dartR Workshop - CBA

Bernd Gruber & Arthur Georges

2018-11-28

Contents

Foreword

What it this tutorial about?

In this workshop we provide an overview on the use of a recently developed R-package (dartR) that aims to integrate as many as possible ways to analyse SNP data sets. [1]. You should also have a copy of the DartBook next to you so you can refer to a more detailed description of the functions and the package.

Below here you will find exercises that should get you familiar with the package. There is some explanatory text, but mainly you will be asked to apply your gained knowledge by solving the tasks outlined in those boxes:

Task

1. Whenever you see such a box, the idea is that you try to run and answer the questions.

Feel free to use the provided sample data or use you own data.

[1] A general overview on the package (though quite a few new function have been added can be found here: Gruber et al. 2018: https://onlinelibrary.wiley.com/doi/full/ 10.1111/1755-0998.12745

0. Installing dartR

Please refer to the manual or the github page in case you have not yet installed the package dartR on your computer. `https://github.com/green-striped-gecko/dartR`

The code to do so is:

```
install.packages("devtools")
library(devtools)
install.packages("BiocManager")
BiocManager::install(c("SNPRelate", "qvalue"))
install_github("green-striped-gecko/dartR")
```

Once installed the command below should run without error: (warnings are most often okay). Be aware we noticed that you cannot use an R version of <3.2 on Macs if you want to

```
library(dartR)
```

You should have version `1.1.6` installed otherwise some code might not run, because the functions are missing.

```
packageVersion("dartR")
```

```
## [1] '1.1.6'
```

1. Preparing data sets

1.1. Loading data sets into R (DArT format to genlight)

SNP data can be read into a genlight object using gl.read.dart(). This function intelligently interrogates the input csv file to determine which can of format you got from DArt (one row, two row format).

For a test we will use the inbuild files. They are stored in your package under:

```
fp <- file.path(system.file(package = "dartR"),
    "extdata")
fp
```

```
## [1] "C:/Program Files/R/library/dartR/extdata"
```

```
dir(fp)
```

```
## [1] "landscape.sim.rdata"
## [2] "platy.csv"
## [3] "testset_metadata.csv"
## [4] "testset_pop_recode.csv"
## [5] "testset_SNPs_2Row.csv"
```

Task

1. Explore the file: **testset_SNPs_2Row.csv** by opening it into Excel, Calc (or similar).
2. How many loci and how many samples are stored there?
3. Is there sequence information stored? Under which header (the default should be TrimmedSequence)?
4. Look at the other metrics provided and try to make sense of them.

You can load the data and convert it into a genlight object via the following code:

```
# create the path to the file
fn <- file.path(fp, "testset_SNPs_2Row.csv")
# read the data and store it in object gl
gl <- gl.read.dart(filename = fn, probar = F)
```

```
## Topskip not provided. Try to guess topskip...
## Set topskip to  3 . Trying to proceed...
```

```
## Trying to determine if one row or two row format...
## Found  2  row(s) format. Proceed...
## Added the following covmetrics:
## AlleleID CloneID AlleleSequence SNP SnpPosition CallRate OneRatioRef OneRatioSnp FreqHomRef FreqHomSnp Fre
## Number of rows per Clone. Should be only  2 s: 2
##  Recognised: 250 individuals and 255  SNPs in a 2 row format using C:/Program Files/R/library/dartR/extdat
## Start conversion....
## Format is 2 rows.
## Please note conversion of bigger data sets will take some time!
## Once finished, we recommend to save the object using save(object, file="object.rdata")
```

Now all the data from this file is stored into a genlight object [2]. In brief the genlight format allows to store large data sets very efficiently (compacted) and at the same time allows to interrogate the data set very conviniently via accessors.

[2] for a detailed description of this format please refer to the dartBook.

We can inspect if the file has been read correctly by typing the name of the object: `gl`.

```
gl
```

```
##  /// GENLIGHT OBJECT /////////
##
##  // 250 genotypes,  255 binary SNPs, size: 624.8 Kb
##  7868 (12.34 %) missing data
##
##  // Basic content
##    @gen: list of 250 SNPbin
##    @ploidy: ploidy of each individual  (range: 2-2)
##
##  // Optional content
##    @ind.names:  250 individual labels
##    @loc.names:  255 locus labels
##    @loc.all:  255 alleles
##    @position: integer storing positions of the SNPs
##    @other: a list containing: loc.metrics
```

A bit of R background. This `gl` object is a so called S4 object, which is R's attempt to implement object oriented programming. The main message is that you need to use the '@' sign to access its slots (sub-components) [if no accessor function exists] and for sub-sub components, it is the '$' sign.

Let's have a look at the other slots:

```
gl
```

```
##  /// GENLIGHT OBJECT /////////
##
##  // 250 genotypes,  255 binary SNPs, size: 624.8 Kb
```

```
##  7868 (12.34 %) missing data
##
##  // Basic content
##    @gen: list of 250 SNPbin
##    @ploidy: ploidy of each individual  (range: 2-2)
##
##  // Optional content
##    @ind.names:  250 individual labels
##    @loc.names:  255 locus labels
##    @loc.all:  255 alleles
##    @position: integer storing positions of the SNPs
##    @other: a list containing: loc.metrics
```

Task

There are more slots in the genlight object (type gl) For some slots accessors exists and for some there are no accessors. For example: position(), indNames(), locNames(), ploidy() are accessors, but there are none for `\@loc.all, \@other, \@gen`.

1. Inspect all the content in those slots to get an overview on the data set. Here it might be helpful to employ functions such as `summary(), table()` or to visualise `barplot(), hist()`.

Now you can try also to load your own data set into R (if it is in DArT format)

You might wonder where is the SNP data actually stored and have explored the @gen slot, which contains that information, but still in a not easy accessible format (class SNPbin). A very important additional function for genlight objects is the `as.matrix()` function. It converts the SNP information in a matrix of individuals/samples across the rows and loci across the columns [3] The entries are either 0, 1, 2 or NA and represent the frequency of the second allele for that individual at that loci.[4].

[3] This is different to the format in the dart csv file!!!.
[4] Please refer to the workbook for a detailed description of this matrix

```
# Dimensions of the matrix ind x loc
dim(as.matrix(gl))
```

```
## [1] 250 255
```

```
# showing the first five individuals and the
# first 3 loci
as.matrix(gl[1:10, 1:3])
```

```
##          100049687-12-A/G 100049698-16-C/T
## AA010915                2               NA
```

```
## UC_00126                2               NA
## AA032760               NA               NA
## AA013214                2               NA
## AA011723                2               NA
## AA012411                2               NA
## AA019237                2               NA
## AA019238                2               NA
## AA019239                2               NA
## AA019235                2               NA
##          100049728-23-T/G
## AA010915                0
## UC_00126                0
## AA032760                0
## AA013214                0
## AA011723                0
## AA012411                0
## AA019237                0
## AA019238                0
## AA019239                0
## AA019235                0
```

Hint

As you have seen above you can subset your data using the indexing function [,].

```
# first individual, loci 1 to 5.
as.matrix(gl[1, 1:5])
```

```
##          100049687-12-A/G 100049698-16-C/T
## AA010915                2               NA
##          100049728-23-T/G 100049805-56-A/T
## AA010915                0                0
##          100049816-51-C/T
## AA010915                0
```

Task

1. Use the indexing function to inspect the loci the individuals 7 to 9, for loci 1 to 4.
2. Create a table on the genotypes for the first 10 individuals. How many heterozygote loci are there?

Explore loci metrics

If you have inspected the provided CSV file or your own data you noticed that dart provides additional data on the quality and contenct of loci. Those metrics can be used to filter loci by quality (CallRate) or information content (minor allele frequence). The information is also stored in the genlight object under the slot `\@other$loc.metrics`.

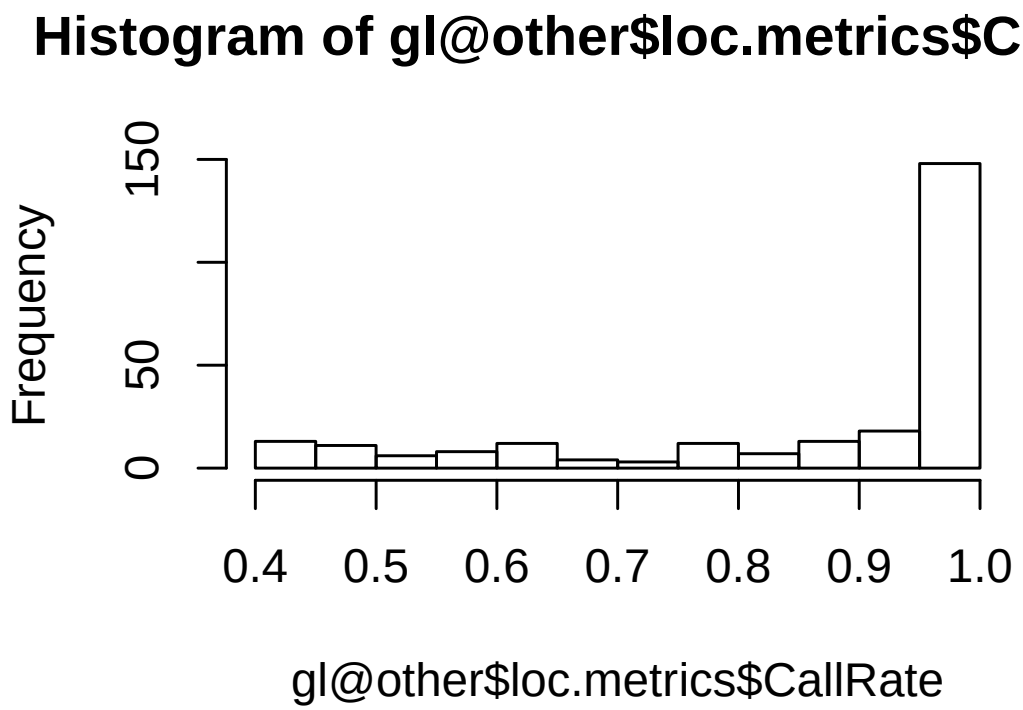
`\@other$loc.metrics` is a data.frame (a table in R), that has a **row** for each SNP loci. You can explore those entries via: [5]

[5] A complete overview of each of the loci metrics is provided in the manual.

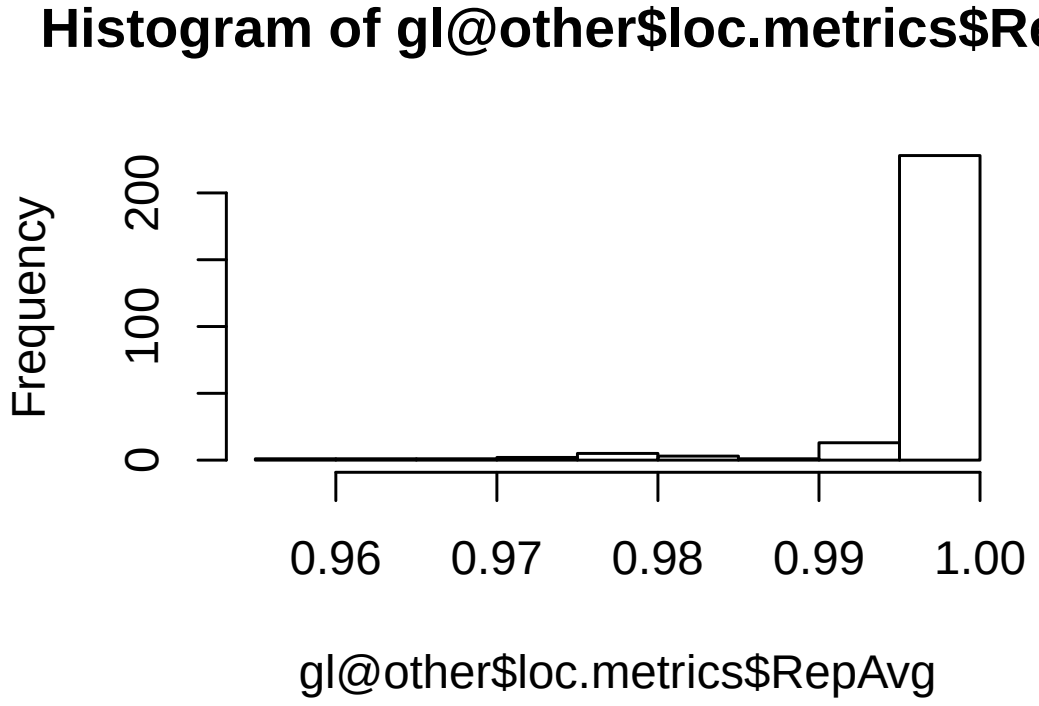
```
# names of loc.metrics
names(gl@other$loc.metrics)
```

```
##  [1] "AlleleID"       "CloneID"
##  [3] "AlleleSequence" "SNP"
##  [5] "SnpPosition"    "CallRate"
##  [7] "OneRatioRef"    "OneRatioSnp"
##  [9] "FreqHomRef"     "FreqHomSnp"
## [11] "FreqHets"       "PICRef"
## [13] "PICSnp"         "AvgPIC"
## [15] "AvgCountRef"    "AvgCountSnp"
## [17] "RepAvg"         "clone"
## [19] "uid"
```

```
hist(gl@other$loc.metrics$CallRate)
```




```
hist(gl@other$loc.metrics$RepAvg)
```



Task

1. Create a histogram over the loci metric RepAvg
2. The position slot has information on the position of the SNP according to the provided sequences (69 bases). What is the distribution of the position over all loci?

Add ind.metrics

As you might noticed there is no information on the individuals/samples provided in the file on the SNP loci. For example there is the accessor `nPop()` which returns the number of populations in your data set.

```
nPop(gl)
```

```
## [1] 0
```

As no information was provided yet, we need to learn how to do so. This information has to be provided with a second file that has - as the most important column - the id of the individuals that needs to match **exacttly** those in the SNP file you received from DArT. As an example there is a data set provided within the dartR package **testset_metadata.csv**:

```
fp
```

```
## [1] "C:/Program Files/R/library/dartR/extdata"
```

```
dir(fp)
```

```
## [1] "landscape.sim.rdata"
## [2] "platy.csv"
## [3] "testset_metadata.csv"
## [4] "testset_pop_recode.csv"
## [5] "testset_SNPs_2Row.csv"
```

Task

Inspect the file **testset_metadata.csv** by opening it into Excel, Calc or similar.

1. How many rows do you expect to be there?
2. Note and remember the header names (spelling) of the first four columns.

There are four important header identifier that have a special function:

header	meaning
id	matching id to link information of loci and individuals [compulsory]. **Samples that could not be matched are dropped!!!**
pop	information on population assignment of individuals (used by many function as the default hierachy) [optional]
lat	latitude of a sample in geographic coordinates (WGS84, GDA94) [optional]
lon	longitude atitude of a sample in geographic coordinates (WGS84, GDA94) [optional]
other	can be provided and are copied but are not used in other functions (except sex, by `gl.sexlinkage`).

We can load and combine the information of both files into a single genlight object using the `gl.read.dart()` function.

```
# filename of the Dart SNP file
fn <- file.path(fp, "testset_SNPs_2Row.csv")
# filename of the file on individuals/samples
ifn <- file.path(fp, "testset_metadata.csv")

gl <- gl.read.dart(filename = fn, ind.metafile = ifn,
```

```
    probar = F)
```

```
## Topskip not provided. Try to guess topskip...
## Set topskip to  3 . Trying to proceed...
## Trying to determine if one row or two row format...
## Found  2  row(s) format. Proceed...
## Added the following covmetrics:
## AlleleID CloneID AlleleSequence SNP SnpPosition CallRate OneRatioRef OneRatioSnp FreqHomRef FreqHomSnp Fre
## Number of rows per Clone. Should be only  2 s: 2
##  Recognised: 250 individuals and 255  SNPs in a 2 row format using C:/Program Files/R/library/dartR/extdat
## Start conversion....
## Format is 2 rows.
## Please note conversion of bigger data sets will take some time!
## Once finished, we recommend to save the object using save(object, file="object.rdata")
## Try to add covariate file: C:/Program Files/R/library/dartR/extdata/testset_metadata.csv .
## Ids of covariate file (at least a subset of) are matching!
## Found  250 matching ids out of 250 ids provided in the covariate file. Subsetting snps now!.
##  Added pop factor.
## Added latlon data.
## Added  id  to the other$ind.metrics slot.
##  Added  pop  to the other$ind.metrics slot.
##  Added  lat  to the other$ind.metrics slot.
##  Added  lon  to the other$ind.metrics slot.
##  Added  sex  to the other$ind.metrics slot.
##  Added  maturity  to the other$ind.metrics slot.
```

Task

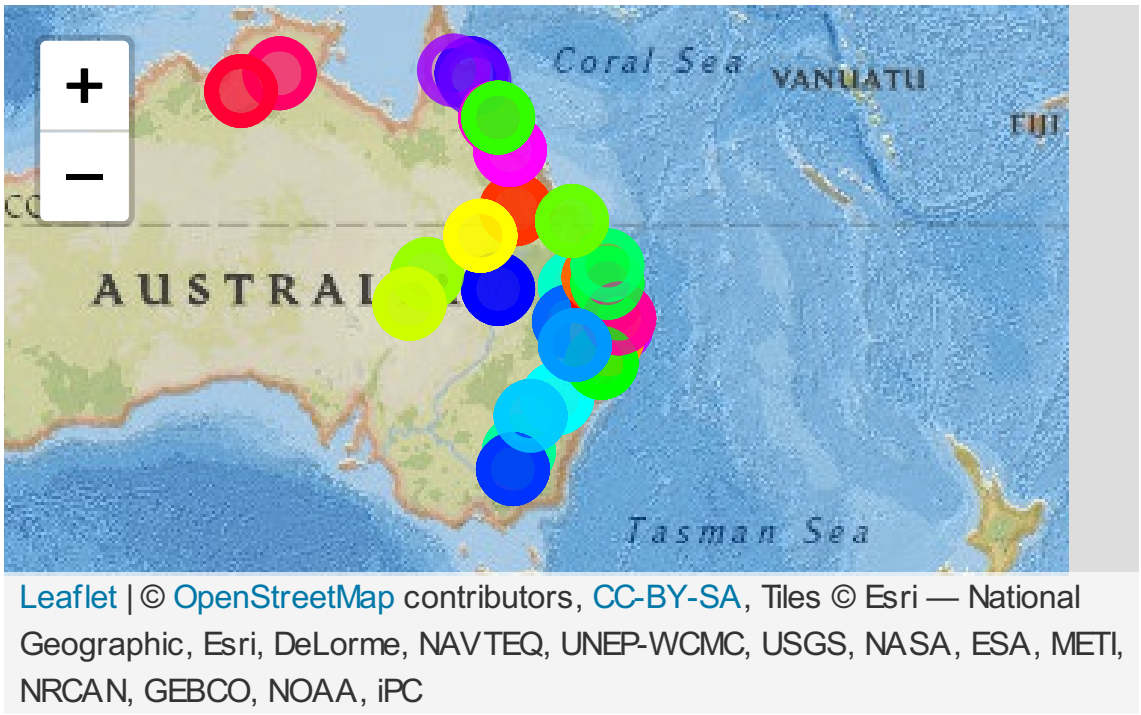
Go through the output in the console. It provides important information if the data have been combined correctly.

1. Were all id's of the samples matched?
2. Find the slot where a copy of the information on individuals is stored?
3. Explore the @pop slot (accessors `nPop()`, `pop()`)
4. Create a table on the number of individuals per population to find out how many individuals per population were sampled.
5. What is the sex ratio of the sampled individuals?

Now we have a "complete" genlight object with data on loci and individuals stored in it. Feel free to explore the example data set or try to import your own data and explore it.

You have finished the first part of the tutorial. To relax try the command below (it uses the lat lon information of the samples in the @other$latlong slot):

```
gl.map.interactive(gl)
```



1.2 How to import other formats to genlight**

Hint

This section is an example how you can load your data and convert it into a genlight object if you do not have data provided by DArT. It demonstrates the principle how that can be achieved, but your data set might be in a very different format. Please ask the tutors in case you got stuck and do not know how to load your data set.

There are many ways to load data sets of other types into R. The main aim you want to achieve is, to resemble the structure of the genlight object as closely as possible, as it allows you to work with dartR and use most of its functions.

The idea is "simple", we first create a genlight object that stores the SNP data and then we add the relevant loci and individual metrics to the required slots.

Below is a table from the publication of the package, which suggests several possible ways to import `data:%5BGruber` et al. 2018: `https://onlinelibrary.wiley.com/doi/full/` 10.1111/1755-0998.12745](`https://onlinelibrary.wiley.com/doi/full/10.1111/1755-0998.12745`)].

You can see there are some options available and most of them use the gi2gl function, hence we will follow this idea and use the read.genetable version.

Again there is an example data set provided in the package, called platy.csv in the package folder.

TABLE 1 Possible import pathways to convert SNP data to genlight format

Import path	Package	Pathway[a]	Description
gl.read.dart	DARTR	–	Based on DaRT data [with optional meta data for individuals]
read.loci	PEGAS	loci2genind, gi2gl	Data set are provided as a csv text file (?read.loci)
read.vcfR	PEGAS	vcfR2genlight	vcf text file (vcfR package)
read.fstat	ADEGENET	gi2gl	Fstat format (version 2.9.3) by Jerome Goudet
read.genetix	ADEGENET	gi2gl	Format Belkhir K., Borsa P., Chikhi L., Raufaste N. & Bonhomme F. (1996–2004) GE
read.structure	ADEGENET	gi2gl	Structure format of Pritchard, J.; Stephens, M. & Donnelly, P. (2000)
read.PLINK	ADEGENET	–	Data provided in PLINK format
fasta2genlight	ADEGENET	–	Extracts SNPs data from fasta format (?ADEGENET)
read.genetable	POPGENREPORT	gi2gl	csv text file based on df2genind Adamack and Gruber (2014) (?read.genetable)

[a]Pathway provides the order of functions needed to convert data to genlight, — indicates that the function directly converts to a genlight object

```
fp
```

```
## [1] "C:/Program Files/R/library/dartR/extdata"
```

```
dir(fp)
```

```
## [1] "landscape.sim.rdata"
## [2] "platy.csv"
## [3] "testset_metadata.csv"
## [4] "testset_pop_recode.csv"
## [5] "testset_SNPs_2Row.csv"
```

Task

Explore the file platy.csv using Excel, Calc etc.

The data set is a "mockup" data set of 13 samples of platypus in Tasmania and the SNPs are provided in the format A/A (meaning at this loci the individual was homozygot for A). The data set stores also information on populations, lat long and some additional information on sex (called group) and age.

To load this data set we will use the function `read.genetable` from package `PopGenReport` (an excellent package if you want to study STR markers (microsatellites) ;-) or only a few SNPs).

The function has some arguments that are explained in detail via its help pages `?read.gene.table`.

There you can specify the columns where the ids of individuals are, the pop, lat, long column and also how the loci are coded. [6]

[6] in one or two colums, what kind of seperator between loci and how missing values are coded.

```
# load the package
library(PopGenReport)
```

```
# create a genind object
platyfile <- file.path(fp, "platy.csv")
platy.gi <- read.genetable(platyfile, ind = 1,
    pop = 2, lat = 3, long = 4, other.min = 5,
    other.max = 6, oneColPerAll = FALSE, sep = "/")
```

You could explore the platy.gi object, but we want to have a genlight object hence we need to convert it using `gi2gl()`.

```
platy.gl <- gi2gl(platy.gi)
platy.gl
```

```
##  /// GENLIGHT OBJECT /////////
##
##  // 13 genotypes,  6 binary SNPs, size: 26.5 Kb
##  0 (0 %) missing data
##
##  // Basic content
##    @gen: list of 13 SNPbin
##    @ploidy: ploidy of each individual  (range: 2-2)
##
##  // Optional content
##    @ind.names:  13 individual labels
##    @loc.names:  6 locus labels
##    @pop: population of each individual (group size range: 4-5)
##    @other: a list containing: latlong  data
```

Task

1. You can now explore the data set `platy.gl`. As you can see some of the slots are filled in (e.g. `pop(), indNAmes()`), but most are empty.
2. How are populations defined?
3. Can you create an interactive map?

There is no slot called @loc.metrics or @ind.metrics, which are necessary for some functions. For example we have information on the individuals in the slot @other$data.

We can simply copy those slots in the right position via:

```
platy.gl@other$ind.metrics <- platy.gl@other$data
```

We do not have metrics on loci, but we we can create those via:

```
platy.gl <- gl.recalc.metrics(platy.gl)
```

```
## No loc.metrics found in gl@other, therefore it will be created to hold the loci metrics. Be aware that sor
## Starting gl.recalc.metrics: Recalculating locus metrics
## Starting utils.recalc.avgpic: Recalculating OneRatioRef, OneRatioSnp, PICRef, PICSnp and AvgPIC
## Completed utils.recalc.avgpic
##
## Starting utils.recalc.callrate: Recalculating CallRate
## Completed utils.recalc.callrate
##
## Starting gl.report.maf: Minimum Allele Frequency
## Starting gl.filter.monomorphs: Deleting monomorphic loci
##   Deleting monomorphic loci and loci with all NA scores
## Completed gl.filter.monomorphs
##
## Starting utils.recalc.freqhets: Recalculating frequency of heterozygotes
## Completed utils.recalc.freqhets
##
## Starting utils.recalc.freqhomref: Recalculating frequency of homozygotes, reference allele
## Completed utils.recalc.freqhomref
##
## Starting utils.recalc.freqhomref: Recalculating frequency of homozygotes, alternate allele
## Completed utils.recalc.freqhomsnp
##
## Completed gl.filter.maf
##
## Note: Locus metrics recalculated
## Completed gl.recalc.metrics
```

We now have a almost complete genlight object, hence we can use most of the functions from dartR. (see next section)

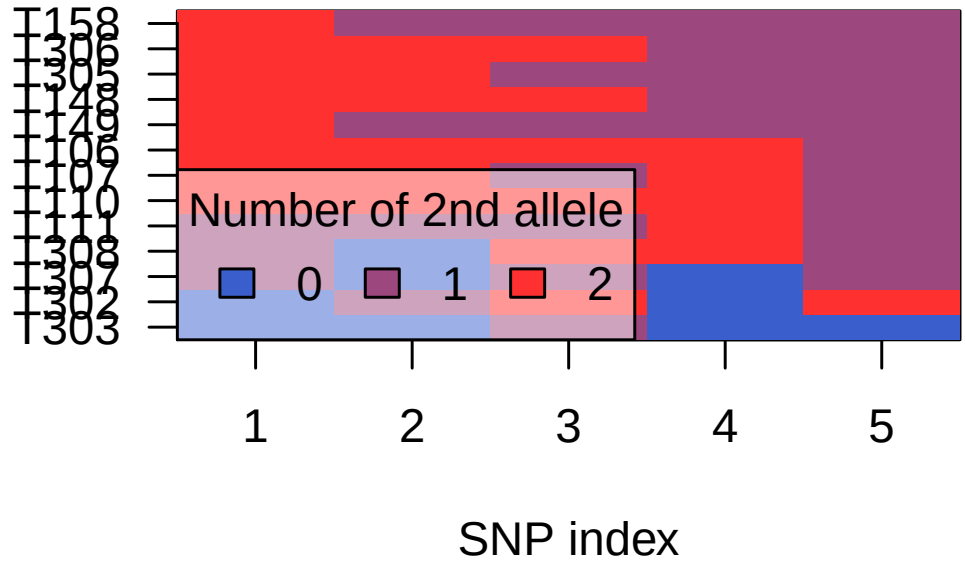
For example we can list the object as usual and also plot the individual over loci matrix.

```
platy.gl
```

```
##  /// GENLIGHT OBJECT /////////
##
##  // 13 genotypes,  5 binary SNPs, size: 32.7 Kb
##  0 (0 %) missing data
```

```
##
##  // Basic content
##    @gen: list of 13 SNPbin
##    @ploidy: ploidy of each individual  (range: 2-2)
##
##  // Optional content
##    @ind.names:  13 individual labels
##    @loc.names:  5 locus labels
##    @pop: population of each individual (group size range: 4-5)
##    @other: a list containing: latlong  data  ind.metrics  loc.metrics
```

```
gl.plot(platy.gl)
```



We do not have sequence information (and SNP position), hence the slot '`platy.gl@other$loc.metrics$TrimmedSequence` is empty so we obviously cannot create a fasta file from this object. In case you have this information, then simply add the TrimmedSequence and snp position information via:

```
platy.gl@other$loc.metrics$TrimmedSequence <- "character.vector of sequences"
position(platy.gl) <- "numeric vector of SNP positions (starting at zero)"
```

Task

1. If you have your own data set. Load it into R to convert it into a genlight object (with individual covariates in a second file). Explore your data set using the functions and framework explained above.
2. In case you did not bring your own data, explore the data set `foxes.gl` by just type foxes.gl into the console and apply the function learned above.

Questions you might tackle are:

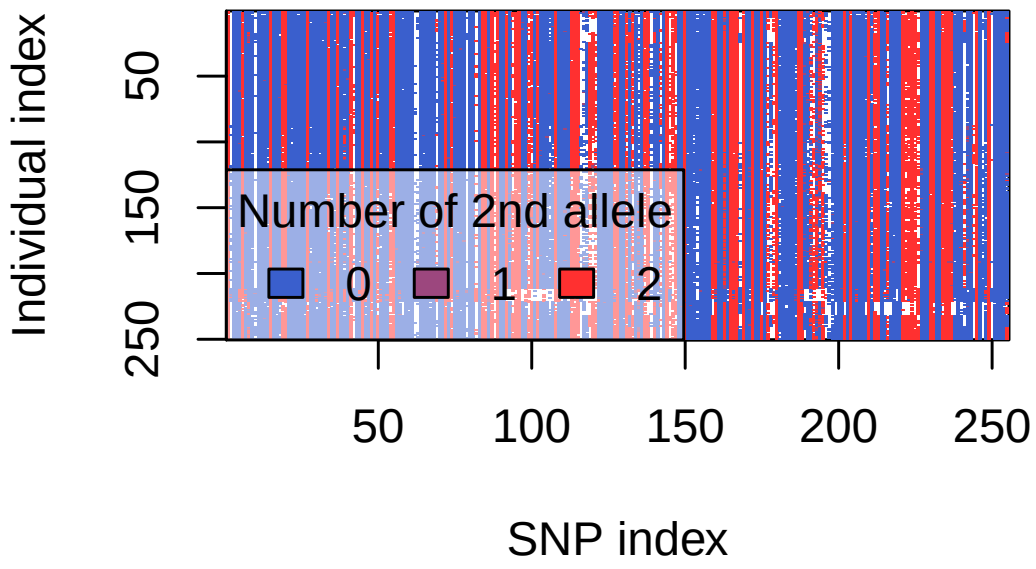
- How many loci are there?
- How many individuals/samples do I have?
- Do I have all the necessary loc.metrics (TrimmedSequence) for my analysis?
- What kind of individual metrics do I have?
- Plot the whole data set and investigate individuals with missing data.

2. The genlight object

2.1 Explore a genlight object

A nice way to get an overview on the object (=whole matrix) is to use:

```
plot(gl)
```




```
# or gl.plot(gl) # if not too many individuals
```

Remember that you can convert your SNP data with the `as.matrix()` function into a matrix of ind (rows) x SNPs (frequency of the second allele). This can help you to explore your data and calculate "basic" statistics.

For example:

```
colMeans(as.matrix(platy.gl), na.rm = T)/2
```

```
##     loci1     loci2     loci3     loci5
## 0.7307692 0.6153846 0.7307692 0.5769231
##     loci6
## 0.5000000
```

Calculates the allele frequency of the second allele for all loci.


Task

1. Plot your own data and explore the pattern. Are there any 'bands' indentified?

The individuals are plotted in the order they are stored in the genlight object. 2. Reorder the individuals alphabetically by population using the `order` function and plot it again.

2.2 Filter your data (by Quality/Ecology)

It is now time to think about your analysis and based on that, what kind of filters you want to apply previous to your analysis.

A range of filters are available for selecting individuals or loci on the basis of quality metrics.

function	explanation
gl.report.callrate()	Calculate and report the number of loci or individuals for which the call rate exceeds a range of thresholds.
gl.filter.callrate()	Calculate call rate (proportion with non-missing scores) for each locus or individual and remove those loci or individuals below a specified threshold.
gl.report.repavg()	Report the number of loci or individuals for which the reproducibility (averaged over the two allelic states) exceeds a range of thresholds.
gl.filter.repavg()	Remove those loci or individuals for which the reproducibility (averaged over the two allelic states) falls below a specified threshold.
gl.report.secondaries()	Report the number of sequence tags with multiple SNP loci, and the number of SNP loci that are part of or individuals for which the reproducibility (averaged over the two allelic states) exceeds a range of thresholds.
gl.filter.secondaries()	Remove all but one locus where there is more than one locus per sequence tag.
gl.report.monomorphs()	Report the number of monomorphic loci and the number of loci for which the scores are all missing (NA).
gl.filter.monomorphs()	Remove all monomorphic loci, including loci for which the scores are all missing (NA).
gl.report.hamming()	Report the distribution of pairwise Hamming distances between trimmed sequence tags.
gl.filter.hamming()	Filter loci by taking out one of a pair of loci with Hamming distances less than a threshold.

function	explanation
gl.filter.hwe	Filters departure of Hardy-Weinberg-Equilibrium for every loci per population or overall
gl.report.hwe	Reports departure of Hardy-Weinberg-Equilibrium for every loci per population or overall
gl.report.bases	Reports statistics on the frequncy of A,G,C,T (TrimmedSequence is needed)

Hint

Refer to the help pages of each function for details on the parameters `?nameoffunction`. Be aware that some of the function require additional parameters/thresholds to work properly. For example if you want to filter on CallRate you need to provide a threshold that is acceptable for you.

Task

Use your data set to report and filter on various quality meassures (RepAvg, Callrate). In case you do not have your own data use either: testset.gl foxes.gl

1. Filter by repeatability (`gl.filter.repavg` in dartR) (a meassurement of quality per loci)
2. Filter by monomorphic loci (`gl.filter.monomorphs`)(as they do not provide information for population structure and simply slow the analysis)
3. Filter by amount of missing data (`gl.filter.callrate, method="loc"`) per locus
4. Filter to remove all but one of multiple snps in the same fragment (`gl.filter.secondaries`)
5. Filter individuals by amount of missing data (`gl.filter.callrate, method="ind"`)

Additional filters to apply could be for excluding possible loci under selection (`gl.outflank`), checking loci for linkage disequilibrium (`gl.report.ld`) or filtering for loci out of Hardy-Weinberg-Equilibrium (`gl.filter.hwe`)

2.3 Subset/recode your data set

Very often you want to subset your data set (e.g. only looking at females or certain population). There are various ways to reassign individuals to populations, rename populations or individuals, delete populations or individuals after the data have been read in to a genlight object.

The initial population assignments via the metafile can be viewed via:

```
# number of populations
nPop(gl)
```

```
## [1] 30
```

```
# population names (#30 populations)
levels(pop(gl))
```

```
##  [1] "EmmacBrisWive"    "EmmacBurdMist"
##  [3] "EmmacBurnBara"    "EmmacClarJack"
##  [5] "EmmacClarYate"    "EmmacCoopAvin"
##  [7] "EmmacCoopCully"   "EmmacCoopEulb"
##  [9] "EmmacFitzAllig"   "EmmacJohnWari"
## [11] "EmmacMaclGeor"    "EmmacMaryBoru"
## [13] "EmmacMaryPetr"    "EmmacMDBBowm"
## [15] "EmmacMDBCond"     "EmmacMDBCudg"
## [17] "EmmacMDBForb"     "EmmacMDBGwyd"
## [19] "EmmacMDBMaci"     "EmmacMDBMurrMung"
## [21] "EmmacMDBSanf"     "EmmacNormJack"
## [23] "EmmacNormLeic"    "EmmacNormSalt"
## [25] "EmmacRichCasi"    "EmmacRoss"
## [27] "EmmacRussEube"    "EmmacTweeUki"
## [29] "EmsubRopeMata"    "EmvicVictJasp"
```

```
# table on individuals per population
table(pop(gl))
```

```
##
##    EmmacBrisWive    EmmacBurdMist
##               10               10
##    EmmacBurnBara    EmmacClarJack
##               11                5
##    EmmacClarYate    EmmacCoopAvin
##                5               10
##   EmmacCoopCully    EmmacCoopEulb
##               10               10
##   EmmacFitzAllig    EmmacJohnWari
##               10               10
##    EmmacMaclGeor    EmmacMaryBoru
##               11                6
##    EmmacMaryPetr     EmmacMDBBowm
##                4               10
##     EmmacMDBCond     EmmacMDBCudg
##               10               10
```

```
##     EmmacMDBForb     EmmacMDBGwyd
##               11                9
##     EmmacMDBMaci EmmacMDBMurrMung
##               10               10
##     EmmacMDBSanf    EmmacNormJack
##               10                6
##    EmmacNormLeic    EmmacNormSalt
##                1                1
##    EmmacRichCasi        EmmacRoss
##               10               10
##    EmmacRussEube     EmmacTweeUki
##               10               10
##    EmsubRopeMata    EmvicVictJasp
##                5                5
```

Task

1. Create a barplot on the number of individuals per population.

If you have only a few reassignments, the simplest way is to use one or more of the functions below

Hint

Try these out for yourself:
gl.keep.pop()
gl.drop.pop() gl.merge.pop() gl.merge.pop()
If only a few populations are involved, then the best option is to use the gl.drop.pop or gl.keep.pop functions.

Individual reassignment

The initial individual labels entered at the time of reading the data into the genlight object can be viewed via [7]:

[7] Please note only first 10 entries are shown here

```
# individual names
indNames(gl)
```

```
##  [1] "AA010915" "UC_00126" "AA032760"
##  [4] "AA013214" "AA011723" "AA012411"
##  [7] "AA019237" "AA019238" "AA019239"
## [10] "AA019235"
```

The individuals can be manipulated using


Task

gl.keep.ind() gl.drop.ind(gl)
Try these out for yourself, by listing the individuals using indNames() and then deleting a selected few.

Recode tables

Alternatively, reassignment and deletion of populations can be effected using a recode table, that is, a table stored as a csv file containing the old population assignments and the new population assignments as two columns. The quickest way to construct a recode table for an active genlight object is using

```
gl.make.recode.pop(gl, outfile = "new_pop_assignments.csv")
```


Hint

Please note we are using the `tempdir()` to read/write files to a location in all examples. Feel free to change that to your needs by just providing a path to the folder of your liking. Normally, this would be your working directory specified with setwd().

This will generate a csv file with two columns, the first containing the existing population assignments, and the second also containing those assignments ready for editing to achieve the reassignments. This editing is best done in Excel.

The population reassignments are then applied using:

```
glnew <- gl.recode.pop(gl, pop.recode = "new_pop_assignments.csv")
```

You can check that the new assignments have been applied with:


Task

Try using those commands in the R editor to create the comma-delimited recode file, edit in in Excel to remove the Emmac prefix from populations, then apply it using the above command from the R editor. Check your results.

Another way of population reassignment is to use:

```
glnew2 <- gl.edit.recode.pop(gl)
```

This command will bring up a window with a table showing the existing population assignments, with a second column available for editing. When the window is closed, the assignments will be applied. If you have optionally nominated a pop.recode file, a recode table will be written to file for future use.

Again, you can check that the new assignments have been applied with `levels(pop(gl))`.

Deleting populations with a recode table

You can delete selected populations from a genlight object using the "Delete" keyword in the population recode file. By reassigning populations to "Delete"", you are flagging them for deletion, and when the recode table is applied, individuals belonging to those populations will be deleted from the genlight object, and any resultant monomorphic loci will be removed.

Again, you can check that the new assignments have been applied and requested populations deleted with `levels(pop(gl))`.


Task

Try deleting some populations, say the outgroup populations (EmsubRopeMata and EmvicVictJasp) using `gl.edit.recode.pop()`from the R editor. Check your results for example using:
`table(pop(gl))`

Relabeling and deleting individuals with a recode table

Recall that the genlight object contains labels for each individual. It obtains these names from the csv datafile provided by DArT at the time of reading these data in. There may be reasons for changing these individual labels - there may have been a mistake, or new names need to be provided in preparation for analyses to be included in publications.

Individual recode tables are csv files (comma delimited text files) that can be used to rename individuals in the genlight object or for deleting individuals from the genlight object. These population assignments can be viewed using

```
# only first 10 entries are shown
indNames(gl)[1:10]
```

```
##  [1] "AA010915" "UC_00126" "AA032760"
##  [4] "AA013214" "AA011723" "AA012411"
##  [7] "AA019237" "AA019238" "AA019239"
## [10] "AA019235"
```

The quickest way to rename individuals is to construct a recode table for an active genlight object is using

```
gl.make.recode.ind(gl, outfile = "new_ind_assignments.csv")
```

This will generate a csv file with two columns, the first containing the existing individual names, and the second also containing those names ready for editing. This editing is best done in Excel.

The population reassignments are then applied using

```
glnew3 <- gl.recode.ind(gl, ind.recode = "new_ind_assignments.csv")
```

You can check that the new assignments have been applied with `indNames(gl)`

Another way of individual reassignment is to use

```
gl <- gl.edit.recode.ind(gl, ind.recode = "new_ind_assignments.csv")
```

This command will bring up a window with a table showing the existing individual labels, with a second column available for editing. When the window is closed, the renaming will be applied. If you have optionally nominated a ind.recode file, a recode table will be written to file for future use. Again, you can check that the new assignments have been applied with `indNames(gl)`.

Deleting individuals

You can delete selected individuals from a genlight object using the "Delete" keyword in the individual recode file. By renaming individuals to Delete, you are flagging them for deletion, and when the recode table is applied, those individuals will be deleted from the genlight object, and any resultant monomorphic loci will be removed.

Again, you can check that the new assignments have been applied and requested populations deleted with `indNames(gl)`.

Note that there are options for recalculating the relevant individual metadata, and for removing resultant monomorphic loci.

Recalculating locus metadata

When you delete individuals or populations, many of the locus metadata (such as Call Rate) are no longer correct. You can recalculate the locus metadata using the code:

`gl <- gl.recalc.metrics(gl)`

This is obviously important if you are drawing upon locus metadata in your calculations or filtering. The script will also optionally remove monomorphic loci.

Using R commands to manipulate the genlight object

With your data in a genlight object, you have the full capabilities of the adegenet package at your fingertips for subsetting your data, deleting SNP loci and individuals, selecting and deleting populations, and for recoding to amalgamate or split populations. Refer to the vignette: [Analysing genome-wide SNP data using adegenet] (`http://adegenet.r-forge.r-project.org/` files/tutorial-genomics.pdf). For example:

```
gl_new <- gl[gl$pop != "EmmacBrisWive", ]
```

removes all individuals of the population EmmacBrisWive from the data set.

Hint

Note that this manual approach will not recalculate the individual metadata nor will it remove resultant monomorphic loci. There are also some challenges with keeping the individual metadata matching the individual records (see below).

The basic idea is here that we can use the indexing function `[ ]` on the genlight object `gl` to subset our data set by individuals(=rows) and loci(=columns) in the same manner as we can subset a matrix in R.

For example:

```
glsub <- gl[1:7, 1:3]
glsub
```

```
##  /// GENLIGHT OBJECT /////////
##
##  // 7 genotypes,  3 binary SNPs, size: 196.3 Kb
##  8 (38.1 %) missing data
##
##  // Basic content
##    @gen: list of 7 SNPbin
##    @ploidy: ploidy of each individual  (range: 2-2)
##
##  // Optional content
##    @ind.names:  7 individual labels
##    @loc.names:  3 locus labels
##    @loc.all:  3 alleles
##    @position: integer storing positions of the SNPs
##    @pop: population of each individual (group size range: 1-1)
##    @other: a list containing: loc.metrics  latlong  ind.metrics
```

Subsets the data to the first seven individuals and the first three loci.

!!!Be aware that the accompanying meta data for individuals are subsetted, but the metadata for loci are not!!!!. So if you check the dimensions of the meta data of the subsetted data set via:

```
dim(glsub@other$ind.metrics)
```

```
## [1] 7 6
```

```
dim(glsub@other$loc.metrics)
```

```
## [1] 255  19
```

you see that the subsetting of the meta data for individuals worked fine (we have seven individuals (=rows)). But we have still all the metadata for all loci (in the rows for the (=107 instead of 3). This "bug/feature" is how the adegenet package implemented the genlight object.

For those not fully versed in R, there are the above {dartR} filters to achieve the same end and the advantage is that the filters do handle subsets of data correctly without any additional need to subset the meta data. The advantage of the R approach is that it is much more useful in case you want to script your analysis without intervention of a user when recoding your data set.

Task

Revision:

1. Interrogate your data to confirm the number of loci, individuals and population assignments.
2. Extract the SNP genotypes into a simple data matrix.
3. Visualize your data in a smear plot to assess structure and frequency of null alleles.
4. Delete individuals and populations, rename individuals and populations, merge populations, assign individuals to new populations.
5. Recalculate locus metrics after deleting individuals or populations.
6. Filter on call rate, repeatability, secondaries, minor allele frequency.
7. Filter out monomorphic loci.

3. Visualisation

3.1. PCoA

Please refer to the dartbook for details on the commands to be used and check the help pages.
`gl.pcoa, gl.pcoa.plot, gl.pcoa.plot.3d, gl.pcoa.scree`

Task

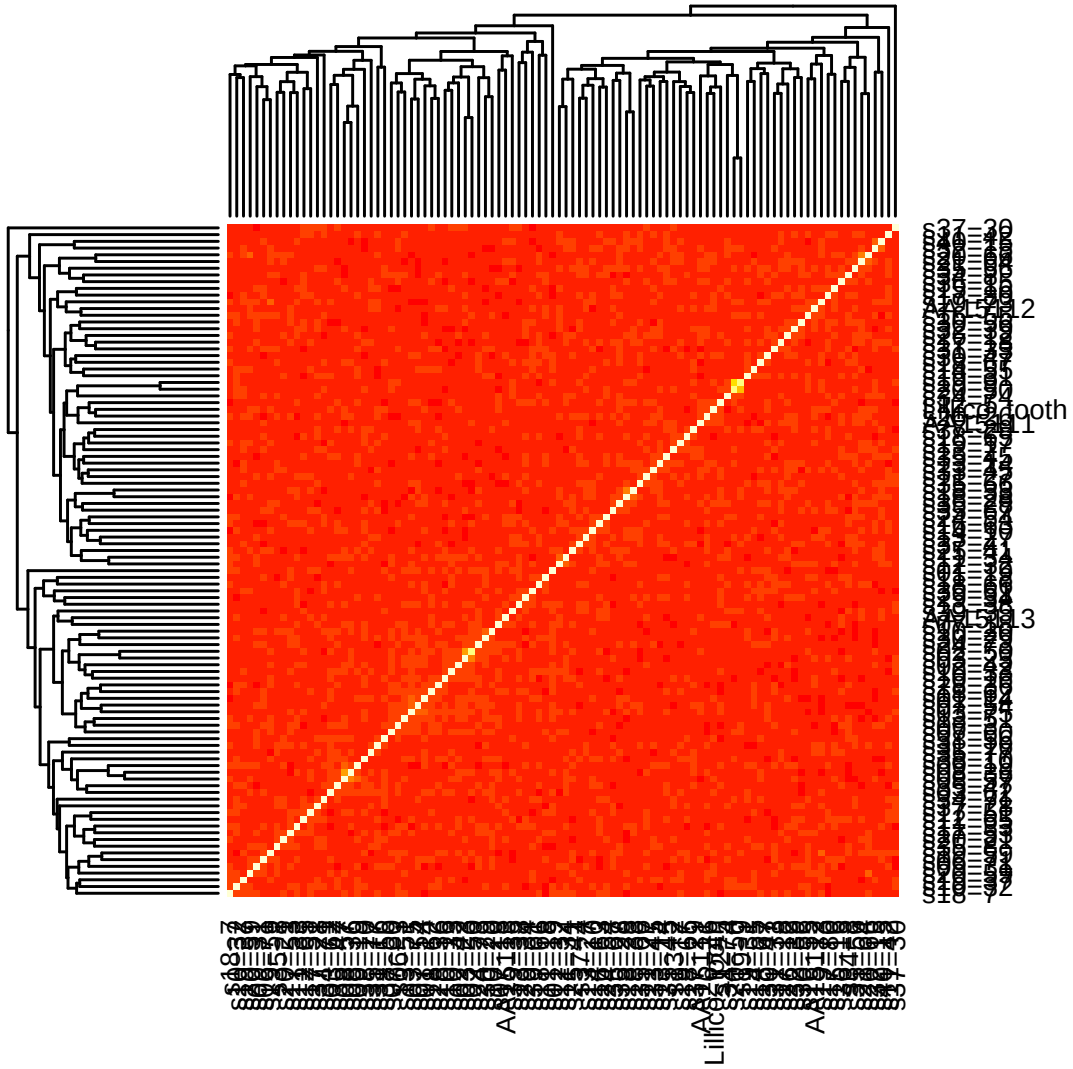
1. Run a PCoA on your data set.
2. Create a scree plot.
3. Change the labels from populations to individuals
4. Use the interactive version that allows you to zoom and interrogate the labels.
5. Generate a three-dimensional plot using the most informative axes, and then undertake a rotation of the 3D solution to display any groupings most clearly.

Advanced 6. Replot your analysis using the sex of the individuals as label (you may want to redefine the pop slot) 7. Identify "outliers" (individuals or pops) in you PCoA and remove them - rerun the PCoA without those "outliers".

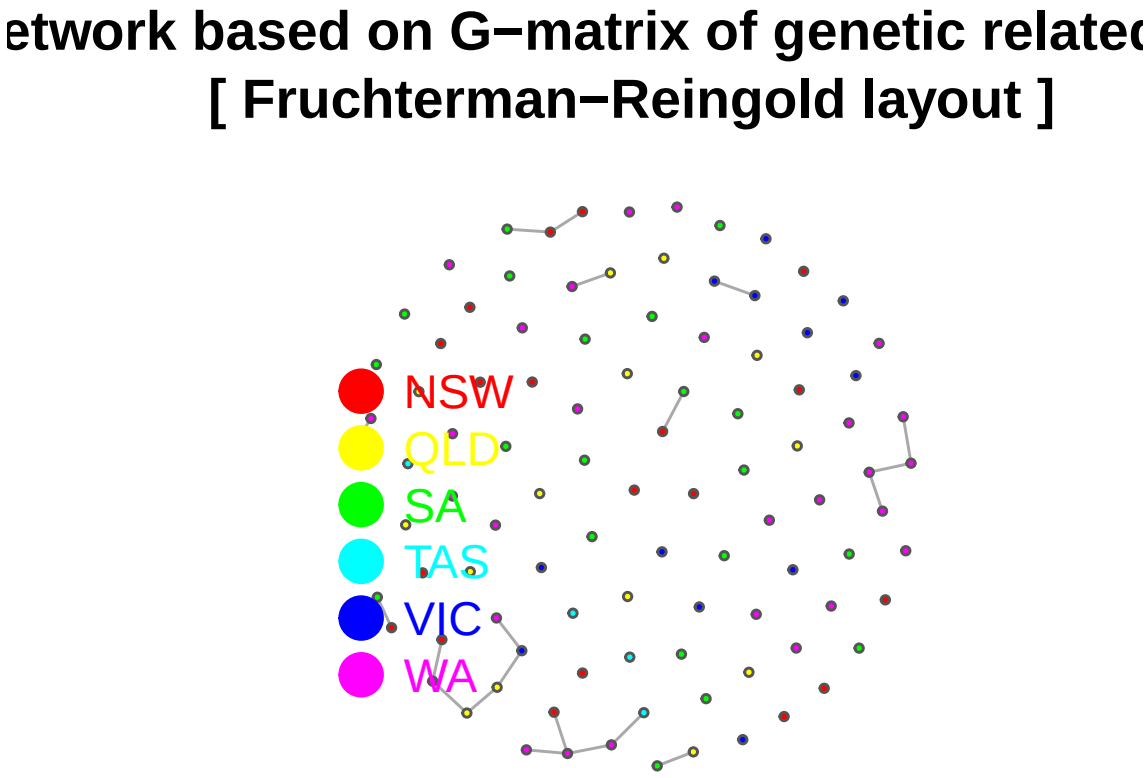
3.2. Genomic relatedness matrix

We can calculate genomic relatedness via:

```
grmatrix <- gl.grm(foxes.gl)
```



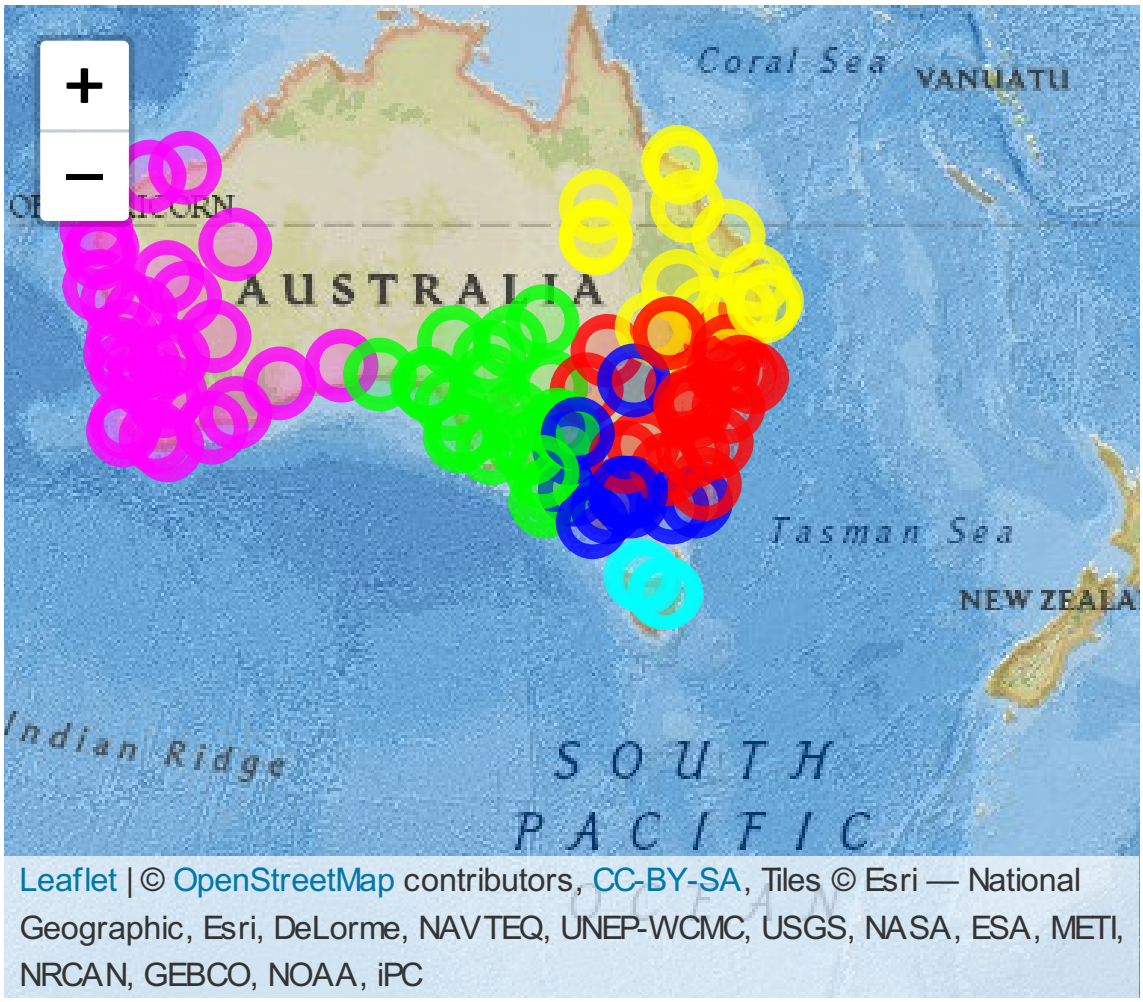
```
gl.grm.network(G = grmatrix, x = foxes.gl)
```



3.3. Mapping your data

Check details via ?gl.map.interactive

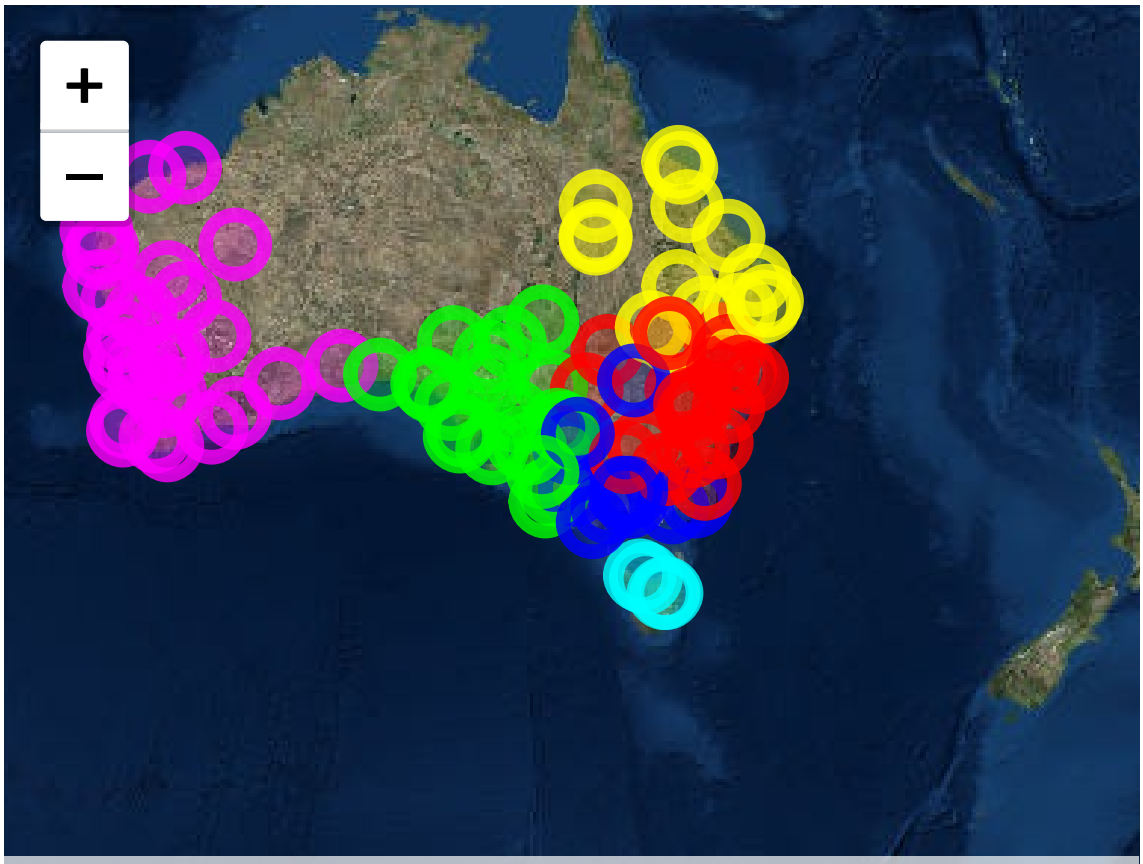
```
gl.map.interactive(foxes.gl)
```



For the map style you need to change the provider string. For example to change to a aerial photo map you could use [8].

[8] More styles can be found via the link in ?gl.map.interactive

```
gl.map.interactive(foxes.gl, provider = "Esri.WorldImagery")
```


+
−
Leaflet | © OpenStreetMap contributors, CC-BY-SA, Tiles © Esri — Source: Esri, i-cubed, USDA, USGS, AEX, GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User Community

4. Export your data

4.1. Save a genlight object

```
# save
saveRDS(gl, file = "gl.rds")

# read (load)
mygl <- readRDS("gl.rds")
```

4.2 Export/convert your data set to other formats

All export functions start with gl2….

function	explanation
gl2fasta	Concatenates DArT trimmed sequences and outputs a fastA file
gl2shp	creates a shp or kml file to be used with ArcGIS or GoogleEarth etc.
gl2structure	creates a file to be use with structure
gl2faststrcture	creates an input file to be used with faststructure
gl2svdquartets	Convert a genlight object to nexus format PAUP SVDquartets
gl.nhybrids	runs a newhybrids analysis (needs to be installed)
gl2gi	converts a genlight to a genind object
gi2g2	converts a genind to a genlight object
gl2plink	converts a genlight into a file in plink format
gl2gds	converts a genlight into a file to be used by SNPRelate
gl2snapp	converts a genlight to a snapp file
gl2sa	converts a genlight to a SNPassoc object

Task

1. Export your data set in any of the formats above and check if you can use it in other packages/softwares.