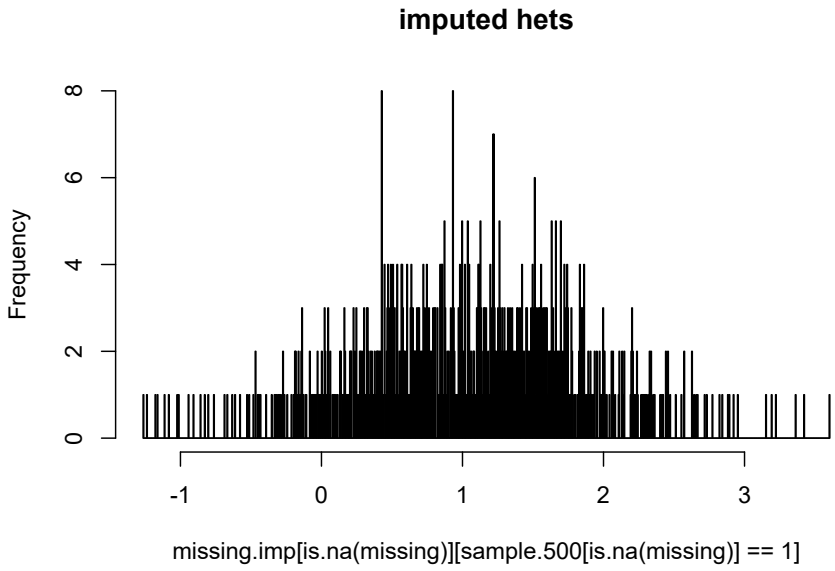
Plot the imputed data for each original class:

```
boxplot(missing.imp[is.na(missing)]~sample.500[is.na(missing)],main="distribution of imputed data",ylab="imp
```

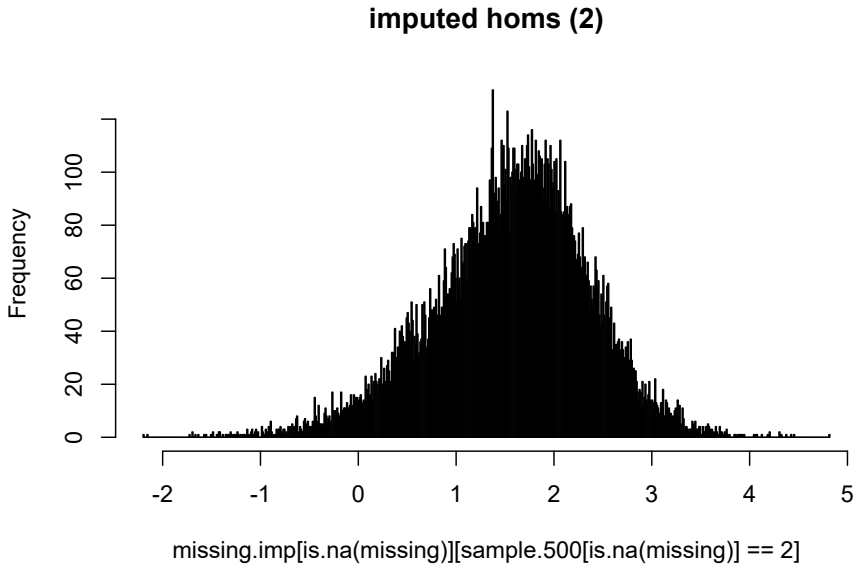


Plot histograms of the imputed data for each class in turn:

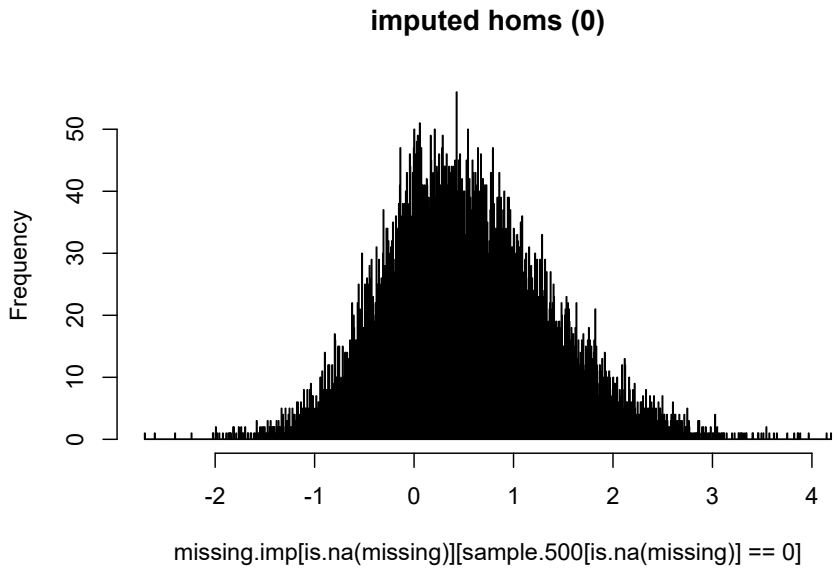
```
hist(missing.imp[is.na(missing)][sample.500[is.na(missing)]==1],breaks=1000,main="imputed hets")
```




```
hist(missing.imp[is.na(missing)][sample.500[is.na(missing)]==2],breaks=1000,main="imputed homs (2)")
```



```
hist(missing.imp[is.na(missing)][sample.500[is.na(missing)]==0],breaks=1000,main="imputed homs (0)")
```



What do you think?

The imputed data are on the original scale, but they are non-integer values. Round these to the nearest whole number. Values of 0, 1 and 2 represent the three genotype classes.

```
missing.imp.round<-round(missing.imp)
```

On your own computer, look at the plots of rounded, imputed genotype data:

```
boxplot(missing.imp.round[is.na(missing)]~sample.500[is.na(missing)],main="distribution of imputed data",yla

hist(missing.imp.round[is.na(missing)][sample.500[is.na(missing)]==1],breaks=1000,main="imputed hets")

hist(missing.imp.round[is.na(missing)][sample.500[is.na(missing)]==2],breaks=1000,main="imputed homs (2)")

hist(missing.imp.round[is.na(missing)][sample.500[is.na(missing)]==0],breaks=1000,main="imputed homs (0)")
```

Some integers are not <0 or >2. Round these up or down to the nearest genotype class.

```
missing.imp.round[missing.imp.round>2]<-2
missing.imp.round[missing.imp.round<0]<-0
```

Now, compare the imputed and original class using 'table':

```
imp.test<-table(missing.imp.round[is.na(missing)],sample.500[is.na(missing)],dnn=c("imputed","original"))
imp.test
```

```
##         original
## imputed     0     1     2
##       0  8205   216  1846
##       1  5624   403  6409
##       2  1917   242 10547
```

A single number we can use summarise the accuracy of the imputation is the contingency chi-squared test statistic – high values are good in this context: we want the rows (imputed) and columns (observed) to be highly dependent on each other.

```
chisq.test(imp.test)
```

```
##
##  Pearson's Chi-squared test
##
## data:  imp.test
## X-squared = 9952.8, df = 4, p-value < 2.2e-16
```

Has the imputation done a good job? Record the chisq value on the board in the classroom for subsequent comparison with other runs.

Aside from our methods comparison, there are a couple more things that can be investigated with softImpute:

a) Round all predictions to homozygotes only.
b) Change rank.max

We have observed above that most misclassifications are of true homozygotes as heterozygotes. e.g. for the full data set:

```
           original
imputed   0     1     2
0       11508   135   824
1       3303    623   3365
2       760     141   14691
```

Suppose we refused to impute missing genotypes as heterozygotes but insisted on classifying them has homozygotes:

```
missing.imp.round.2 <- 2*round(missing.imp/2)

missing.imp.round.2[missing.imp.round.2>2]<-2
missing.imp.round.2[missing.imp.round.2<0]<-0
imp.test.2<-table(missing.imp.round.2[is.na(missing)],sample.500[is.na(missing)],dnn=c("imputed","original")
```

The first line above forces genotypes which would be otherwise be imputed as heterozygotes into the nearest homozygous class. All the other lines of code are the same as before. The results seem better:

```
imp.test.2
```

```
##         original
## imputed     0     1     2
##       0 11641   415  4215
##       2  4105   446 14587
```

In practice, the wisdom of this reclassification is unclear. Carrying out the same exercise using mean imputation amounts to substituting all missing values by the most common allele, which doesn't seem very sensible.

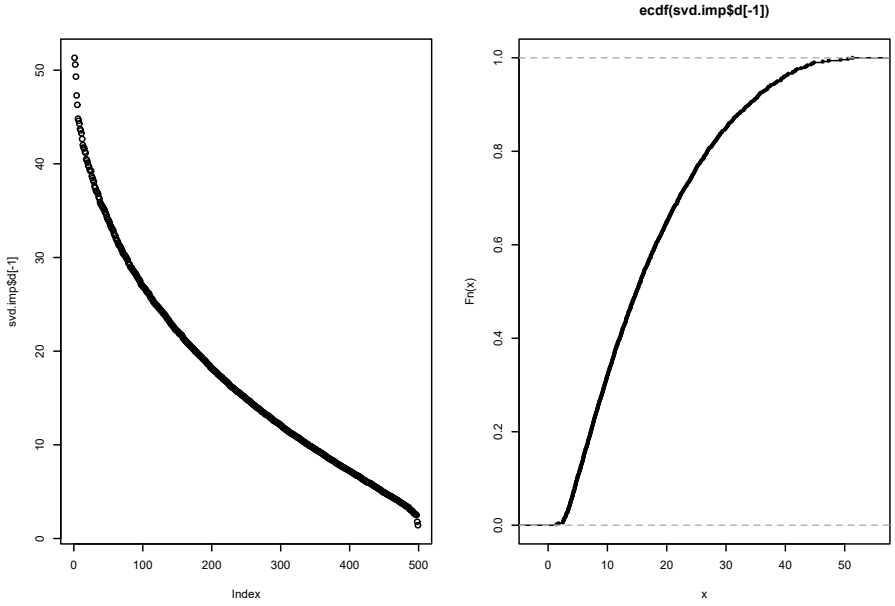
Why did we choose a value for `max.rank` of 100? This was initially by trial and error on my part. Too low a rank and the prediction is too poor. Too high a value and there is a risk of overfitting: additional eigenvalues give a better fit to the observed data but are adding noise to the predictions.

One approach to determining an appropriate value which works on these data, at least, is as follows:

First impute the missing data using the mean. To do this we need to work out the means:

```
av<-colMeans(missing,na.rm=T)

#Then, as before:

missing.imp.av<-missing
for (i in 1:500) {
  missing.imp.av[is.na(missing[,i]),i]<-av[i]
```

```
}

#Then compute the eigenvalues and plot them, individually and cumulatively:

svd.imp<-svd(missing.imp.av)
par(mfcol=c(1,2),cex=0.5)
plot(svd.imp$d[-1])
plot(ecdf(svd.imp$d[-1]))
```



`svd` is the standard R function to carry out SVD. `ecdf` is a function for plotting empirical cumulative distributions. We drop the first eigenvalue from the plot (the [-1]) since this is very large and its vector merely predicts the mean allele frequency of each marker. We could avoid this problem by rescaling the data first, but as we are aware of the problem it is hardly necessary.

The cumulative distribution would suggest a rank of about 50 (though my results were better using 100). If you have time, try altering `rank.max`, repeat the analyses and compare results.

8.5 Mean imputation

First work out the means:

```
av<-colMeans(missing,na.rm=T)
```

For each missing value, insert the mean allele frequency:

```
missing.imp.av<-missing
for (i in 1:500) {
  missing.imp.av[is.na(missing[,i]),i]<-av[i]
}
```

Round up or down as before and tabulate

```
missing.imp.av<-round(missing.imp.av)
missing.imp.av[missing.imp.av>2]<-2
missing.imp.av[missing.imp.av<0]<-0

imp.test.av<-table(missing.imp.av[is.na(missing)],sample.500[is.na(missing)],
                   dnn=c("imputed","       original"))
```

Inspect the results:

```
imp.test.av
```

```
##               original
## imputed    0    1    2
##       0 6316  142 1054
##       1 7693  489 7813
##       2 1737  230 9935
```

```
chisq.test(imp.test.av)
```

```
##
##  Pearson's Chi-squared test
##
## data:  imp.test.av
## X-squared = 9415.2, df = 4, p-value < 2.2e-16
```

What do you think? We'll collate the results over repeat runs and different values of LD threshold and discuss.

8.6 Random Forest Imputation

We shall also look at the random forest, as implemented in the R package `missForest`. This gave the most accurate results in a recent review Rutkoski et al. (2013). However, for accurate results it is slow to run.

Work on the same dataset as you used for svd and mean imputation.

`missForest` can impute missing factors as well as missing numerical values, which ought to be better for marker data, since we will no longer have numerical values which need to be rounded up or down to be {0, 1, 2}. When I tested this, this was indeed the case, though the analysis took two hours compared to twenty minutes. Nevertheless, we'll work on factors, but shall reduce the running time for demonstration purposes.

First we therefore need to change our numeric data to factor:

```
missing.factor<-missing
missing.factor[]<-lapply(missing.factor,factor)
```

Apply applies a function, here `factor`, across all the elements of the data frame. Using the `[]` at the end of `missing.factor` ensures the data frame structure is maintained. It runs quicker than a loop doing the same thing.

The Random Forest method takes bootstrap samples of the original data. In `missForest`, the default number of bootstrap samples is 100. In the interests of speed we'll just take 10 (`ntree=10`). If you have the computing power, missForest can be run in parallel. In addition to bootstrapping the data, the random forest algorithm selects a random sample of variables for each split of the tree. This prevents the bootstrapped trees from being too similar to each other. The recommended default number to be selected is √(no. of variables). Reducing the number will make the algorithm run faster. Increasing it may result in overfitting. The option mtry can be used to increase or reduce the number of selected variables. It is more important for the number of bootstrap samples to be large enough. Ten is too small: many lines will never be included in the analysis. There is no harm in having a very large number of bootstraps, but once a limit is reached there is no improvement either.

Install and then load the package in your R:

```
library(missForest) #you will also need to install
```

Running an analysis with 10 bootstraps only to save time:

```
#this took 2 minutes on my laptop
missing.imp.forest<-missForest(missing.factor,ntree=10,mtry=20)
```

```
##   missForest iteration 1 in progress...done!
##   missForest iteration 2 in progress...done!
```

We can also estimate imputation error using the function `mixError`:

```
mixError(missing.imp.forest$ximp,missing,sample.500)
```

```
##       PFC
## 0.2545398
```

for mixError, values close to 0 are good, close to 1 are bad.

We compare across methods using our earlier code, summarising the results:

```
chi.forest<-table(missing.imp.forest$ximp[is.na(missing)],sample.500[is.na(missing)],dnn=c("imputed","
chi.forest
```

```
##               original
## imputed     0     1     2
##       0 11593   427  3996
##       1    21    20    23
##       2  4132   414 14783
```

```
chisq.test(chi.forest)
```

```
## Warning in chisq.test(chi.forest): Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  chi.forest
## X-squared = 9738.7, df = 4, p-value < 2.2e-16
```

How does this compare with what we had before? Are there any differences in the pattern of assignment to different classes?

8.7 Summary

Now we have collected all the data, we can compare chi-squared values across methods and across different LD values. Is one method generally better, or do different methods work better with different amounts of LD in the data set? Do some methods perform more strongly as LD decreases?

8.8 Conclusions and comments

Before adopting any method of imputation, test it against simply substituting the mean (it could do worse).

For many analyses, substitution of the mean allele frequency is probably adequate. This applies particularly to kinship matrices.

With a new dataset, it is worth experimenting by deleting data to see how well an imputation method works.

You need a lot of markers for imputation. Therefore impute before you delete highly correlated markers, not after. i.e. thin markers after imputation.

Just because you've run an imputation programme, it doesn't mean the imputed values are accurate.

Problems surrounding imputation are going to become more important in the future as datasets continue to get bigger and methods such as genotyping by sequencing, become more common.

Random forest looks good but is slow: run overnight or in parallel. Random forest approaches are now becoming widely used in genetics in general, e.g. to predict trait values. Try the R package `randomForest`.

Chapter 9

Association Mapping.

9.1 Introduction

Association Mapping with GWASpoly

We are going to use the R package `GWASpoly` to carry out association mapping using the mixed model. We shall analyse the TriticeaeGenome data set[1] which we have used previously in this course. To recap: it was a European collaborative project which developed a panel of 384 UK, French and German winter wheat varieties. Here we are going to use the variety means across trials for a range of phenotypes together with a genome-wide set of DArT markers for association mapping.

`GWASpoly` is an R package from the Endelman group in the University of Wisconsin. The group is more noted for the well-established package `rrBLUP` for genomic prediction. You can see the group's software on their website[2]. The installation process for GWASpoly has recently changed. You use to be able to download from their website. However, the package is now hosted on GitHub[3]. If you carry on working with `R` you will find that a number of packages are hosted on GitHub. The only real difference between a package hosted on Github and through CRAN that we need to worry about for now is the installation process. `install.packages()` will not work in this case. Instead we will use the `devtools` package to install from Github:

```
#install devtools
install.packages("devtools")
#use devtools function to install from Github
# you may be asked to update some packages, just follow the promts in the console
devtools::install_github("jendelman/GWASpoly")
```

[1] http://www.triticeaegenome.eu
[2] http://potatobreeding.cals.wisc.edu/software/
[3] https://github.com/jendelman/GWASpoly

That should of installed `GWASpoly` and all the required associated packages. We should load the package and test it's installed ok (there should be no error messages):

```
library(GWASpoly)
```

GWASpoly is relatively straightforward to use. Its unique selling point is that it works on autopolyploid crops too. For these, the form of the kinship matrix changes, and this is accounted for. The main weakness of `GWASpoly` is that is can only handle bi-allelic markers, thus making it less useful for haplotype analysis. Several other GWAS packages are available for use in plants, notably TASSEL[4] and GAPIT (`http://www.zzlab.net/GAPIT/`), both originally developed in the Buckler lab at Cornell. More recently, the authors of GAPIT have released an improvement over GAPIT called FarmCPU[5], which incorporates recent methodological advances to reduce the number of false negatives which occur as a result of the adjustment for kinship and population structure. `GAPIT` and `FarmCPU` are both R packages.

For `GWASpoly`, there is a vignette available here[6], which is worth reading at some stage.

9.2 Preparing the data set

The complete TG data set is found in the file "TG data for GWAS.poly.xlsx". For reading into `GWASpoly`, two data files are required: a phenotype file and a genotype file which includes map co-ordinates. Here, phenotype data are in the file "TG_phens.csv". The first column must be the accession names. The next set of columns contains data on the traits to be analysed. There is then an optional final set of columns which gives covariates to be included in the analysis. In the TriticeaeGenome data there are six covariates:

1. PCR marker for the Rht2 dwarfing allele
2. PCR marker for the Ppd flowering time insensitivity allele
3. Year of registration of the variety
4. Country of variety origin (DEU, FRA, GBR)
5. 2 columns for country 0/1 "DEU", "FRA"

Genotype data for the GWAS are in the file "TGgeno.csv". Data here are DArT markers, coded 0/1(this is a data set of inbred lines). The data are transposed compared to the phenotypic data: variety names are in the first row rather than the first column. The first column here is the marker name, the second the chromosome number and the third the position of the marker in cM. As this is wheat, the chromosomes are numbered consecutively 1...21 for chromosomes 1A, 1B, 1D,2A, 2B..., 7B, 7D , Chromosome

[4] `https://www.maizegenetics.net/tassel,%20which%20can%20deal%20with%20multiple%20alleles`

[5] `http://www.zzlab.net/FarmCPU`

[6] `www.jendelman.github.io/GWASpoly/GWASpoly.html`

22 is of SNPs provided by BioGemma. Chromosome 23 is of unmapped DArT markers, and chromosome 24 is a small number of diagnostic markers for known QTL. Subsequent columns contain the marker data.

In preparing the data for analysis, markers with a minor allele frequency (maf) <4% were deleted. Most association mapping studies delete loci with low maf as they are unlikely, on their own, to detect any association, even if they are functional. The threshold is somewhat subjective. In the current case, <4% means less than 15 copies of the rare allele, which seems reasonable: think of doing a t-test where one group has fewer than 15 samples. The package itself, however, sets the minimum maf for inclusion in the analysis to 0.05, so we shall work with that. The maf threshold can be changed, however. Markers with >10% missing data were also deleted, as they are likely to be of poor quality.

The sheet "TGkin" contains a subset of the markers to use for kinship calculation. Markers are frequently unevenly distributed across plant genomes. If kinship matrices are dominated by dense clusters of markers, then any operations upon these matrices, such as PCO and tree construction, will be biased towards these markers. So a tree, for example, will not necessarily reflect average relationships over the genome, but the relationships among varieties in the genomic regions which have the highest number of markers. This is particularly a problem if, as is often the case, the dense clusters of markers are also in high LD. Such clusters are frequently found in crop species where non-recombining genomic regions from related species have been introgressed by plant breeders. For this reason, it is good practice to thin the markers to give a more uniform distribution over the whole genome. One way of doing this is to calculate the correlation coefficient among all markers and delete one marker of each pair in high LD (say r>0.9). Another is to delete markers if they are too closely linked. There is a fine balancing act in thinning markers. If markers are not uniformly distributed over the genome, then marker-trait associations will be over-corrected in some parts of the genome and under-corrected in others. If too few markers are left, then kinship will not be accurately estimated and adjusted for. For our data, we shall use only the mapped markers for kinship estimation, deleting all but one of any markers which map to the same position. This left 1,044 markers (from 2686 before thinning) selected for kinship calculation.

For some reason, no packages we are aware of provide this option of thinning markers selectively by correlation coefficient or by map position, despite the necessity of doing so. There are exceptions: Speed et al. (2012) developed an approach to calculating a kinship matrix which takes into account the clustering of markers without the need for thinning markers.

9.3 Running GWASpoly

A script has been provided with this tutorial. You can access it from the data folder for this Chapter. Open the script, load the GWASpoly package and set the working directory to the directory containing the tutorial data. You can also copy from the tutorial as we've done in the other tutorials.

The command to read in the data is `read.GWASpoly()`. Various data formats are available. Markers can be coded in AB format (common in linkage analysis), as ACGT nucleotides, or numeric; 0, 1, 2, 3, 4 for a tetraploid for instance. As this is an R package, missing data are coded as `NA`. GWASpoly only works with SNPs or binary markers. To analyse SSRs or haplotype data, each allele could be tested for association independently, possibly with some amalgamation of the results at the end, but this is an unnecessary complication for demonstration purposes. If you have a haplotype data set, using TASSEL is a better option.

Read in the data as follows. In my current working directory I have a subfolder called data, where all the data files are kept. Therefore, I start each file name with "data/", you will need to take this out if you copy the below:

```
TGall<-read.GWASpoly(ploidy=1,"data/TGphens.csv","data/TGgeno.csv","numeric",n.traits=18,delim=",")
```

```
## Number of polymorphic markers: 2686
## Missing marker data imputed with population mode
## N = 376 individuals with phenotypic and genotypic information
## Detected following fixed effects:
## Rht2_400
## PpdD1_297
## YEAR
## COUNTRY
## FRA
## DEU
## Detected following traits:
## YLD_ALL_BLUP
## CALLOW_2011
## FRANCE_2010
## FRANCE_2011
## LGE_2010
## LGE_2011
## NIAB_2011
## FT
## HT
```

```
## AWNS
## EARS_M2
## LODG
## MAT
## PROT
## SP_WT
## TGW
## TILLERS
## WINT_KILL
```

Once again, we are treating inbred wheat lines as haploids here as also reflected in the numerically coded data, where we have 0 or 1 copy of an allele. Note that we had to declare the number of traits. The extra columns in the input file will be considered (correctly) as covariates in the analysis by GWASpoly. Finally we have declared the field separator to be a ',' as we are reading a csv file. For other options use `help`.

On reading in the data you will see a summary of the data structure, now all stored in `TGall`. Notice that the output says `Missing marker data imputed with population mode`. We had some missing marker data but GWASpoly requires complete data so has imputed the missing values on data input using "population mode". This substitutes missing data by the most frequent genotype class at each marker. This is not the best method. Ideally, we would impute missing data by a more accurate method prior to analysis (see the imputation tutorial).

We also need to read in the version of the data with just the markers required for kinship calculation.

```
#take out the "data/" if you copy
TGkin<-read.GWASpoly(ploidy=1,"data/TGphens.csv","data/TGkin.csv","numeric",n.traits=18,delim=",")
```

Note, we have 1044 markers in the kinship dataset.

9.4 The exercise

The aim of the exercise is to familiarise yourselves with the package and compare the results obtained when:

a) Using different kinship adjustments, or without including kinship at all
b) Including known trait covariates (e.g. Rht, Ppd loci)
c) Using different population structure adjustments, or without including population structure at all.

9.4.1 Calculation and comparison of kinship

To estimate kinship in GWASpoly, we simply type:

```
TGall<-set.K(TGall)
TGkin<-set.K(TGkin)
```

Each version of the data has now had a kinship matrix calculated from its associated marker set. We'll see later how to specify different kinship matrices to be used. There are multiple forms of kinship matrices. GWASpoly centres each marker (so the mean of each marker is zero and the variance is one). If the centred data are in a matrix M, the kinship matrix is then MMt. The kinship matrix is also scaled so that the mean of the diagonal elements are one. This is the VanRaden kinship matrix (VanRaden, 2008), which is in common use. Alternatively, you can supply your own matrix (we'll see how shortly). MtM/n would be the variance/covariance matrix among the n markers if M was centred, and would be the correlation matrix if each marker was initially adjusted to have a variance of one too.

The kinship matrix is stored in a 'slot' in the GWASpoly object storing the data. Slots are accessed using `@` just as named vectors of data accessed using `$` Thus:

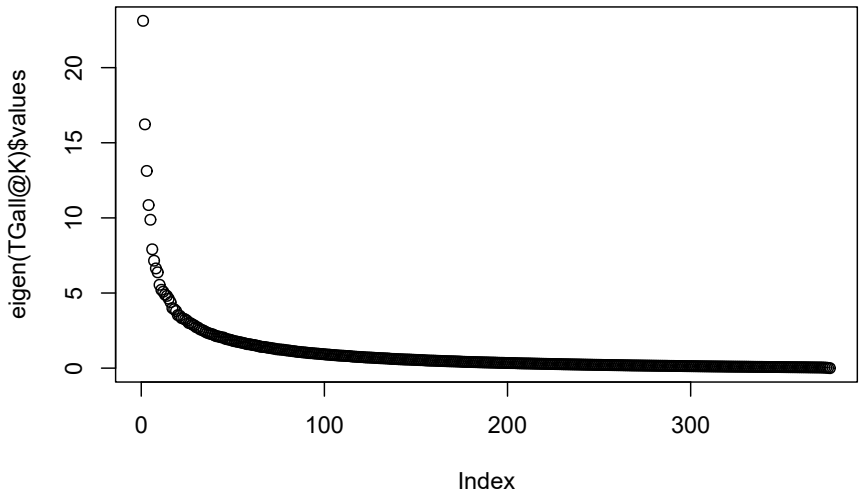
```
TGall@K[1:5,1:5]
```

```
##               AARDEN    AARDVARK       ABELE         ABO     ACCESS
## AARDEN    0.89425094  0.06820056 -0.08325033 -0.17401122  0.1904552
## AARDVARK  0.06820056  0.98608975  0.04025471 -0.03429426 -0.0102782
## ABELE    -0.08325033  0.04025471  1.16491365 -0.03983785  0.1856692
## ABO      -0.17401122 -0.03429426 -0.03983785  1.30763030 -0.1575344
## ACCESS    0.19045520 -0.01027820  0.18566924 -0.15753443  1.0314240
```

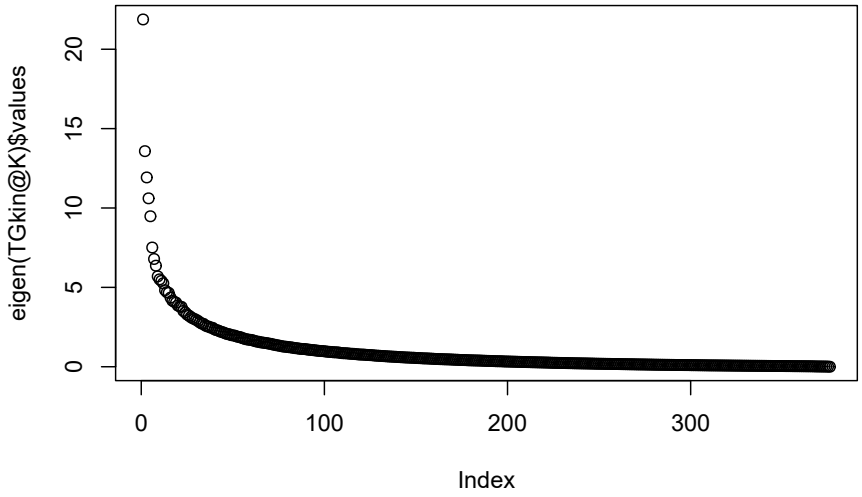
This shows the top left hand corner of the matrix made from all markers. Compare to `TGkin`.

These kinship matrices are essentially measuring the genetic distance between varieties using different marker sets. So, as you now know from the Population Genetics exercise, we can now visualise the genetic distances in a PCoA – principal coordinate analysis. The standard R function for spectral decomposition of a matrix is `eigen()`, which produces eigenvalues and vectors. We can plot the eigenvalues for both data sets:

```
plot(eigen(TGall@K)$values)
```




```
plot(eigen(TGkin@K)$values)
```

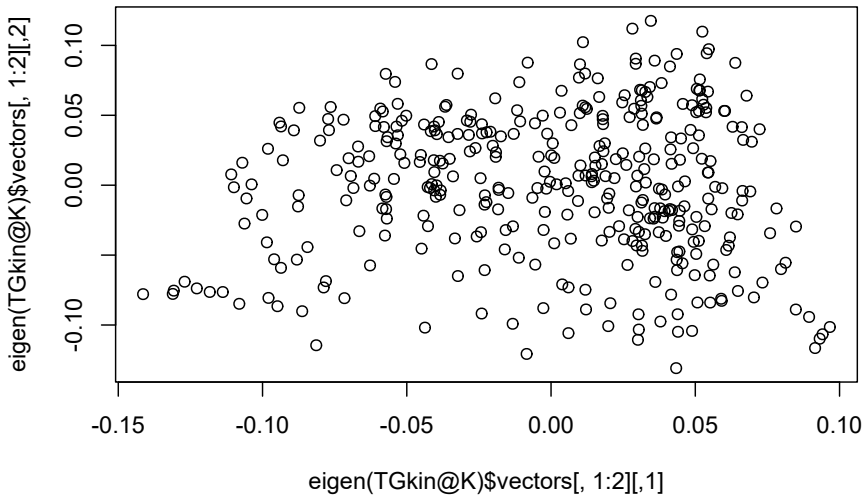


You should see that in both cases, the first few eigenvalues are much larger than the

remainder: they therefore account for most of the variation in their respective matrices. We can now plot the largest and second largest eigenvectors against each other for both data sets:

```
plot(eigen(TGall@K)$vectors[,1:2])
```


```
plot(eigen(TGkin@K)$vectors[,1:2])
```



1

The plots for all the data, and for the thinned markers alone are very different. What is the explanation for this? On the basis of this, which kinship matrix is most representative of relationships over the whole genome?

We now want to make two data sets for the GWAS analysis. The GWAS analysis itself will be conducted with all the genotype data, but we want one data set with kinship calculated from all markers:

```
TGanalysisUnthinned<-TGall
```

... and one data set with kinship calculated from only the thinned markers. This is remarkably easy to create: we just substitute in the kinship matrix calculate on the thinned markers:

```
TGanalysisThinned<-TGall
TGanalysisThinned@K<-TGkin@K
```

Then check:

```
TGanalysisUnthinned@K[1:5,1:5]
```

```
##               AARDEN    AARDVARK       ABELE         ABO     ACCESS
## AARDEN    0.89425094  0.06820056 -0.08325033 -0.17401122  0.1904552
## AARDVARK  0.06820056  0.98608975  0.04025471 -0.03429426 -0.0102782
## ABELE    -0.08325033  0.04025471  1.16491365 -0.03983785  0.1856692
## ABO      -0.17401122 -0.03429426 -0.03983785  1.30763030 -0.1575344
## ACCESS    0.19045520 -0.01027820  0.18566924 -0.15753443  1.0314240
```

```
TGanalysisThinned@K[1:5,1:5]
```

```
##               AARDEN    AARDVARK       ABELE         ABO      ACCESS
## AARDEN    0.94938532  0.07464131 -0.07579459 -0.17446488  0.24676194
## AARDVARK  0.07464131  0.92078798  0.07606171 -0.03447680  0.03070368
## ABELE    -0.07579459  0.07606171  1.04717181  0.02871512  0.08796148
## ABO      -0.17446488 -0.03447680  0.02871512  1.30082327 -0.16499556
## ACCESS    0.24676194  0.03070368  0.08796148 -0.16499556  1.04546733
```

9.4.2 GWAS

Strictly, in using the mixed model for GWAS, genetic variances and covariances should be estimated separately for every marker, since the estimates will vary depending on the magnitude of the SNP effect and the distribution of the marker alleles over individuals. However, given that most markers have very little effect, the variances change very little. Therefore GWASpoly follows an approximate approach outlined by Kang et al. (2010) and Zhang et al. (2010) in which the variances are estimated only once, in a model without any SNP effect. This greatly speeds up the analysis with little loss of accuracy or bias. To analyse all traits for the thinned data set:

```
TG.GWAS.Thinned<-GWASpoly(TGanalysisThinned,models="additive")
```

A choice must be made of what model of marker effects to fit. For pseudo-haploid data, as here, the choice makes no difference. In general a simple additive model is best. Other options (see `help(GWASpoly()`) are targeted at analyses with polyploids, where the multiple heterozygous classes can be pooled in various ways. `models ="general"` would fit separate additive and a dominance terms in a diploid species.

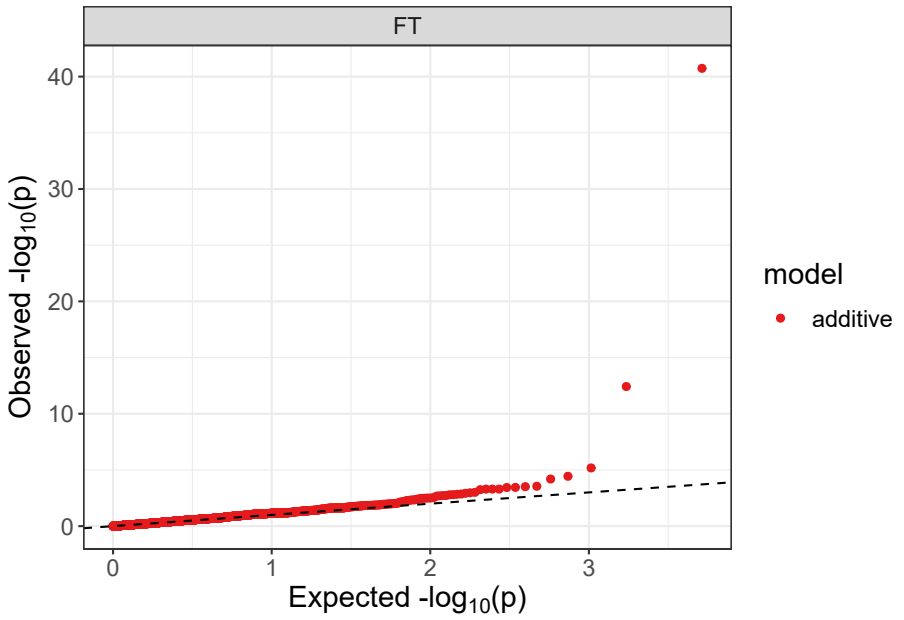
You can specify multiple models in the same analysis. More usefully for us, you can restrict the analysis to certain traits only. We will now focus on analysing only two traits, height and flowering time:

```
TG.HT.FT.Thinned<-GWASpoly(TGanalysisThinned, models="additive", traits=c("HT","FT"))
```

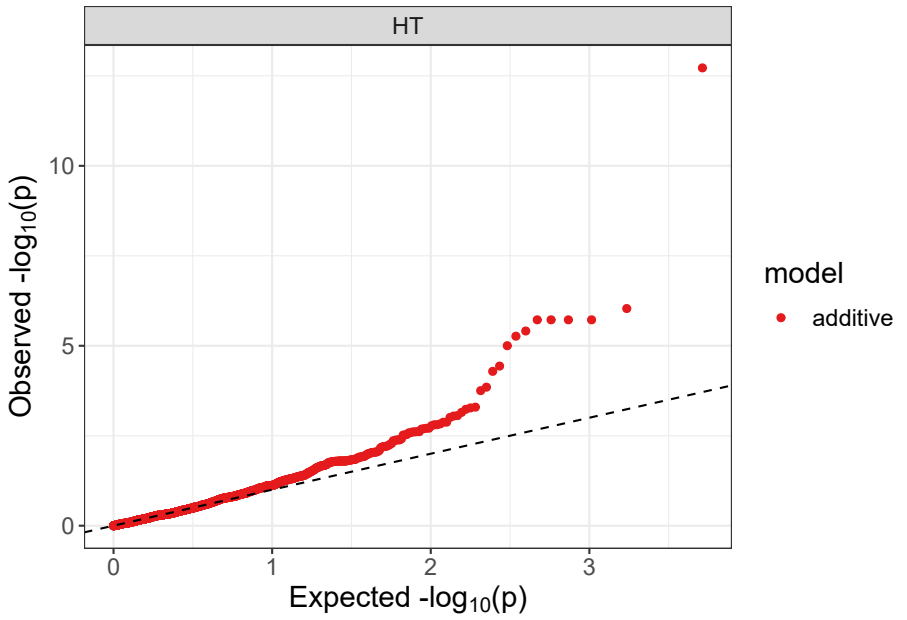
```
## Analyzing trait: HT
## P3D approach: Estimating variance components...Completed
## Testing markers for model: additive
## Analyzing trait: FT
## P3D approach: Estimating variance components...Completed
## Testing markers for model: additive
```

We now need to look at QQ plots:

```
qq.plot(TG.HT.FT.Thinned,trait="FT",model="additive")
```




```
qq.plot(TG.HT.FT.Thinned,trait="HT",model="additive")
```



These look OK. If the lower values did not fall close to the line, we would need to re-assess our corrections for kinship and/or take into account other covariates (such as population structure, as below).

What QTL have we found? First we need to specify a significance threshold:

```
TG.HT.FT.Thinned.Bonf<-set.threshold(TG.HT.FT.Thinned,method="Bonferroni",level=0.05)
```

```
## Thresholds
##    additive
## HT     4.71
## FT     4.71
```

You have three choices for setting the threshold: "`Bonferroni`", "`FDR`" (False Discovery Rate) and "`permute`" (permutation test). The permutation test is relatively quick but to do the default 1000 permutations would take too long for this exercise! I will show the results from this in the class exercise. For now, you can run the `"FDR"` threshold for comparison:

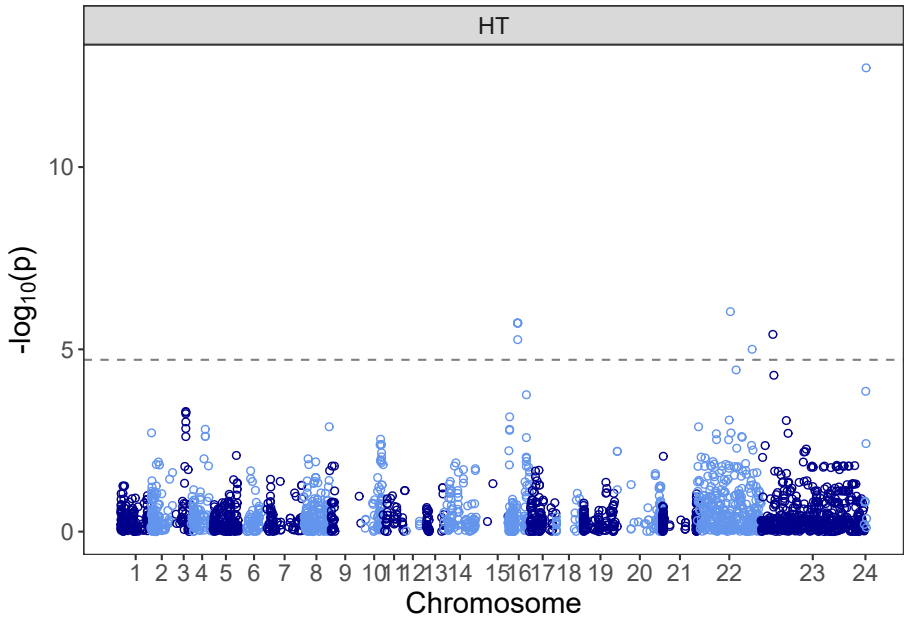
```
TG.HT.FT.Thinned.fdr<-set.threshold(TG.HT.FT.Thinned,method="FDR",level=0.05)
```

```
## Thresholds
```

```
##    additive
## HT     3.47
## FT     3.73
```

Having set the threshold, you will find it is plotted on any Manhattan plots you create provided there is something that exceeds the threshold. We can plot Manhattan plots from our analyses as:

```
manhattan.plot(TG.HT.FT.Thinned.Bonf,trait=c("HT"),model="additive")
```



You must specify a single trait and a single model.

We can examine the significant results directly:

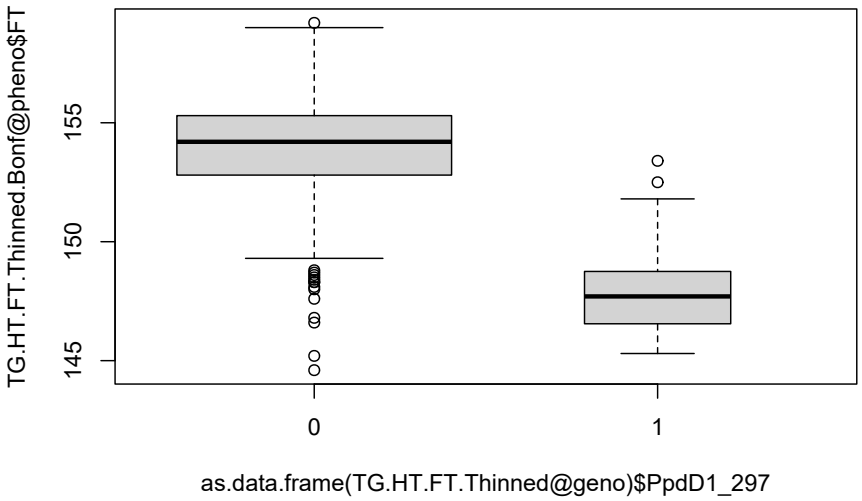
```
get.QTL(TG.HT.FT.Thinned.Bonf)
```

```
##      Trait    Model Threshold    Marker Chrom Position Ref Alt Score    Effect
## 1363    HT additive      4.71 DArT_1448    16  48.7860   0   1  5.72  4.87e+00
## 1364    HT additive      4.71 DArT_1449    16  48.7860   0   1  5.72  4.87e+00
## 1365    HT additive      4.71 DArT_1451    16  49.1736   0   1  5.27  4.70e+00
## 1366    HT additive      4.71 DArT_1452    16  49.1736   0   1  5.72  4.87e+00
## 1367    HT additive      4.71 DArT_1453    16  49.1736   0   1  5.72  4.87e+00
```

```
## 2042    HT additive      4.71   SNP_168    22 167.0000   0   1  6.03  3.32e+00
## 2150    HT additive      4.71   SNP_278    22 277.0000   0   1  5.00 -3.03e+00
## 2253    HT additive      4.71 DArT_2072    23  59.0000   0   1  5.41 -3.18e+00
## 2684    HT additive      4.71  Rht2_400    24   9.0000   0   1 12.72 -5.51e+00
## 654     FT additive      4.71  DArT_678     6  84.5399   0   1  5.18  1.66e+00
## 2065    FT additive      4.71   SNP_191    22 190.0000   0   1 12.42 -4.22e+00
## 2682    FT additive      4.71 PpdD1_297    24   7.0000   0   1 40.74 -4.81e+00
```

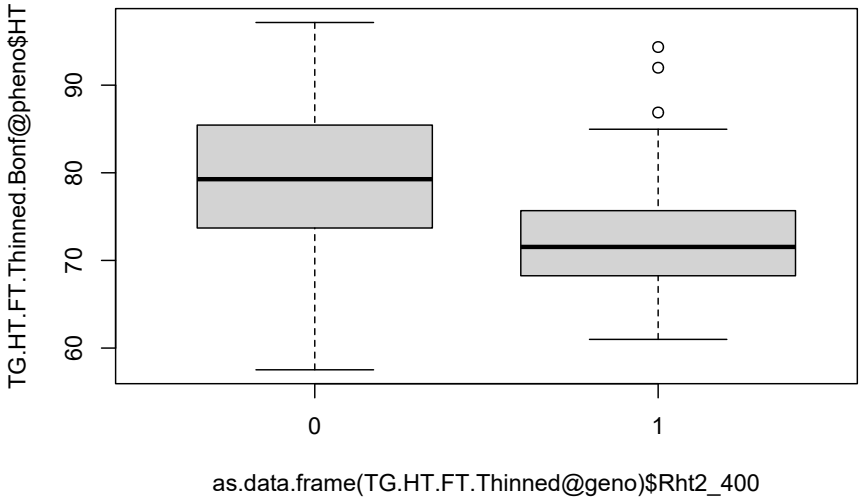
"Score" is the −log10(p-value) from the analyses, to be compared with the Threshold – which is also a −log10(p-value). "Ref and "Alt" give the reference allele and the alternative allele. "Effect" gives the gene effect for the alternative allele. In the example, The alternative allele (1) for PpdD1_297 has an effect of -4.81 on flowering time. (i.e. it reduces flowering time by nearly 5 days). We can confirm this by extracting and plotting individual trait – marker combinations:

```
boxplot(TG.HT.FT.Thinned.Bonf@pheno$FT~as.data.frame(TG.HT.FT.Thinned@geno)$PpdD1_297,varwidth=T)
```



Similarly, the top hit for height is the unmapped Rht2 gene (one of the 'green revolution' dwarfing loci), where the reduced height allele has an effect size of about 5.5 cm relative to the reference allele.

```
boxplot(TG.HT.FT.Thinned.Bonf@pheno$HT~as.data.frame(TG.HT.FT.Thinned@geno)$Rht2_400,varwidth=T)
```



2

a). Compare the results for "HT" and "FT" between the 'Bonferroni' and 'FDR', and, if possible, 'permutation' thresholds. What do you conclude?

b). Rerun the analysis using the Non-thinned data. What differences might you expect to see? What do you observe and what are the implications?

We will now run the analysis unadjusted for kinship. The unadjusted analysis is simply linear regression, or a t-test, on each marker in turn. We could do this by putting `lm()` or `t.test()` inside a loop in R. If you have time, this is a good exercise to try. However, we can get the same result directly from GWASpoly by analysing the data with a kinship matrix which has '1' down the leading diagonal and '0' everywhere else – all the lines are unrelated. Strictly, it isn't that the lines are unrelated, it is that they have equal relationships: we should get the same answer if you used, say, 0.25, for the off-diagonal elements (another good exercise to try).

First take a copy of the data and then create the diagonal matrix:

```
TGdiag<-TGanalysisThinned
#There is a command which will create diagonal matrices in R: diag().
TGdiag@K <- diag(1,length(rownames(TGanalysisThinned@geno)))
```

We've also picked up the required dimensions of the kinship matrix using `length(rownames (....))`. The new kinship matrix must also have row names and column names which match the names of the original matrix:

```
rownames(TGdiag@K)<-(rownames(TGanalysisThinned@geno))
colnames(TGdiag@K)<-(rownames(TGanalysisThinned@geno))
```

Check all looks ok:

```
TGdiag@K[1:5,1:5]
```

```
##          AARDEN AARDVARK ABELE ABO ACCESS
## AARDEN        1        0     0   0      0
## AARDVARK      0        1     0   0      0
## ABELE         0        0     1   0      0
## ABO           0        0     0   1      0
## ACCESS        0        0     0   0      1
```

And run the analyses, just as before:

```
TGdiag <-GWASpoly(TGdiag,models="additive")
TGdiag.fdr <-set.threshold(TGdiag,method="FDR",level=0.05)
get.QTL(TGdiag.fdr)
manhattan.plot(TGdiag.fdr,trait=c("HT"),model="additive")
```

3

How do the results compare with the results of the mixed model which we ran previously? Are there many significant results arising from not considering kinship? Was the mixed model analysis necessary?

9.4.3 Analysis with covariates

For height, in the unadjusted results, you should find Rht2 is the most significant marker and has the largest effect. It is the most important dwarfing gene employed in Europe, so this is not surprising. In the adjusted results, you will still find Rht2 on top, but the p-value is much reduced, as is the estimate of its effect. Height is a major contributor to population structure in wheat: German varieties tend to rely less on dwarfing genes for example and so are taller. Adjusting for kinship has the effect of leaving less variation to be accounted for by genuine marker relationships.

Perhaps Rht2 is dominating the results. If we include Rht2 as a covariate in the analysis, will other QTL for reduced height be revealed? For this we need to include Rht2 in the analysis as a covariate. Recall that the phenotypic data input included some additional columns, to be used as covariates. These have been stored in a slot: `@fixed`:

```
summary(TGanalysisThinned@fixed)
```

```
##     Rht2_400        PpdD1_297           YEAR        COUNTRY
##  Min.   :0.0000   Min.   :0.0000   Min.   :1946   Length:376
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:1988   Class :character
##  Median :1.0000   Median :0.0000   Median :1997   Mode  :character
##  Mean   :0.5455   Mean   :0.2253   Mean   :1994
##  3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:2003
##  Max.   :1.0000   Max.   :1.0000   Max.   :2007
##       FRA              DEU
##  Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000
##  Median :1.0000   Median :0.0000
##  Mean   :0.5656   Mean   :0.2385
##  3rd Qu.:1.0000   3rd Qu.:0.0000
##  Max.   :1.0000   Max.   :1.0000
```

I've included Rht2 as one of these covariates. To select other SNPs as covariates, I expect you could add or edit `TGanalysisThinned@fixed` to include the desired SNPs, but I've not tried this myself. It's another exercise in R and GWAS for you to try! To add a covariate to the mixed model, first provide their names and their data type. The covariates can be numeric or factors, such as country of origin.

```
Rht2<- set.params(fixed=c("Rht2_400"),fixed.type=c("numeric"))
```

We could include multiple covariates in the analysis (by using `c(cov1,cov2...)`. In the interests of speed, we'll just use Rht2:

```
TG.HT.Rht2<-GWASpoly(TGanalysisThinned,models="additive",trait=c("HT"),params=Rht2)
TG.HT.Rht2 <-set.threshold(TG.HT.Rht2,method="FDR",level=0.05)
```

Now compare the results on your own PC:

```
get.QTL(TG.HT.Rht2,traits=c("HT"))
get.QTL(TG.HT.FT.Thinned.fdr,traits=c("HT"))
```

4

What is the main difference of including a covariate? Do any hits vanish? If so, why do you think this might be? Do new hits appear? If so, why do you think they were not previously detected? (For those of you who work on wheat, PpdD1 and Rht1 should be familiar, in particular what do you think is going on with Rht1?

9.4.4 Accounting for population structure

We could use as covariates the population membership as coefficients estimated in the program STRUCTURE. To do this, we would have to run STRUCTURE and import these as covariates with the rest of the trait and covariate data. Including these together with kinship coefficients is often called the "Q + K" analysis. Alternatively, as we have seen, for crop germplasm "populations" such as this one, principal coordinate analysis may be a more appropriate choice for estimating population structure. For this reason, the principal components from a PCoA may be the best choice to include as covariates. GWASpoly has options to do this automatically. Here we use the first 10 eigenvectors:

```
params <- set.params(n.PC=10)
TG.PCO<-GWASpoly(TGanalysisThinned,models="additive",traits=c("HT","FT"),params=params)
TG.PCO<-set.threshold(TG.PCO,method="Bonferroni",level=0.05)
get.QTL(TG.PCO)
get.QTL(TG.HT.FT.Thinned.fdr)
```

Then inspect he results:

```
qq.plot(TG.PCO,trait="FT",model="additive")
qq.plot(TG.HT.FT.Thinned,trait="FT",model="additive")
```

5

Does this make any difference to the results (note the subtle difference in the qq plots as well as the qtls detected)? Do you think the results might differ for a different kind of mapping population? Do you think it is a problem that we are using the kinship matrix twice?

Rather than estimate population structure from the kinship matrix, we can analyse the data with country of varietal origin as a factor. To do this,

```
country<-set.params(fixed = "COUNTRY",fixed.type = "factor")
TG.country<-GWASpoly(TGanalysisThinned,models="additive",traits=c("HT","FT"), params=country)
TG.country<-set.threshold(TG.country,method="FDR",level=0.05)
get.QTL(TG.country)
get.QTL(TG.HT.FT.Thinned.fdr)
```

For covariates to be included, there must be no missing data. A pseudo country has been added here for the three missing values. A better way of dealing with the unknown factor levels would be to include two columns for the three countries coded as FRA (1) and not-FRA (0) in the first column and DEU and notDEU in the second. If a country is not-FRA and not-DEU then it must be GBR: we have three countries but only two degrees of freedom so the two columns alone are adequate. Then the missing values in each column could be coded as 1/3: the average population membership over the three countries. This is an example of the equivalence of regression and the analysis of factors with multiple levels in the analysis of variance.

6

Did using countries as a covariate make any difference to the results?

9.4.5 Further analysis

There are other things that can be done with the `set.params()` command: the minor allele frequency for inclusion in the analysis can be set. The default is 0.05. Also, the approximate method of carrying out the mixed analysis can be turned off. See `help(set.params)` for details.

We have seen that there remains some art in carrying out association analyses: what kinship matrix to use, what covariates to include and so on. The best approach is probably to simulate traits from the marker data for your data set and test analysis methods on those. In simulations on UK barley and UK wheat data we concluded that the best all-round method is the mixed model, with a kinship matrix similar to the one used

here and no additional covariates to account for population structure. But this wasn't always best for all traits and datasets. If one has time, it is worth experimenting. And of course simulations are always needed to estimate the power of your analysis, and hence the data collection requirements, whichever approach you take.

7

How would we run simulations to estimate the power of different analysis approaches?

Bibliography

Adler, J. (2010). *R in a nutshell*. O'Reilly, Sebastopol, CA, 1st ed edition. OCLC: ocn432987461.

Bates, D., Maechler, M., Bolker, B., and Walker, S. (2020). *lme4: Linear Mixed-Effects Models using Eigen and S4*. R package version 1.1-26.

Bentley, A. R., Scutari, M., Gosman, N., Faure, S., Bedford, F., Howell, P., Cockram, J., Rose, G. A., Barber, T., Irigoyen, J., Horsnell, R., Pumfrey, C., Winnie, E., Schacht, J., Beauchêne, K., Praud, S., Greenland, A., Balding, D., and Mackay, I. J. (2014). Applying association mapping and genomic selection to the dissection of key traits in elite European wheat. *Theoretical and Applied Genetics*, 127(12):2619–2633.

Dalgaard, P. (2008). *Introductory statistics with R*. Statistics and computing. Springer, New York, 2nd ed edition. OCLC: ocn253729714.

de Mendiburu, F. (2020). *agricolae: Statistical Procedures for Agricultural Research*. R package version 1.3-3.

Earl, D. and vonHoldt, B. (2012). Structure harvester: A website and program for visualizing structure output and implementing the evanno method. *Conservation Genetics Resources*, 4(2):359–361. Funding Information: Acknowledgments We thank J Goudet for improving our implementation of the Evanno method (personal communications). We thank C Taylor at University of California at Los Angeles for hosting the website. This work was supported by the National Cancer Institute grants 1U24CA143858-01 and 5R21CA135937.

Edmondson., R. N. (2021). *blocksdesign: Nested and Crossed Block Designs for Factorial and Unstructured Treatment Sets*. R package version 4.7.

Evanno, G., Regnaut, S., and Goudet, J. (2005). Detecting the number of clusters of individuals using the software structure: a simulation study. *Molecular Ecology*, 14(8):2611–2620.

Gardner, K. A., Wittern, L. M., and Mackay, I. J. (2016). A highly recombined, high-density, eight-founder wheat magic map reveals extensive segregation distortion and genomic locations of introgression segments. *Plant Biotechnology Journal*, 14(6):1406–1417.

Goudet, J. and Jombart, T. (2020). *hierfstat: Estimation and Tests of Hierarchical F-Statistics*. R package version 0.5-7.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning*, volume 103 of *Springer Texts in Statistics*. Springer New York, New York, NY.

Jombart, T. (2008). adegenet: a r package for the multivariate analysis of genetic markers. *Bioinformatics*, 24:1403–1405.

Kabacoff, R. (2011). *R in action: data analysis and graphics with R*. Manning, Shelter Island, NY. OCLC: ocn609544307.

Kang, H. M., Sul, J. H., Service, S. K., Zaitlen, N. A., Kong, S.-y., Freimer, N. B., Sabatti, C., and Eskin, E. (2010). Variance component model to account for sample structure in genome-wide association studies. *Nature Genetics*, 42(4):348–354.

Lenth, R. V. (2020). *emmeans: Estimated Marginal Means, aka Least-Squares Means*. R package version 1.5.3.

Luo, D., Ganesh, S., and Koolaard, J. (2020). *predictmeans: Calculate Predicted Means for Linear Models*. R package version 1.0.4.

Mead, R. (1994). *The design of experiments: statistical principles for practical applications*. Cambridge Univ. Press, Cambridge, 1. paperback ed. (with corr.), reprinted edition. OCLC: 258038533.

Paradis, E., Jombart, T., Kamvar, Z. N., Knaus, B., Schliep, K., Potts, A., and Winter, D. (2020). *pegas: Population and Evolutionary Genetics Analysis System*. R package version 0.14.

Patterson, H. D. and Williams, E. R. (1976). A new class of resolvable incomplete block designs. *Biometrika*, 63(1):83–92.

Rodríguez-Álvarez, M. X., Boer, M. P., van Eeuwijk, F. A., and Eilers, P. H. (2017). Correcting for spatial heterogeneity in plant breeding experiments with p-splines. *Spatial Statistics*, 23:52 – 71.

Rodríguez-Álvarez, M. X., Boer, M. P., van Eeuwijk, F. A., and Eilers, P. H. C. (2016). Spatial models for field trials.

Rutkoski, J. E., Poland, J., Jannink, J.-L., and Sorrells, M. E. (2013). Imputation of Unordered Markers and the Impact on Genomic Selection Accuracy. *G3 Genes|Genomes|Genetics*, 3(3):427–439.

Speed, D., Hemani, G., Johnson, M., and Balding, D. (2012). Improved heritability estimation from genome-wide snps. *The American Journal of Human Genetics*, 91(6):1011–1021.

VanRaden, P. (2008). Efficient Methods to Compute Genomic Predictions. *Journal of Dairy Science*, 91(11):4414–4423.

Velazco, J. G., Rodríguez-Álvarez, M. X., Boer, M. P., Jordan, D. R., Eilers, P. H. C., Malosetti, M., and van Eeuwijk, F. A. (2017). Modelling spatial trends in sorghum breeding field trials using a two-dimensional P-spline mixed model. *Theoretical and Applied Genetics*, 130(7):1375–1392.

Weber, A., Clark, R. M., Vaughn, L., de Jesús Sánchez-Gonzalez, J., Yu, J., Yandell, B. S., Bradbury, P., and Doebley, J. (2007). Major Regulatory Genes in Maize Contribute to Standing Variation in Teosinte (*Zea mays* ssp. *parviglumis*). *Genetics*, 177(4):2349–2359.

Wickham, H. and Grolemund, G. (2016). *R for data science: import, tidy, transform, visualize, and model data*. O'Reilly, Sebastopol, CA, first edition edition. OCLC: ocn968213225.

Wright, K. (2020). *desplot: Plotting Field Plans for Agricultural Experiments*. R package version 1.8.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2020). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.21.

Zhang, Z., Ersoz, E., Lai, C.-Q., Todhunter, R. J., Tiwari, H. K., Gore, M. A., Bradbury, P. J., Yu, J., Arnett, D. K., Ordovas, J. M., and Buckler, E. S. (2010). Mixed linear model approach adapted for genome-wide association studies. *Nature Genetics*, 42(4):355–360.