dartR Workshop - CBA

Bernd Gruber & Arthur Georges

2018-11-27

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].

This tutorial is meant to be run on your own pace, but obviously we encourage discussion while you type (copy/paste) the code. You can run the tutorial with the example data provided or simply adapt it to your own data.

We will cover the following topics:

[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

Overview

Tuesday

1. Preparing data sets

   1.1 Loading data sets into R (DArT format)
   1.2 How to load other formats

2. The genlight object

   2.1 Explore a genlight object
   2.2 quality filter your data
   2.3 subset/recode your data set

3. Visualisation

   3.1. PCoA
   3.2. Genomic relatedness matrix
   3.3. Mapping your data

Wednesday

4. Population genetics

   4.1. Heterozygosity
   4.2. HWE
   4.3. Private and fixed alleles

5. Landscape genetics

   5.1 Isolation by distance
   5.2 Landscape resistance

6. Export data set to other formats/packages

   6.1 saving a genlight object
   6.2 FASTA
   6.3 STRUCTURE, fastSTRUCTURE, NewHybrid

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`

Please note: For this workshop you need to install the github version (which has some additional functions) to be able to run all code examples.

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.

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

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

1. Preparing data sets

1.1. Loading data sets into R (DArT format)

Diversity Arrays Technology Pty Ltd (DArT™) supplies your data as excel spreadsheets in comma delimited format (.csv). Several files are provided.

- **SNP_1row.csv** contains the SNP genotypes in one row format
- **SNP_2row.csv** contains the SNP genotypes in two row format
- **SilicoDArT.csv** contains the presence(1)/absence(0) of the sequence tag at a locus for each individual (analogous to AFLPs)
- **metadata.csv** contains a report of the success of the sequencing and an explanation of the locus metadata provided in the above spreadsheets.

Reading DArT Files into a Genlight Object

SNP data can be read into a genlight object using gl.read.dart(). This function intelligently interrogates the input csv file to determine * if the file is a 1-row or 2-row format, as supplied by Diversity Arrays Technology Pty Ltd. * the number of locus metadata columns to be input (the first typically being AlleleID and the last repAvg). * the number of lines to skip at the top of the csv file before reading the specimen IDs and then the SNP data themselves. * if there are any errors in the data.

For a test we 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)?

You can load the data and convert it into a genlight object (the format supported by all dartR functions) 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 workshop manual.

We can inspect if the 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].

To report the number of loci, individuals and number of populations we can use:

```
# number of loci
nLoc(gl)
```

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


UNIVERSITY OF CANBERRA dartR DArTSeq™ Data

```
# or you could use
length(gl@loc.names)
```

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

```
# number of individuals
nInd(gl)
```

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

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 accessors exist and for some don't. For example: position(), indNames(), locNames(), ploidy() are accessors, but there are none for `\@loc.all, \@other, \@gen`.
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()`.

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). To support here a 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. The entries are either 0, 1, 2 or NA and represent the frequency of the second allele for that individual at that loci.[3].

The matrix has dimensions of nInd x nLoc (250 x 255 in out test data set).

[3] 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
```

Explore loci metrics

If you inspected the provided 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`.

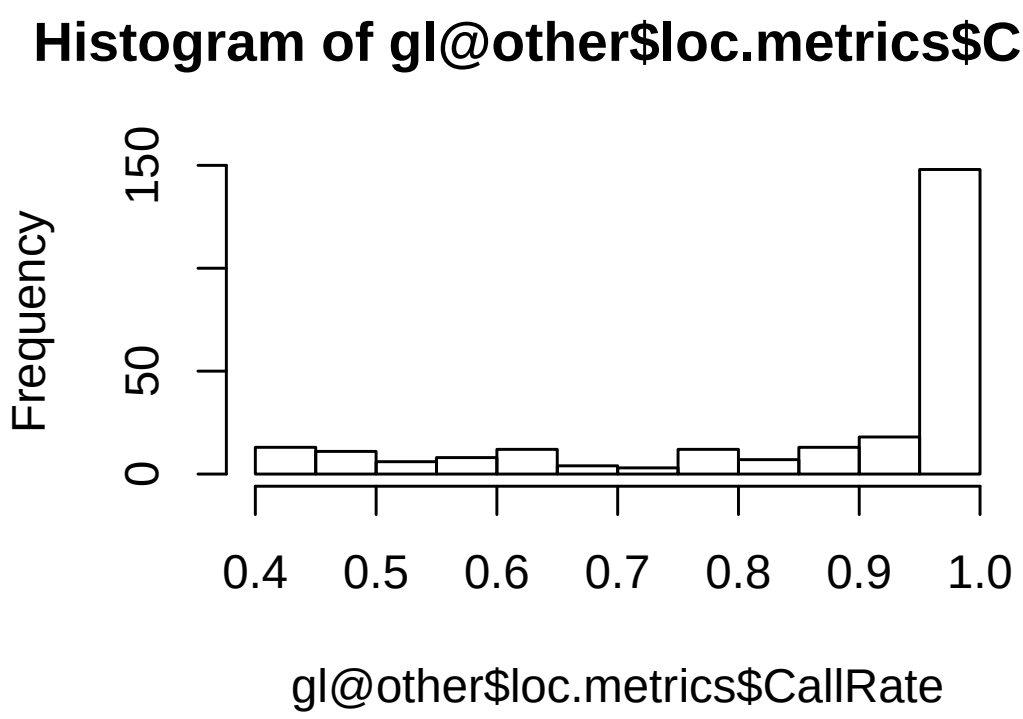
`\@other$loc.metrics` is a data.frame (a table in R), that has a **row** for each SNP loci. You can explore those entries via: [4]

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

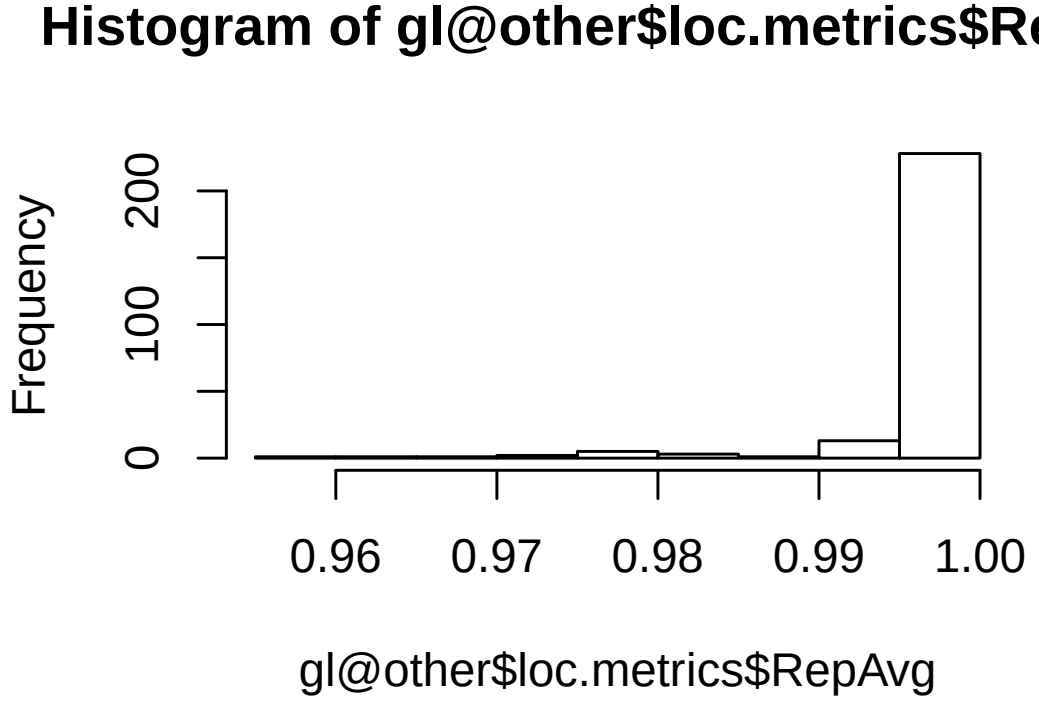
```
# names of loc.metrics
names(gl@other$loc.metrics)
```

```
##  [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)
```



Add ind.metrics

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

```
nPop(gl)
```

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

As none 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 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...
```


UNIVERSITY OF CANBERRA dartR DArTSeq™ Data

```
## 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

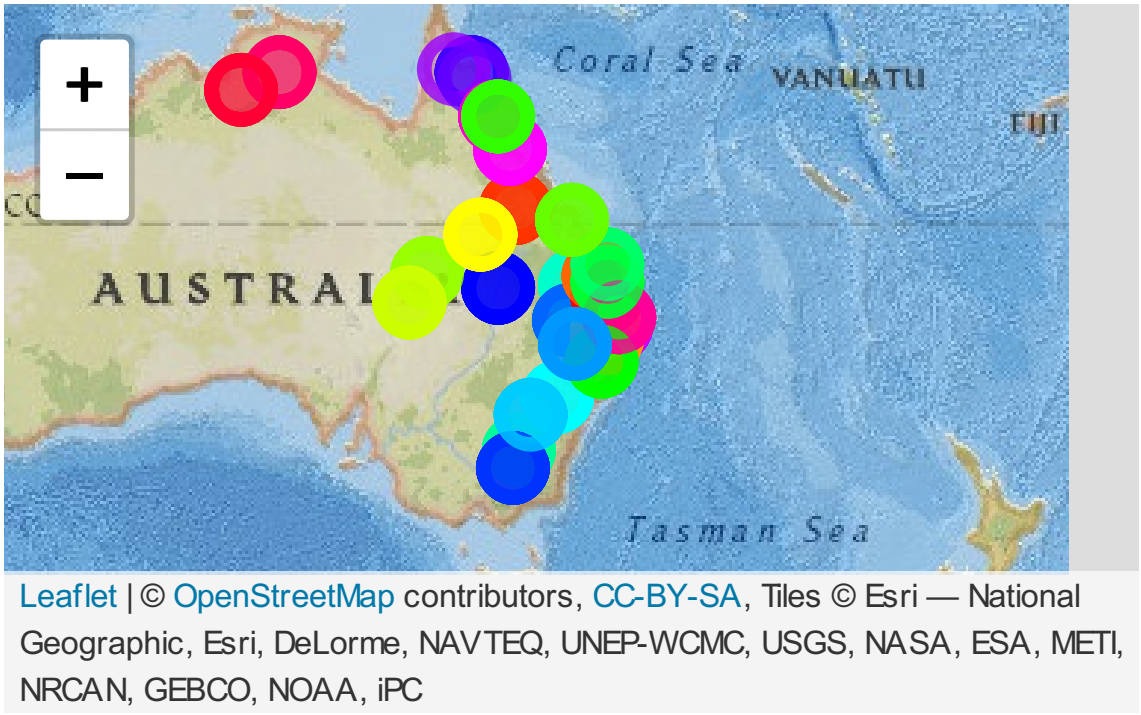
Go through the output in the console. It provides important information if the data have been combined correctly.

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?
6. Explore the data set `foxes.gl` by just type foxes.gl into the console.

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 load other formats

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 the 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:Gruber et al. 2018: https://onlinelibrary.wiley.com/doi/full/10.1111/1755-0998.12745].

You can see there are some options available and most of the 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.

```
fp
```

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

```
dir(fp)
```

```
## [1] "landscape.sim.rdata"
## [2] "platy.csv"
## [3] "testset_metadata.csv"
```

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

```
## [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 wanted to study 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 locis are coded (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

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.

There is no slot called @loc.metrics or @ind.metrics, which are necessary for some functions. For example we have information of 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
```

As you may remember, we do have quite some meta data on loci and 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
```


UNIVERSITY OF CANBERRA dartR DArTSeq™ Data

```
##
## 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
```

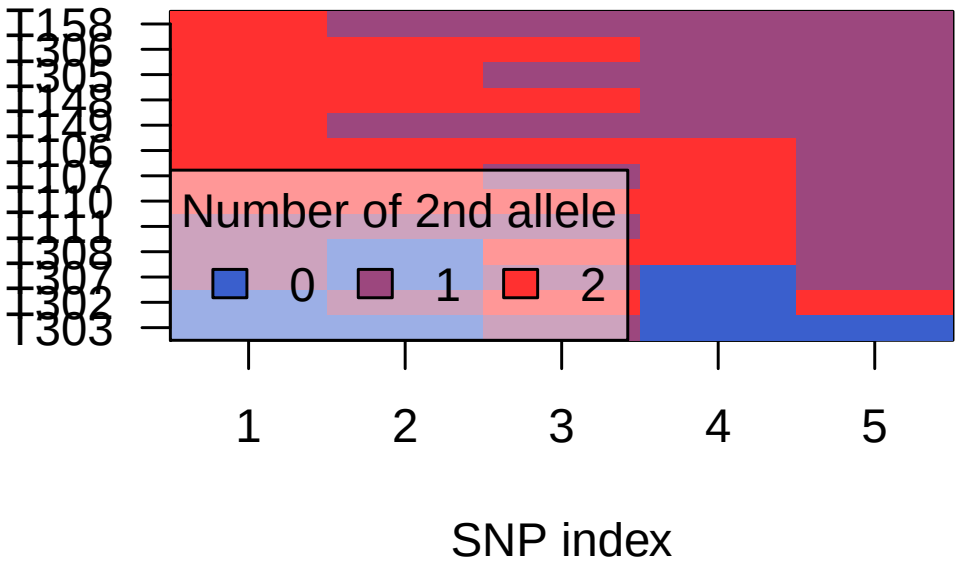
We now have a almost complete genlight object, hence we can use most of the functions from dartR. (see next section)

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:

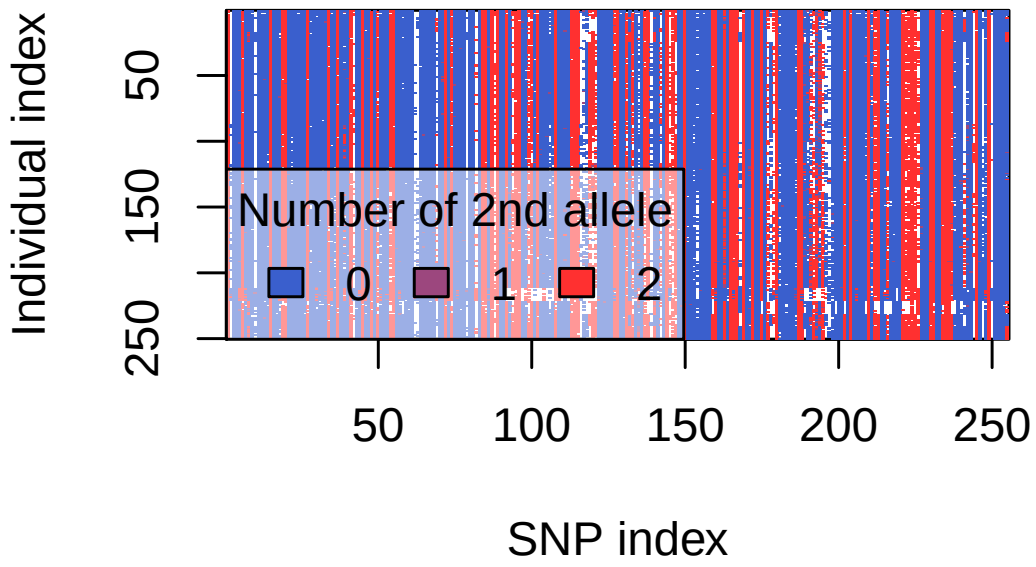
```
platy.gl@other$loc.metrics$TrimmedSequence <- "character.vector of sequences"
position(platy.gl) <- "numeric vector of SNP positions (starting at zero)"
```

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.

2.2 quality filter your data

If 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.
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)

Refer to the help pages of each function for details on the parameters `?nameoffunction`.

1. Filter by repeatability (`gl.filter.repavg` in dartR) (a meassurement of quality per loci)
2. Fitler 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`)

Simple examples how to apply some of the filters are provided below.

For the below examples we will use our testset.gl data set. Feel free to use your own data set. To make life easier we rename our data set to `gl`.

```
gl <- testset.gl
```

1. Filter on call rate, threshold = at least 95% loci called

```
gl2 <- gl.filter.callrate(gl, method = "loc",
    threshold = 0.95)
```

```
## Starting gl.filter.callrate: Filtering on Call Rate
## Starting utils.recalc.callrate: Recalculating CallRate
## Completed utils.recalc.callrate
##
##   Removing loci based on Call Rate, threshold = 0.95
## Note: Locus metrics recalculated
## Note: Resultant monomorphic loci deleted
##   Warning: Some individuals with a CallRate initially >= 0.95 may have a CallRate lower than 0.95 when cal
## gl.filter.callrate completed
```

2. Filter individuals on call rate (threshold =90%)

```
gl2 <- gl.filter.callrate(gl, method = "ind",
    threshold = 0.9)
```

```
## Starting gl.filter.callrate: Filtering on Call Rate
## Starting utils.recalc.callrate: Recalculating CallRate
## Completed utils.recalc.callrate
##
##   Removing individuals based on Call Rate, threshold = 0.9
##   Individuals deleted (CallRate <=  0.9 :
## AA010915[EmmacMDBForb], AA032760[EmmacMDBMaci], AA011723[EmmacBurnBara], AA012411[EmmacCoopEulb], AA01923
##   Deleting monomorphic loci and loci with all NA scores
## Completed gl.filter.monomorphs
##
## Note: Locus metrics not recalculated
## Note: Resultant monomorphic loci deleted
##   Warning: Some individuals with a CallRate initially >= 0.9 may have a CallRate lower than 0.9 when call
## gl.filter.callrate completed
```

3. Filter on reproducibility, threshold (here called t, do not ask why) 100% reproducible

```
gl2 <- gl.filter.repavg(gl, t = 1)
```

```
## Starting gl.filter.repavg: Filtering on repeatability
##   Removing loci with RepAvg < 1
## gl.filter.repavg completed
```

4. Filter out multiple snps in single sequence tags (!!!!!produces an error currently!!!!!)

5. Filter out monomorphic loci

```
gl2 <- gl.filter.monomorphs(gl, v = 0)
```

6. Filter out loci with trimmed sequence tags that are too similar (possible paralogues). Only works if TrimmedSequence is available in the loci metadata, therefore we use another test data set here.

```
gl2 <- gl.filter.hamming(testset.gl, t = 0.25,
    pb = F)
```

```
## Starting gl.filter.hamming: Filtering on Hamming Distance
##   Calculating Hamming distances between sequence tags
## gl.filter.hamming completed
```

Note: This filter and its accompanying report function is slow when there are many loci. Recommended that it be applied after all other filtering, and only if less than 20,000 loci remain. May require an overnight run.

Please note in the examples we always stored the resulting filter into a new `genlight` object `gl2`, `gl3` etc. Though it is a bit of a waste in terms of memory, it avoids confusion which filter you have already applied. A series of filter could then look like:

```
gl2 <- gl.filter.callrate(gl, method = "loc",
    threshold = 0.95)
gl3 <- gl.filter.callrate(gl2, method = "ind",
    threshold = 0.9)
gl4 <- gl.filter.repavg(gl3, t = 1)
```

Note that filters that result in the removal of populations or individual have optional parameters to request that the locus metadata be recalculated or for resultant monomorphic loci to be removed. Recalculation of the locus metadata is necessary because callrate, for example, will no longer be accurate once some individuals have been removed from the dataset.

2.3 subset/recode your data set

Recall that the metadata file provided when the data are initially input contains information assigned to each individual including, often at a minimum, population assignment. 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:

```
# 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"
```

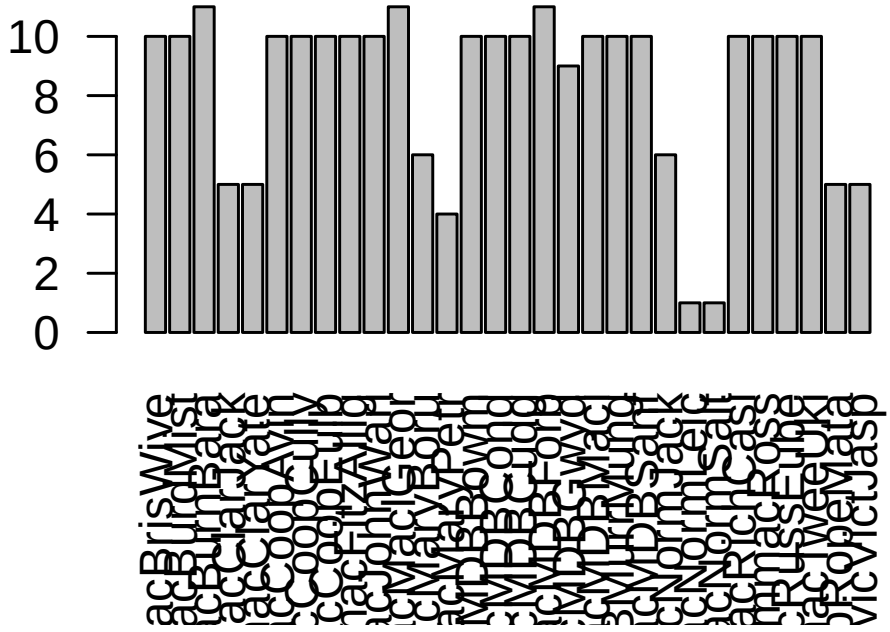
```
# table on individuals per population
table(pop(gl))
```

```
##
##    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
```

It is easy to create a barplot on the number of individuals per population:

```
barplot(table(pop(gl)), las = 2)
```



If you have only a few reassignments, the simplest way is to use one or more of the scripts gl <- gl.keep.pop(gl, c(pop1, pop5, pop7)) # Retains only populations 1, 5 and 7 gl <- gl.drop.pop(gl, c(pop2, pop3, pop4, pop6)) # Deletes populations 2, 3, 4, 6 gl <- gl.merge.pop(gl, old=c("pop1","pop2"), new="pop1") # Merges populations 1 and 2 as pop1 gl <- gl.merge.pop(gl, old=c("pop1"), new="outgroup") # Renames population 1 to a population labelled outgroup

Try these out for yourself.

If only a few populations are involved, then the best option is to use the gl.drop.pop or gl.keep.pop functions.

```
glnew3 <- gl.keep.pop(gl, pop.list = c("EmsubRopeMata",
    "EmvicVictJasp"))
```

will delete all individuals in all populations except those listed.

```
glnew3 <- gl.drop.pop(gl, pop.list = c("EmsubRopeMata",
    "EmvicVictJasp"))
```

will delete all individuals in the listed populations.

```
glnew3 <- gl.merge.pop(gl, old = c("EmsubRopeMata",
    "EmvicVictJasp"), new = "outgroup")
```

will reassign individuals in populations EmsubRopeMata and EmvicVictJasp to a new population called outgroup.

```
glnew3 <- gl.merge.pop(gl, old = "EmsubRopeMata",
    new = "Emydura_victoriae")
```

will rename population EmvicVictJasp to Emydura_victoriae.

Note that there is an option for recalculating the relevant individual metadata, and for removing resultant monomorphic loci.

Individual reassignment

You can reassign individuals to new populations in a number of ways.A similar set of scripts apply to individuals.

The initial individual labels entered at the time of reading the data into the genlight object can be viewed via [5]:

[5] 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

gl <- gl.keep.ind(gl, c(ind1, ind5, ind7)) # Retains only individuals labelled ind1, 5 and 7 gl <- gl.drop.pop(gl, c(ind2, ind3, ind4, ind6)) # Deletes populations 2, 3, 4, 6

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

```
glnew3 <- gl.keep.ind(gl, ind.list = c("AA019073",
    "AA004859"))
```

will delete all individuals except those listed.

```
glnew3 <- gl.drop.pop(gl, ind.list = c("AA019073",
    "AA004859"))
```

will delete all individuals listed.

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:

```
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"
```

Task

Try this using 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 script

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 manual [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.

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: 284.1 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  18
```

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.

To take care for the correct filtering for loci and individuals we suggest therefore to use the following approach:

1. create an index for individuals (if you want to subset by individuals)
2. create an index for loci (if you want to subset by loci))

For example you want to have only individuals of two populations ("EmmacRussEube" or "EmvicVictJasp") and 30 randomly selected loci you could type:

```
index.ind <- pop(gl) == "EmmacRussEube" | pop(gl) ==
    "EmvicVictJasp"
# check if the index worked
table(pop(gl), index.ind)
```

```
##                  index.ind
##                   FALSE TRUE
##   EmmacBrisWive       10    0
##   EmmacBurdMist       10    0
##   EmmacBurnBara       11    0
##   EmmacClarJack        5    0
##   EmmacClarYate        5    0
##   EmmacCoopAvin       10    0
##   EmmacCoopCully      10    0
##   EmmacCoopEulb       10    0
```


UNIVERSITY OF CANBERRA dartR DArTSeq™ Data

```
##   EmmacFitzAllig      10    0
##   EmmacJohnWari       10    0
##   EmmacMaclGeor       11    0
##   EmmacMaryBoru        6    0
##   EmmacMaryPetr        4    0
##   EmmacMDBBowm        10    0
##   EmmacMDBCond        10    0
##   EmmacMDBCudg        10    0
##   EmmacMDBForb        11    0
##   EmmacMDBGwyd         9    0
##   EmmacMDBMaci        10    0
##   EmmacMDBMurrMung    10    0
##   EmmacMDBSanf        10    0
##   EmmacNormJack        6    0
##   EmmacNormLeic        1    0
##   EmmacNormSalt        1    0
##   EmmacRichCasi       10    0
##   EmmacRoss           10    0
##   EmmacRussEube        0   10
##   EmmacTweeUki        10    0
##   EmsubRopeMata        5    0
##   EmvicVictJasp        0    5
```

```
index.loc <- sample(nLoc(gl), 30, replace = F)
index.loc
```

```
##  [1] 241  91 172 161  89 244 140  80 116  51
## [11] 200 164  54 202 175  69  76  45   8 177
## [21]  36 125  44  27 215  83 162  11 178 184
```

and then

3. apply the indices to the genlight object and the meta data at the same time:

```
glsub2 <- gl[index.ind, index.loc]
glsub2@other$ind.metrics <- gl@other$ind.metrics[index.ind,
    ]  #not necessary
glsub2@other$loc.metrics <- gl@other$loc.metrics[index.loc,
    ]  #necessary
```

We can check the result via:

```
glsub2
```

```
##  /// GENLIGHT OBJECT /////////
```


UNIVERSITY OF CANBERRA dartR DArTSeq™ Data

```
##
##  // 15 genotypes,  30 binary SNPs, size: 272.9 Kb
##  65 (14.44 %) missing data
##
##  // Basic content
##    @gen: list of 15 SNPbin
##    @ploidy: ploidy of each individual  (range: 2-2)
##
##  // Optional content
##    @ind.names:  15 individual labels
##    @loc.names:  30 locus labels
##    @loc.all:  30 alleles
##    @position: integer storing positions of the SNPs
##    @pop: population of each individual (group size range: 5-10)
##    @other: a list containing: loc.metrics  latlong  ind.metrics
```

```
dim(glsub2@other$ind.metrics)
```

```
## [1] 15  6
```

```
dim(glsub2@other$loc.metrics)
```

```
## [1] 30 18
```

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.

3. Visualisation

3.1. PCoA

Recall that the metadata file provided when the data are initially input contains information assigned to each individual including, often at a minimum, population assignment. 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:

```
# 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"
```

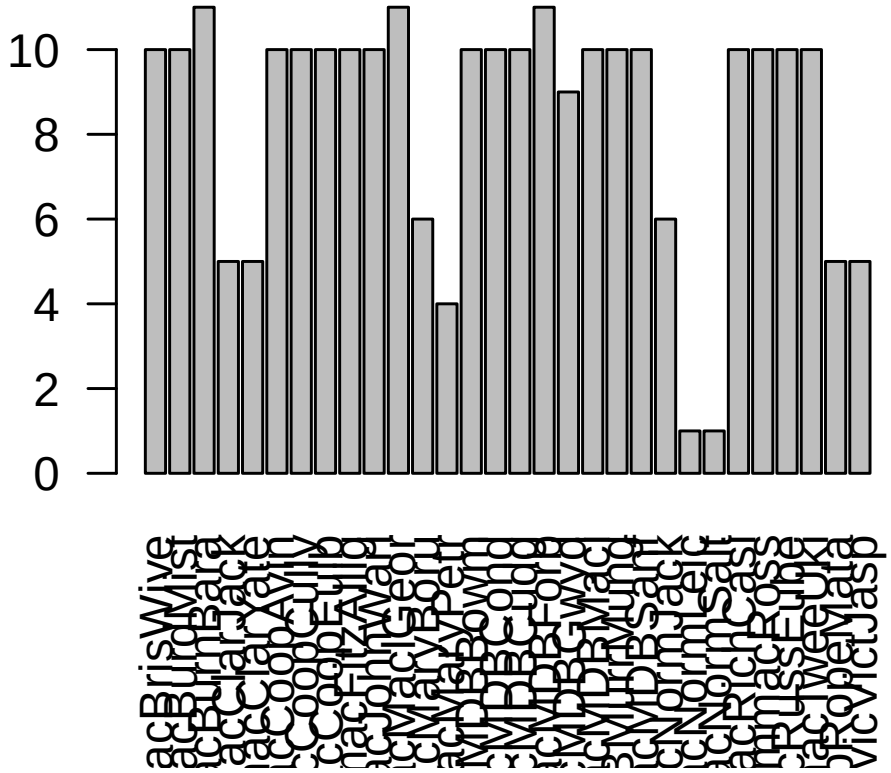
```
# table on individuals per population
table(pop(gl))
```

```
##
##    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
```

It is easy to create a barplot on the number of individuals per population:

```
barplot(table(pop(gl)), las = 2)
```



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

gl <- gl.keep.pop(gl, c(pop1, pop5, pop7)) # Retains only populations 1, 5 and 7 gl <- gl.drop.pop(gl, c(pop2, pop3, pop4, pop6)) # Deletes populations 2, 3, 4, 6 gl <- gl.merge.pop(gl, old=c("pop1","pop2"), new="pop1") # Merges populations 1 and 2 as pop1 gl <- gl.merge.pop(gl, old=c("pop1"), new="outgroup") # Renames population 1 to a population labelled outgroup

Try these out for yourself.

If only a few populations are involved, then the best option is to use the gl.drop.pop or gl.keep.pop functions.

```
glnew3 <- gl.keep.pop(gl, pop.list = c("EmsubRopeMata",
    "EmvicVictJasp"))
```

will delete all individuals in all populations except those listed.

```
glnew3 <- gl.drop.pop(gl, pop.list = c("EmsubRopeMata",
    "EmvicVictJasp"))
```

will delete all individuals in the listed populations.

```
glnew3 <- gl.merge.pop(gl, old = c("EmsubRopeMata",
    "EmvicVictJasp"), new = "outgroup")
```

will reassign individuals in populations EmsubRopeMata and EmvicVictJasp to a new population called outgroup.

```
glnew3 <- gl.merge.pop(gl, old = "EmsubRopeMata",
    new = "Emydura_victoriae")
```

will rename population EmvicVictJasp to Emydura_victoriae.

Note that there is an option for recalculating the relevant individual metadata, and for removing resultant monomorphic loci.

Individual reassignment

You can reassign individuals to new populations in a number of ways.A similar set of scripts apply to individuals.

The initial individual labels entered at the time of reading the data into the genlight object can be viewed via[6]:

[6] 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

gl <- gl.keep.ind(gl, c(ind1, ind5, ind7)) # Retains only individuals labelled ind1, 5 and 7 gl <- gl.drop.pop(gl, c(ind2, ind3, ind4, ind6)) # Deletes populations 2, 3, 4, 6

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

```
glnew3 <- gl.keep.ind(gl, ind.list = c("AA019073",
    "AA004859"))
```

will delete all individuals except those listed.

```
glnew3 <- gl.drop.pop(gl, ind.list = c("AA019073",
    "AA004859"))
```

will delete all individuals listed.

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:

```
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"
```

Task

Try this using 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 script

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 manual [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.

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: 284.1 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  18
```

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.

To take care for the correct filtering for loci and individuals we suggest therefore to use the following approach:

1. create an index for individuals (if you want to subset by individuals)
2. create an index for loci (if you want to subset by loci))

For example you want to have only individuals of two populations ("EmmacRussEube" or "EmvicVictJasp") and 30 randomly selected loci you could type:

```
index.ind <- pop(gl) == "EmmacRussEube" | pop(gl) ==
    "EmvicVictJasp"
# check if the index worked
table(pop(gl), index.ind)
```

```
##                  index.ind
##                   FALSE TRUE
##   EmmacBrisWive       10    0
##   EmmacBurdMist       10    0
##   EmmacBurnBara       11    0
##   EmmacClarJack        5    0
##   EmmacClarYate        5    0
##   EmmacCoopAvin       10    0
##   EmmacCoopCully      10    0
##   EmmacCoopEulb       10    0
```


UNIVERSITY OF CANBERRA dartR DArTSeq™ Data

```
##   EmmacFitzAllig      10    0
##   EmmacJohnWari       10    0
##   EmmacMaclGeor       11    0
##   EmmacMaryBoru        6    0
##   EmmacMaryPetr        4    0
##   EmmacMDBBowm        10    0
##   EmmacMDBCond        10    0
##   EmmacMDBCudg        10    0
##   EmmacMDBForb        11    0
##   EmmacMDBGwyd         9    0
##   EmmacMDBMaci        10    0
##   EmmacMDBMurrMung    10    0
##   EmmacMDBSanf        10    0
##   EmmacNormJack        6    0
##   EmmacNormLeic        1    0
##   EmmacNormSalt        1    0
##   EmmacRichCasi       10    0
##   EmmacRoss           10    0
##   EmmacRussEube        0   10
##   EmmacTweeUki        10    0
##   EmsubRopeMata        5    0
##   EmvicVictJasp        0    5
```

```
index.loc <- sample(nLoc(gl), 30, replace = F)
index.loc
```

```
##  [1]  45  21 101 187 182  88 143 230 242 196
## [11] 206  41  54 223  93  17 236 102 215  14
## [21]  34 156  16  70 237  35  56  80 145 124
```

and then

3. apply the indices to the genlight object and the meta data at the same time:

```
glsub2 <- gl[index.ind, index.loc]
glsub2@other$ind.metrics <- gl@other$ind.metrics[index.ind,
    ]  #not necessary
glsub2@other$loc.metrics <- gl@other$loc.metrics[index.loc,
    ]  #necessary
```

We can check the result via:

```
glsub2
```

```
##  /// GENLIGHT OBJECT /////////
```

```
##
##  // 15 genotypes,  30 binary SNPs, size: 272.9 Kb
##  57 (12.67 %) missing data
##
##  // Basic content
##    @gen: list of 15 SNPbin
##    @ploidy: ploidy of each individual  (range: 2-2)
##
##  // Optional content
##    @ind.names:  15 individual labels
##    @loc.names:  30 locus labels
##    @loc.all:  30 alleles
##    @position: integer storing positions of the SNPs
##    @pop: population of each individual (group size range: 5-10)
##    @other: a list containing: loc.metrics  latlong  ind.metrics
```

```
dim(glsub2@other$ind.metrics)
```

```
## [1] 15  6
```

```
dim(glsub2@other$loc.metrics)
```

```
## [1] 30 18
```

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.

3. Population structure

Visualisation

Genetic similarity of individuals and populations can be visualized by way of Principal Coordinates Analysis (PCoA) ordination (Gower, 1966). Individuals (entities) are represented in a space defined by loci (attributes) with the position along each locus axis determined by genotype (0 for homozygous reference SNP, 2 for homozygous alternate SNP, and 1 for the heterozygous state). Alternatively, populations can be regarded as the entities to be plotted in a space defined by the loci, with the position along each locus axis determined by the relative frequency of the alternate allele.

Orthogonal linear combinations of the original axes are calculated and ordinated such that the first PCoA axis explains the most variation, PCoA-2 is orthogonal to PCoA-1 and explains the most residual variation, and so on. A scree plot of eigenvalues provides an indication of the number of informative axes to examine, viewed in the context of the average percentage variation explained by the original variables. The data are typically presented in two or three dimensions in which emergent structure in the data is evident.

PCoA in dartR

The script `gl.pcoa()` is essentially a wrapper for `glPca()` of package `adegenet` with default settings apart from setting parallel=FALSE, converting the eigenvalues to percentages and some additional diagnostics.

```
pc <- gl.pcoa(gl, nfactors = 5)
```

```
## Performing a PCoA, individuals as entities, SNP loci as attributes
## Ordination yielded 14 informative dimensions from 249 original dimensions
##   PCoA Axis 1 explains 23.3 % of the total variance
##   PCoA Axis 1 and 2 combined explain 42.8 % of the total variance
##   PCoA Axis 1-3 combined explain 54.4 % of the total variance
```

Please note, in case you are using a non-windows system you can use the argument '"parallel=TRUE"', which speeds up the calculation. The resultant object pc contains the eigenvalues, factor scores and factor loadings that can be accessed for subsequent analyses.

```
names(pc)
```

```
## [1] "eig"      "scores"   "loadings"
## [4] "call"
```

The eigenvalues give the scaling factor for the eigenvectors (PCoA axis 1 - n), the scores give the coordinates of the points (the entities, be they individuals or populations) in the new ordinated space, and the loadings give the correlations of the original variables (the loci) against the new axes. Loci that load high on axis 1 are influential in discrimination among the entities in the direction of axis 1.

For example the percentage of variation the is represented by the axes can be calculated and visualised via:

```
barplot(pc$eig/sum(pc$eig) * 100, )
```

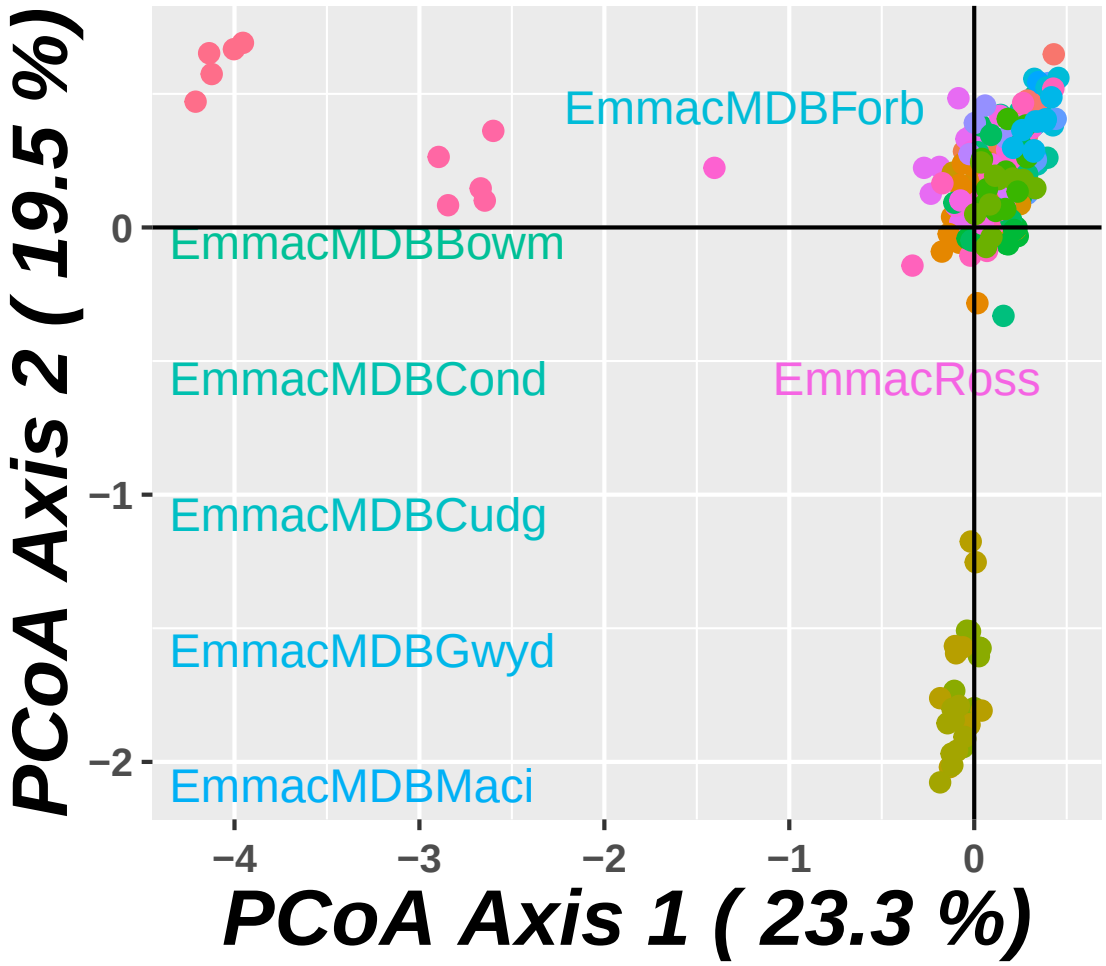
Plotting the results of a PCoA

The results of the PCoA can be plotted using gl.pcoa.plot() with a limited range of options. The script is essentially a wrapper for plot {ggplot2} with the added functionality of {directlabels} and {plotly}.

The plotting script is not intended to produce publication quality plots, but should form a basis for importing the plots to illustrator for subsequent amendment. The command

```
gl.pcoa.plot(pc, gl, labels = "pop", xaxis = 1,
    yaxis = 2)
```

```
## Plotting populations
```



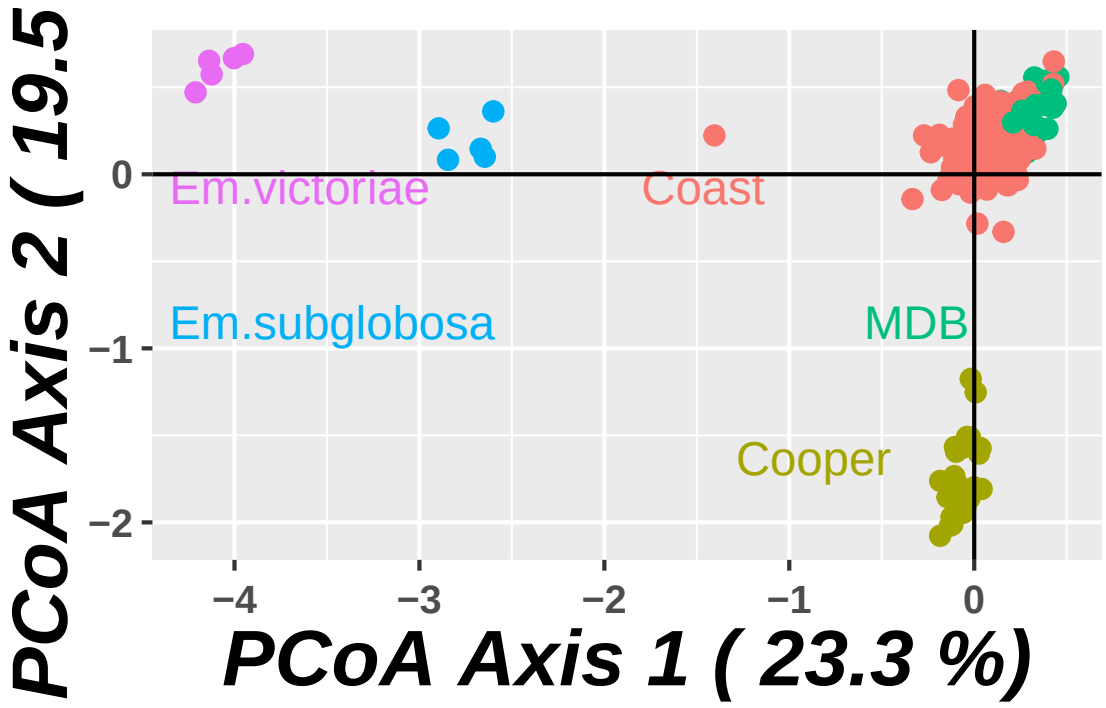
You can see that this plot is very busy, and that the many labels are displaced quite some distance from their associated points. This is because there is a tradeoff between avoiding overlap of the labels and proximity of the labels - you can use colour to identify which labels go with which points. More sensibly, recoding populations would be in order. We could use

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

or using R

```
glnew <- testset.gl
levels(pop(glnew)) <- c(rep("Coast", 5), rep("Cooper",
    3), rep("Coast", 5), rep("MDB", 8), rep("Coast",
    7), "Em.subglobosa", "Em.victoriae")
gl.pcoa.plot(pc, glnew, labels = "pop", xaxis = 1,
    yaxis = 2)
```

```
## Plotting populations
```



Note that we did not need to re run the PCoA analysis, only to recode the pop labels in the genlight object that we hand to the plotting routine. Much clearer plot now.

There are other options for gl.pcoa.plot() that allow the axes to be scaled on the basis of proportion of variation explained, to select other combinations of axes to plot, and for adding confidence ellipses. Use the R help facility to explore these additional options.

Note that there is one point that seems intermediate between *Emydura macquarii* from the coast, and Emydura subglobosa (from northern Australia west of the Great Dividing Range). How do we find out what individual that point represents? Replot the data using labels="interactive" to prime the plot for analysis using ggplotly {plotly}:

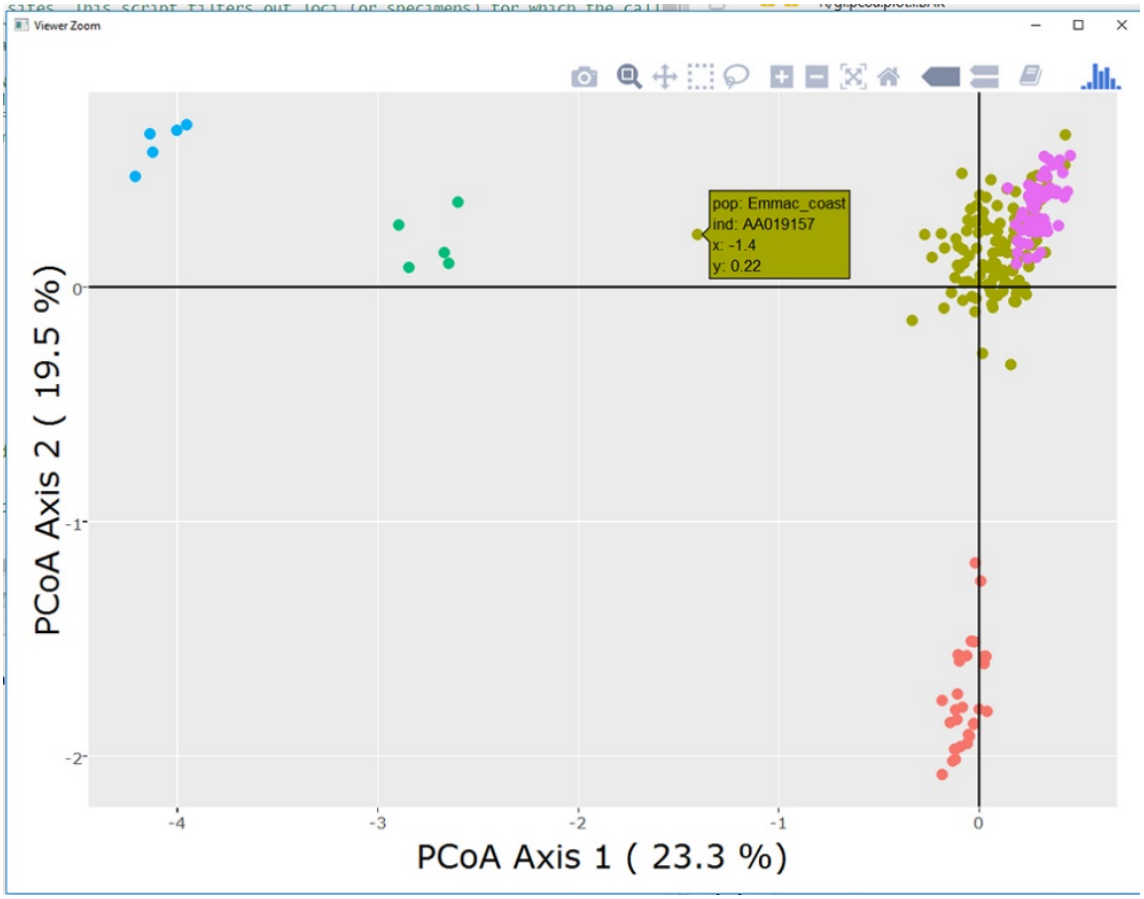
In case ggplotly is not installed, please type:

```
install.packages("devtools")
library(devtools)
install_github("hadley/ggplot2")
library(ggplot2)
```

The commands

```
gl.pcoa.plot(pc, glnew, labels = "interactive",
    xaxis = 1, yaxis = 2)
ggplotly()
```

will plot the individuals in the top two dimensions of the ordinated space, colour the points in accordance to the population to which they belong, and allow points to be identified interactively using the mouse.



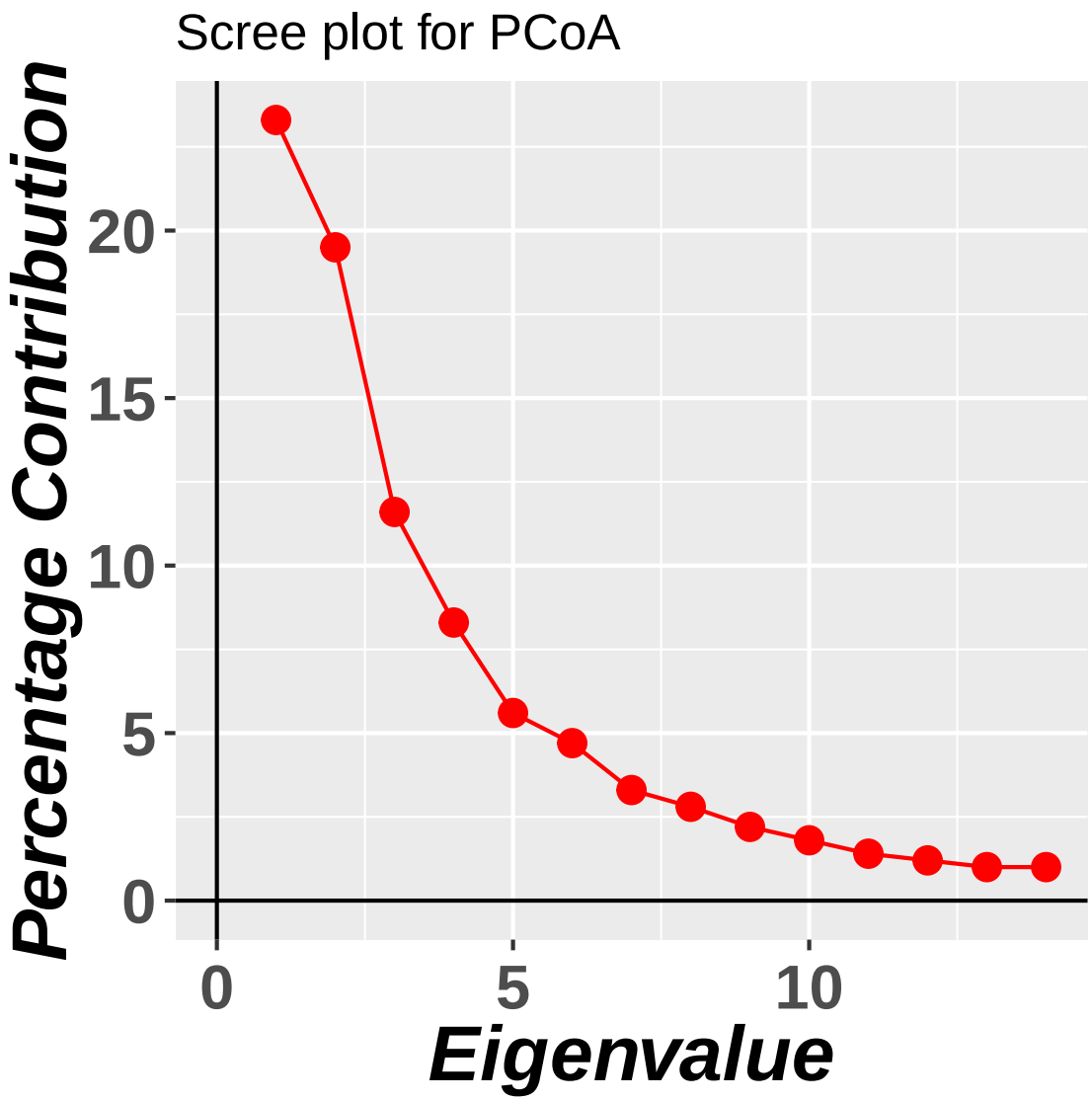
Now moving the mouse over the point reveals its identity. The animal is AA19157, from the coastal populations, and further scrutiny reveals it is from the Barron River in northern Queensland. Seems there has been some allelic exchange there.

The Scree Plot

The number of dimensions with substantive information content can be determined by examining a scree plot (Cattell, 1966).

```
gl.pcoa.scree(pc)
```

```
## Note: Only eigenvalues for dimensions that explain more that the average of the original variables are sho
## No. of axes each explaining 10% or more of total variation: 3
```

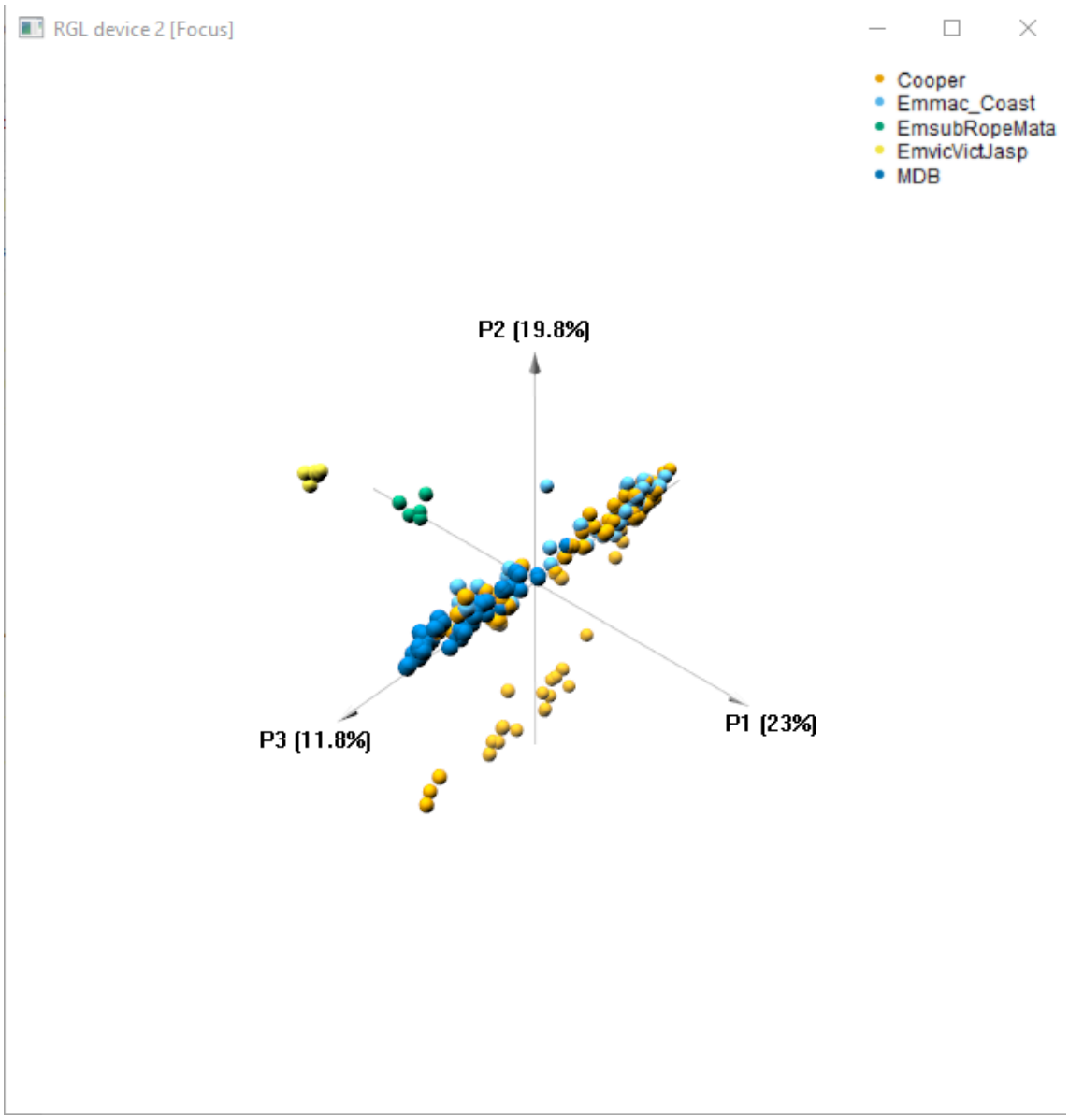


This plot, by default, will show the percentage variation in the data explained by each axis successively where the amount of variation is substantive. By substantive, I mean explaining more than the original variables did on average. As a rule of thumb, one should examine all dimensions that explain more than 10% of the variation in the data.

3D Plot

Should you find that 2 dimensions are insufficient to capture all substantive variation, you can examine a plot of PCoA axis 2 against axis 1 and axis 3 against axis 1 and so on, taking care to note the proportion of variation explained by each axis. Alternatively, when the data cluster tightly, additional dimensions can be examined by removing all individuals from the analysis except those belonging to a single cluster and re-running the PCoA (Georges and Adams, 1992). If three dimensions are indicated by the scree plot, as in our current case, an interactive 3D plot can be produced

```
gl.pcoa.plot.3d(pc, glnew)  #does not work on cluster
```



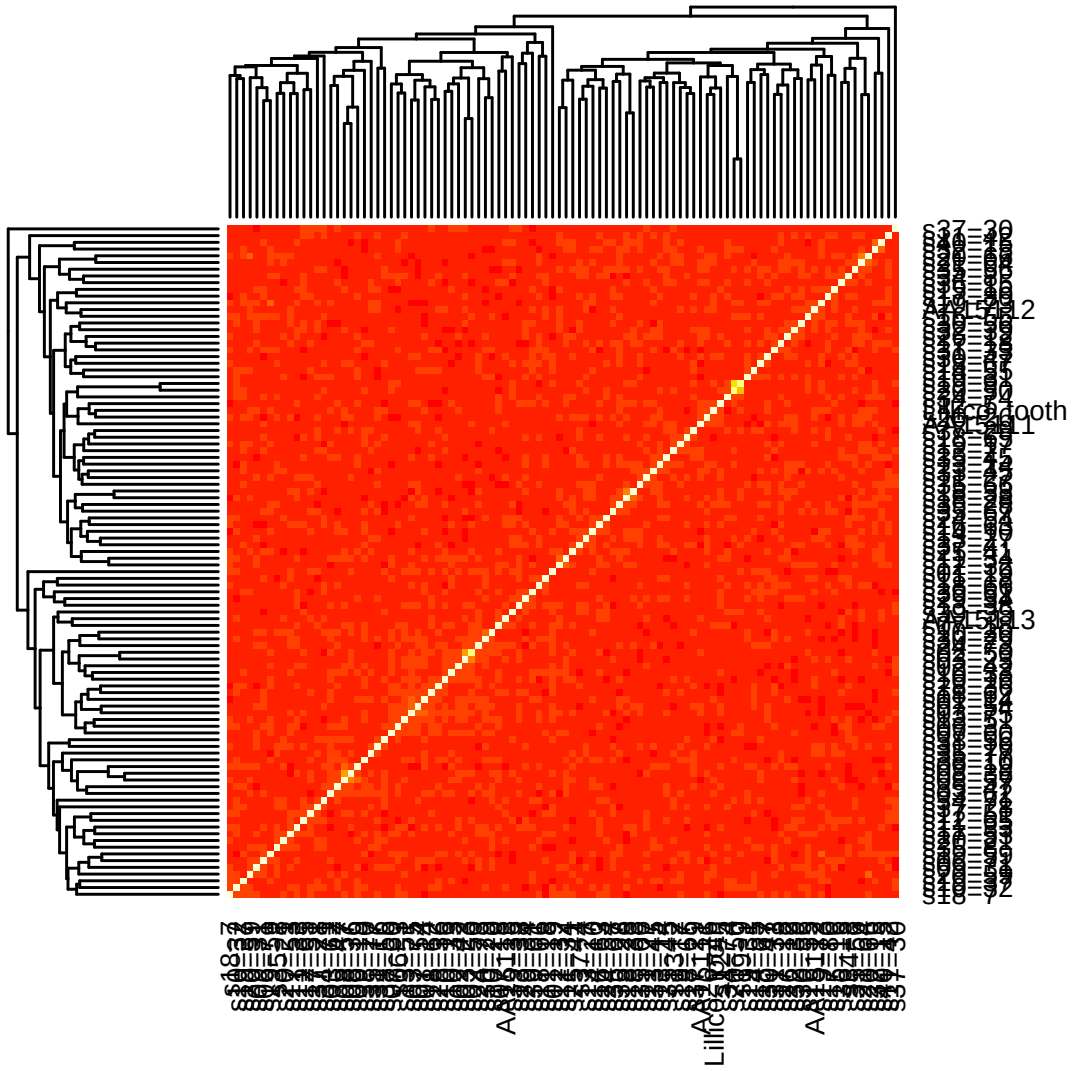
Note that the plot appears in a new window, outside R Studio, and that it is interactive in the sense that you can rotate the plot using the mouse to obtain the most discriminatory view.

This function is essentially a wrapper for the corresponding function in {pca3d}, adding percentage variation explained to each axis and fixing some parameters.

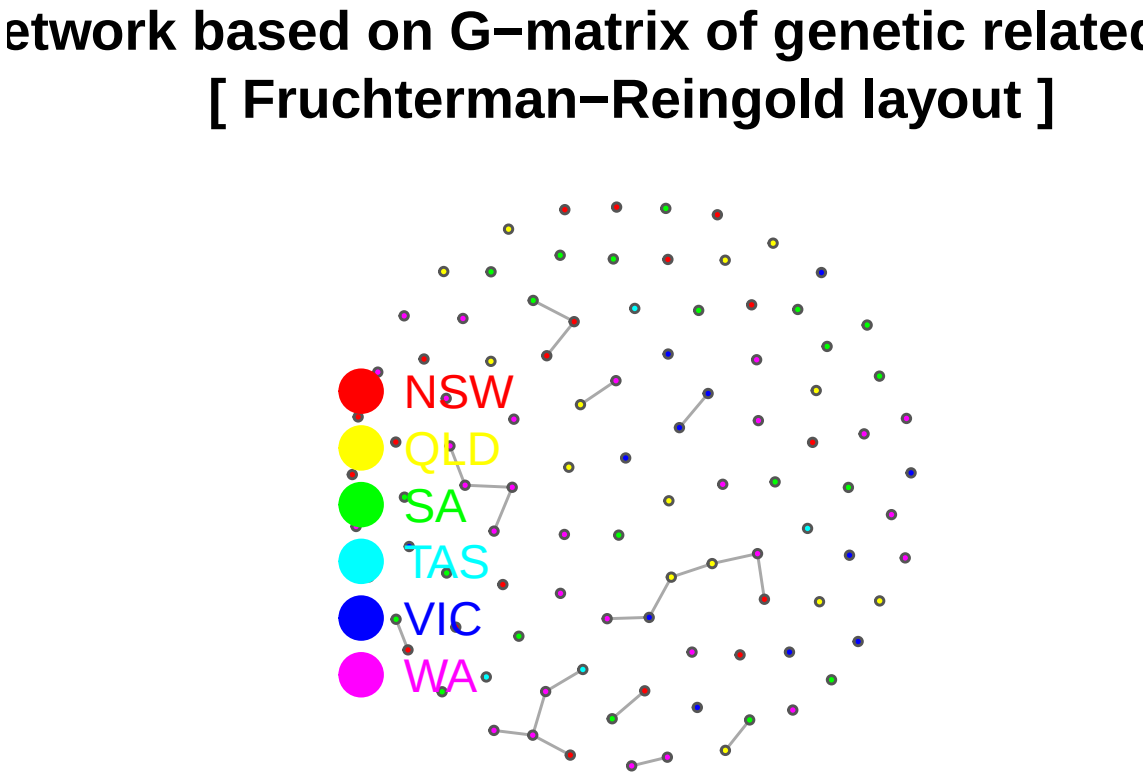
3.2. Genomic relatedness matrix

We can calculate genomic relatedness via:

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



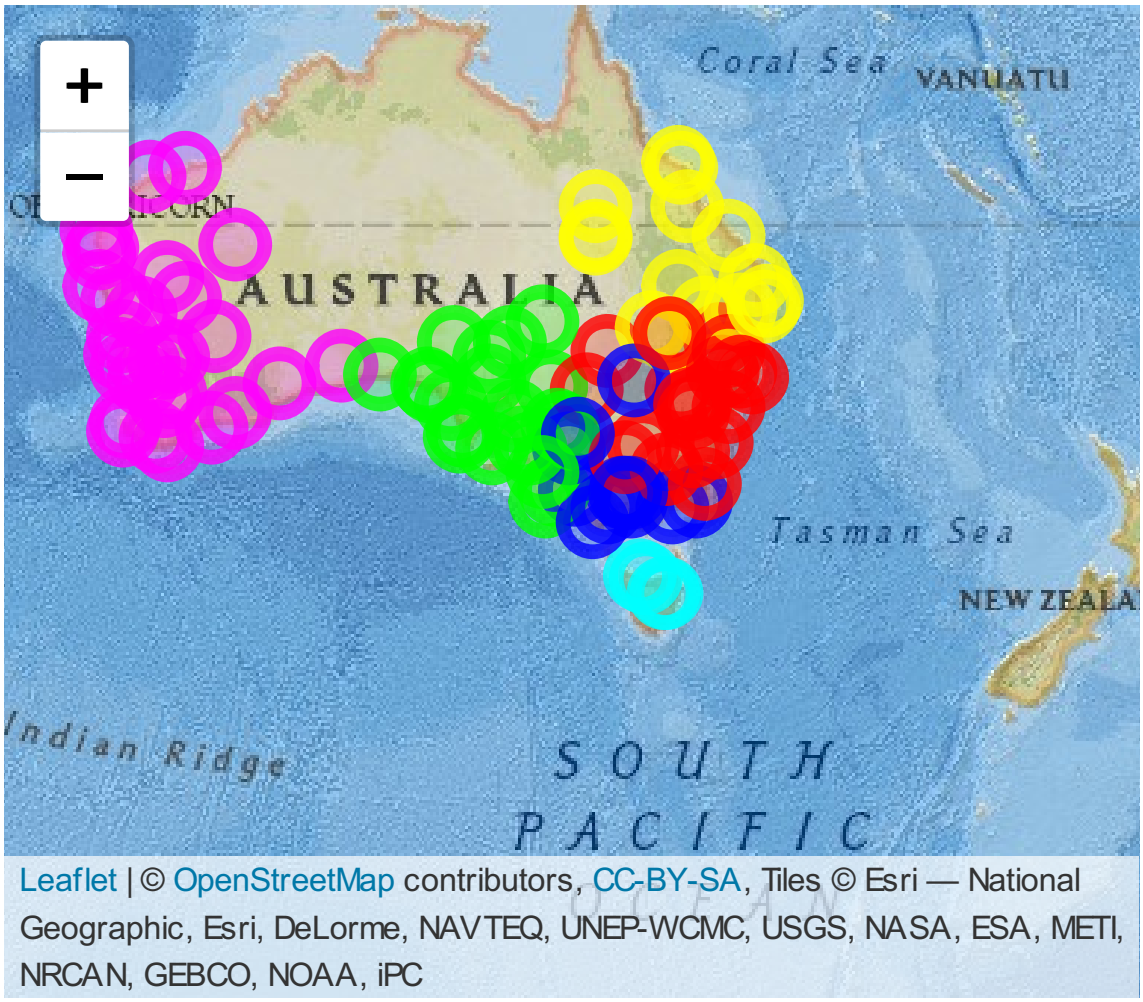
```
gl.grm.network(G = grmatrix, x = foxes.gl)
```



3.3. Mapping your data

Check details via ?gl.map.interactive

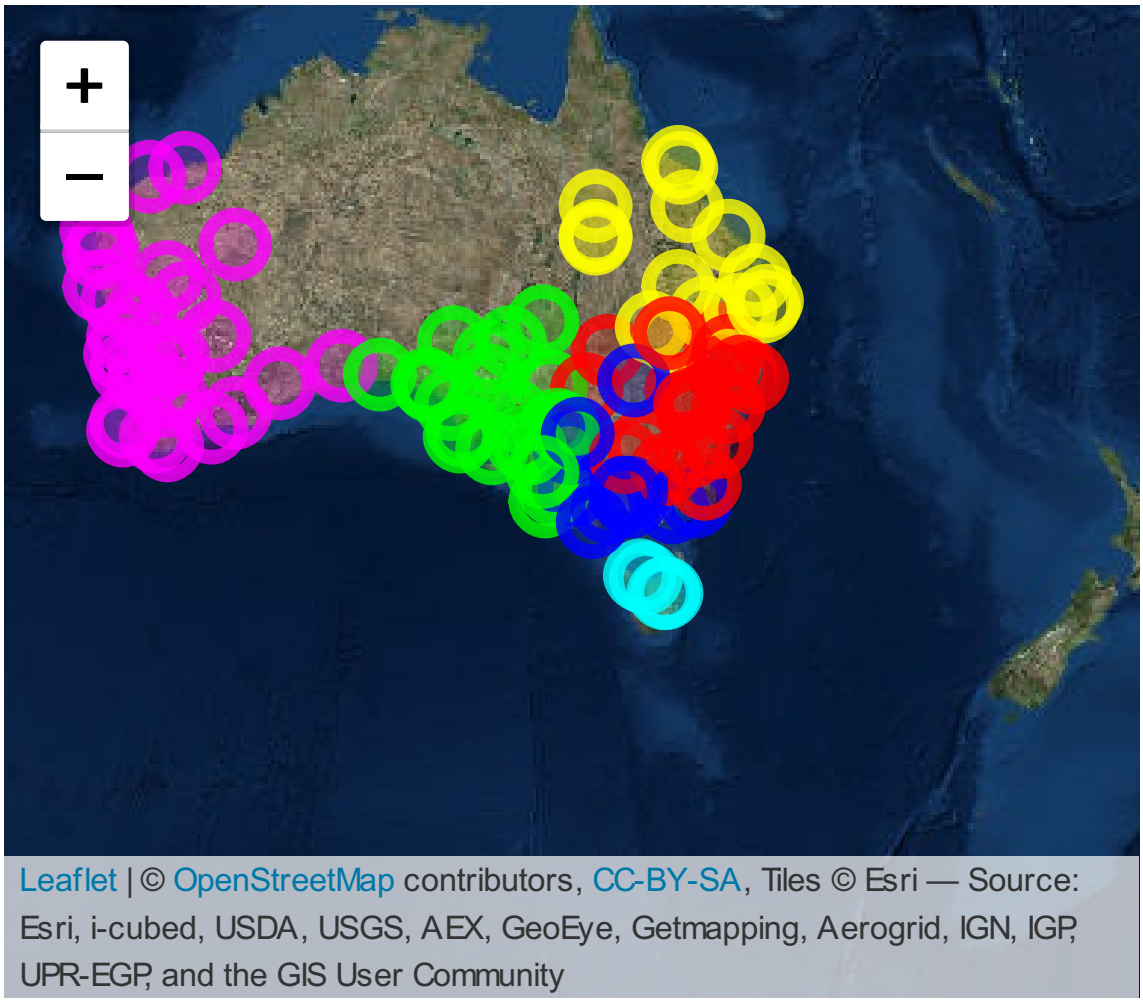
```
gl.map.interactive(foxes.gl)
```



For the map style you need to change the provider string. For example to change to a aerial photo map you could use [7].

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

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



5. Landscape genetics

The idea of a landscape genetic analysis is that genetic similarity is between individuals/populations is dependent on the distance between individuals and [potentially] on the "resistance" of the landscape between individuals/populations.

For this example we first load a data set called possums, which is already in genlight format.

```
possums <- readRDS("./data/scratch/gsa/possums.rdata")
```


Task

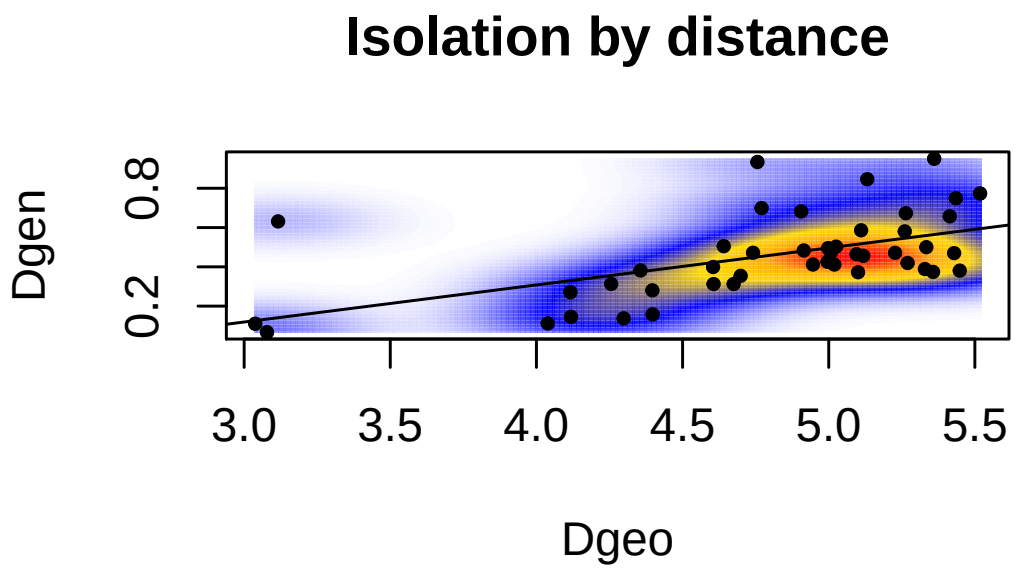
1. Study the possum genlight object (how many individuals per population)
2. Overall how many loci are in the data set?

Isolation by distance

The "null model" in landscape genetics is that there is a simple relationship between genetic and euclidean distance. A standard procedure is to study the relationship between log(euclidean distance) and Fst/1-Fst (see ?gl.ibd for details). For a quick check we can use the `gl.ibd`. To be able to use the function the genlight object needs to have the coordinates for each individual in the `@other$latlong` slot. Further we need to provide information if the coordinates are already projected or given in lat/lon.

```
iso <- gl.ibd(possums, projected = TRUE)
```

```
## Standard analysis performed on the genlight object. Mantel test and plot will be Fst/1-Fst versus log(dist
## Coordinates not transformed. Distances calculated on the provided coordinates.
## Mantel statistic based on Pearson's product-moment correlation
##
## Call:
## mantel(xdis = Dgen, ydis = Dgeo, permutations = 999, na.rm = TRUE)
##
## Mantel statistic r: 0.5513
##       Significance: 0.001
##
## Upper quantiles of permutations (null model):
##   90%   95% 97.5%   99%
## 0.225 0.278 0.338 0.394
## Permutation: free
## Number of permutations: 999
```



The function returns a list of the following components: Dgen (the genetic distance matrix), Dgeo (the Euclidean distance matrix), mantel (the statistics of the mantel test)

A mantel test is basically a simple regression, but the significance takes the non-independence of pairwise distances via a bootstrap approach into account.

Landscape genetics using a landsacpe restistance approach

Often ecologists want to know if a particular landscape feature is affecting population structure on top of Euclidean distance. The idea is that a particular feature is causing some cost for individuals when moving through it, hence modifying the actual euclidean distance between individuals/populations. Commonly used approaches to calculate so-called cost-distances are the "least-cost" and "circuitscape" approach. Both approaches require a landscape that represents landscape features in terms of the "resistance" values

Calculation of cost distances

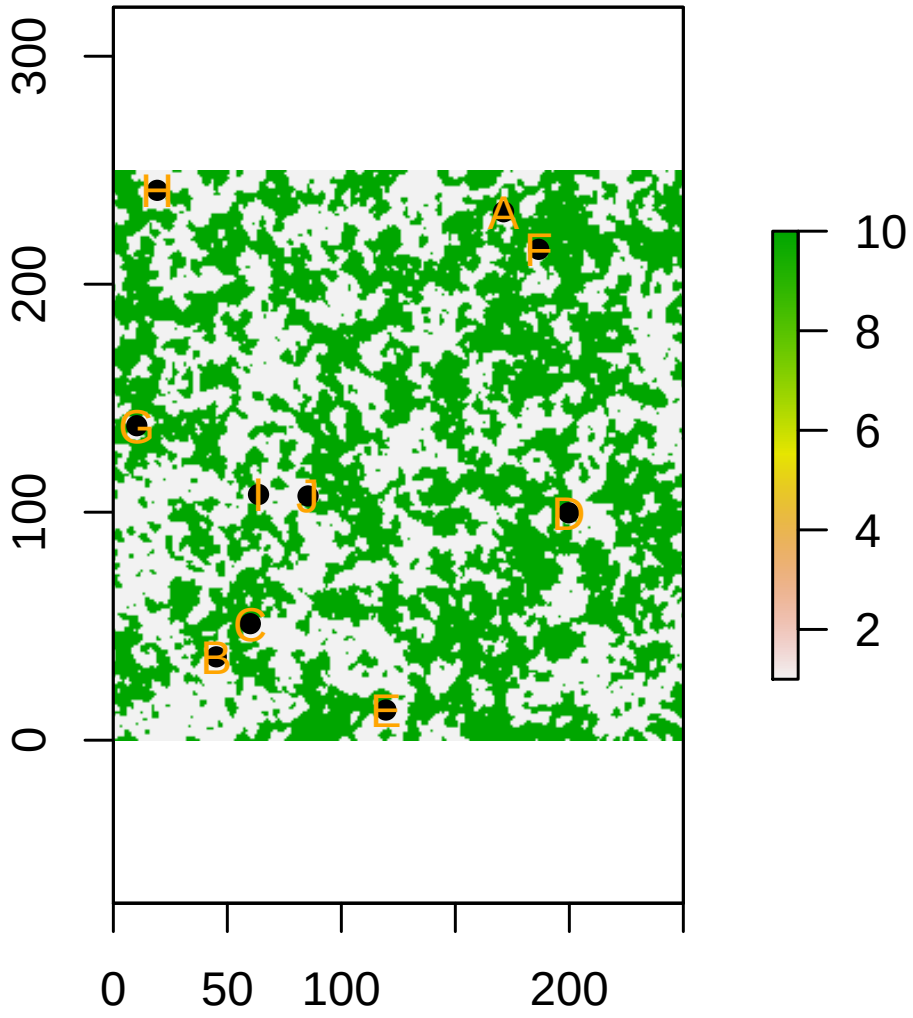
In this example we use populations as the entity of interest. Hence we need to calculate three distance matrices, namely a Euclidean distance matrix, a cost distance matrix and finally a genetic distance matrix. The two distance matrices can then be used (very similar to the partial mantel test above) to compete against each other how well they explain genetic distances. As mentioned we base our analysis on indviduals, therefore we first need to calculate the coordinates of our population centers. But first we load our (resistance) landscape.

```
landscape <- readRDS(("/data/scratch/GSA/landscape.rdata"))
```

We calculate the population centers via:

```
xs <- tapply(possums@other$latlong[, "lon"], pop(possums),
    mean)
ys <- tapply(possums@other$latlong[, "lat"], pop(possums),
    mean)

plot(landscape)
points(xs, ys, pch = 16)
text(xs, ys, popNames(possums), col = "orange")
```




```
coords <- cbind(xs, ys)
```

Euclidean distance

```
eucl <- as.matrix(dist(coords))
```

Costdistances

```
library(PopGenReport)
cost <- costdistances(landscape = landscape, locs = coords,
    method = "leastcost", NN = 8)
```

genetic distance

For simplicity we will use pairwise Fsts between population here

```
library(StAMPP)
```

```
## Loading required package: pegas
```

```
## Loading required package: ape
```

```
##
## Attaching package: 'ape'
```

```
## The following objects are masked from 'package:raster':
##
##     rotate, zoom
```

```
##
## Attaching package: 'pegas'
```

```
## The following object is masked from 'package:ape':
##
##     mst
```

```
## The following object is masked from 'package:ade4':
##
##     amova
```

```
gd <- as.matrix(as.dist(stamppFst(possums, nboots = 1)))
```

And finally run a partial mantel test

```
wassermann(gen.mat = gd, eucl.mat = eucl, cost.mats = list(cost = cost),
    plot = F)
```

```
## $mantel.tab
##                    model       r     p
## 1 Gen ~cost | Euclidean  0.4945 0.033
## 2 Gen ~Euclidean | cost -0.2502 0.836
```

Library PopGenReport has a convinience function that does all in once, but is less flexible Please note we need to transform the possums genlight to a genind object. It has the benefit that it shows the actual least cost path in the landscape (but runs longer).

```
pgi <- gl2gi(possums)
```

```
## Start conversion....
## 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")
##
  |
  |                                      |   0%
  |
  |                                      |   1%
  |
  |=                                     |   2%
  |
  |=                                     |   3%
  |
  |=                                     |   4%
  |
  |==                                    |   4%
  |
  |==                                    |   5%
  |
  |==                                    |   6%
  |
  |==                                    |   7%
  |
  |===                                   |   7%
  |
  |===                                   |   8%
  |
  |===                                   |   9%
  |
  |===                                   |  10%
  |
  |====                                  |  10%
  |
  |====                                  |  11%
```

```
|
|====                                   |  12%
|
|====                                   |  13%
|
|=====                                  |  13%
|
|=====                                  |  14%
|
|=====                                  |  15%
|
|=====                                  |  16%
|
|======                                 |  16%
|
|======                                 |  17%
|
|======                                 |  18%
|
|=======                                |  19%
|
|=======                                |  20%
|
|=======                                |  21%
|
|========                               |  22%
|
|========                               |  23%
|
|========                               |  24%
|
|=========                              |  24%
|
|=========                              |  25%
|
|=========                              |  26%
|
|=========                              |  27%
|
|==========                             |  27%
|
|==========                             |  28%
|
|==========                             |  29%
```

```
|
|==========                                |  30%
|
|==========                                |  30%
|
|==========                                |  31%
|
|==========                                |  32%
|
|==========                                |  33%
|
|===========                               |  33%
|
|===========                               |  34%
|
|===========                               |  35%
|
|===========                               |  36%
|
|============                              |  36%
|
|============                              |  37%
|
|============                              |  38%
|
|============                              |  39%
|
|============                              |  40%
|
|============                              |  41%
|
|===============                           |  42%
|
|===============                           |  43%
|
|==============                            |  44%
|
|================                          |  44%
|
|================                          |  45%
|
|================                          |  46%
|
|================                          |  47%
```

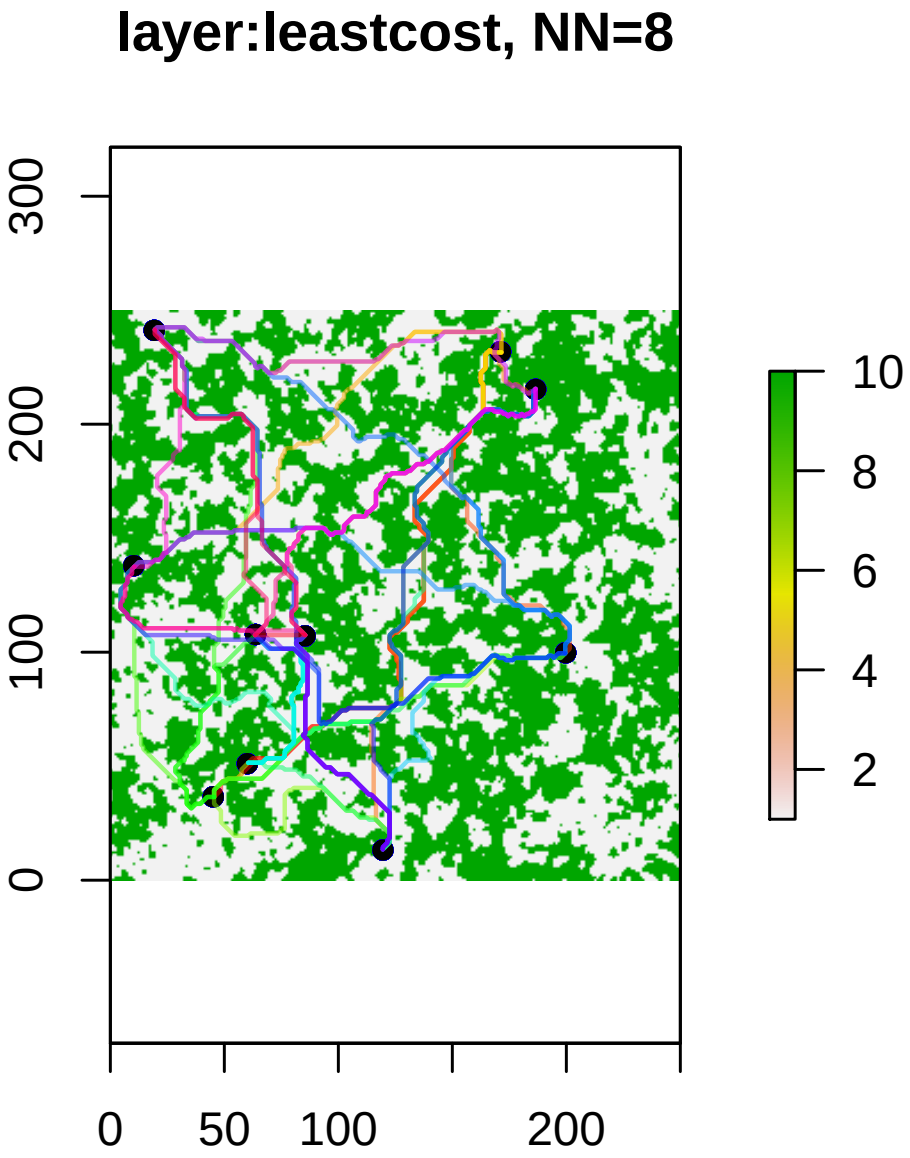
```
|
|================                  |  47%
|
|================                  |  48%
|
|=================                 |  49%
|
|=================                 |  50%
|
|==================                |  50%
|
|==================                |  51%
|
|==================                |  52%
|
|==================                |  53%
|
|===================               |  53%
|
|===================               |  54%
|
|===================               |  55%
|
|===================               |  56%
|
|====================              |  56%
|
|====================              |  57%
|
|====================              |  58%
|
|=====================             |  59%
|
|=====================             |  60%
|
|====================              |  61%
|
|======================            |  62%
|
|======================            |  63%
|
|======================            |  64%
|
|======================            |  64%
```

```
|
|======================               |  65%
|
|======================               |  66%
|
|======================               |  67%
|
|=======================              |  67%
|
|=======================              |  68%
|
|=======================              |  69%
|
|=======================              |  70%
|
|========================             |  70%
|
|========================             |  71%
|
|========================             |  72%
|
|========================             |  73%
|
|=========================            |  73%
|
|=========================            |  74%
|
|=========================            |  75%
|
|=========================            |  76%
|
|==========================           |  76%
|
|==========================           |  77%
|
|==========================           |  78%
|
|===========================          |  79%
|
|===========================          |  80%
|
|===========================          |  81%
|
|============================         |  82%
```

```
|
|============================       |  83%
|
|============================       |  84%
|
|=============================      |  84%
|
|=============================      |  85%
|
|=============================      |  86%
|
|=============================      |  87%
|
|==============================     |  87%
|
|==============================     |  88%
|
|==============================     |  89%
|
|==============================     |  90%
|
|===============================    |  90%
|
|===============================    |  91%
|
|===============================    |  92%
|
|===============================    |  93%
|
|================================   |  93%
|
|================================   |  94%
|
|================================   |  95%
|
|================================   |  96%
|
|================================== |  96%
|
|================================== |  97%
|
|================================== |  98%
|
|==================================|  99%
```

```
  |
  |==================================| 100%
## Matrix converted.. Prepare genind object...
##
## Finished! Took 0 seconds.
```

```
glc <- genleastcost(pgi, fric.raster = landscape,
    gen.distance = "D", NN = 8, pathtype = "leastcost")
```


layer:leastcost, NN=8
300
200
100
0
0
50
100
200
10
8
6
4
2

6. Export data set to other formats (13:15-13:45)

Send to a friend

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

mygl <- readRDS("gl.rds")
```

All export functions start with gl2….

function	explanation
gl2fasta	Concatenates DArT trimmed sequences and outputs a fastA file
gl2shp	creates a shp file to be used with ArcGIS and the like
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

Fasta file

check methods

```
gl2fasta(testset.gl[1:5, 1:7], method = 1)
```

FastSTRUCTURE

STRUCTURE is one of the most widely used population analysis tools that allows researchers to assess patterns of genetic structure in a set of samples (Porras-Hurtado et al., 2013). STRUCTURE is freely available software for population analysis (Pritchard et al., 2000). STRUCTURE analyses differences in the distribution of genetic variants amongst populations and places individuals into groups where they share similar patterns of variation. STRUCTURE both identifies populations from the data and assigns individuals to those populations. FastSTRUCTURE is an improved implementation to analyse large quantities of data (Raj et al., 2014).

To generate an input file for fastSTRUCTURE, use the function (this format can also be used for input to STRUCTURE, though the meta data options are not supported yet):

```
gl2faststructure(gl, outfile = file.path(tempdir(),
    "myfile.fs"), probar = FALSE)
```