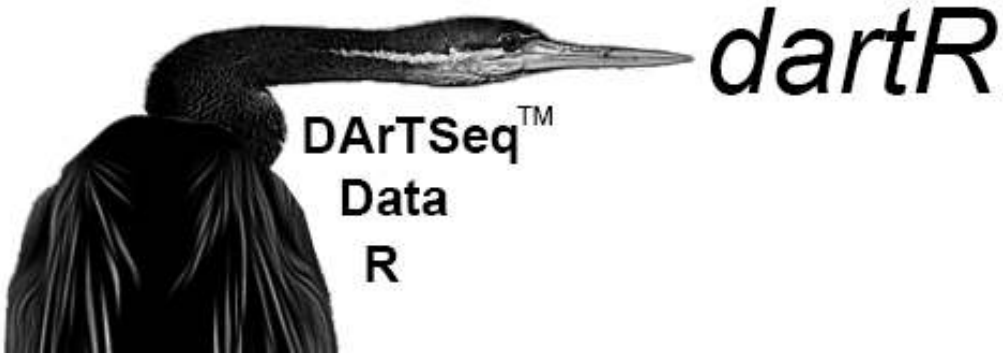


Introduction to dartR

Bernd Gruber, Peter Unmack, Oliver Berry & Arthur Georges

2020-09-01

Contents

#What is dartR?

Package `dartR` is an R package for a) loading DArT ™ SNP and SilicoDArT data generated from the commercial service provided by Diversity Arrays Technology Pty Ltd; (b) applying filters to those data based on locus metadata such as call rate, information content or reproducibility; (c) assigning individuals to populations and selecting subsets of individuals or populations; (d) visualization using Principal Coordinates Analysis (PCoA), (e) initial calculation of indices such as heterozygosity and Fst ; and (f) providing a conduit to a range of standard data formats and R packages for analysis.

Provided the data are in the `genlight` format (package `adegenet`), the package can be used to analyse SNP data from sources other than DArT ™. Please note we provide give a detailed example below and outline several ways how to import data from other formats into `genlight`.

In most cases, the scripts in {dartR} are wrappers for scripts included in other already available packages, to provide easy access to these packages for analyzing DArT data, and to provide some enhanced output diagnostics. Relatively few scripts provide novel analyses. We make no apologies for this, as the objective of `dartR` is to provide fundamental tools for accessing and manipulating SNP data in preparation for analysis by the vast suite of packages available in R through the CRAN repository, and make them easily accessible for users not deeply accustomed with the R language, without "re-inventing the wheeel".

A summary of the capabilities of `dartR` is as follows:

- Intelligent interpretation and input of DArT comma-delimited files to a compact genlight form of the R {adegenet} package.
- Filtering loci and individuals on criteria drawn from the DArT locus metadata (such as repAvg, AvgPIC) or on computed statistics (such as call rate or Hamming distance).
- Relabelling individuals and recoding populations into new aggregations, and deleting selected individuals or populations.
- Visualization using Principal Coordinates Analysis (PCoA), Neighbour-joining trees and Isolation-by-distance analysis.
- Translation to other R packages (e.g. NewHybrids), to other {adegenet} objects (e.g. genind), and to standard data formats (e.g. fastA).
- A few specific analyses not available elsewhere (e.g. fixed difference analysis, assignment testing, fast calculation or linkage disquilibria per population).

1 R installation

This tutorial will not provide information how to use R. For that please search the web using the keywords "R tutorial introduction beginner" and you will find many excellent resources. To be able to run the code in this tutorial you first need to download and install R via https://www.r-project.org/. Once installed we strongly recommend to download and install R-studio via https://www.rstudio.com/, which is becoming the defacto GUI for R. When using different packages you may get warning messages about packages built under different versions of R, these warnings can generally be ignored. If you are unfamiliar with using R there are a few important issues to be aware of. Commands and file names are case sensitive, when providing a path only use single forward slashes "/" (even if your operating system uses back slashes).

#Using this Vignette

To use this vignette interactively, you need to install and load the `dartR` package, and set the default directory using setwd(). To install the latest developmental version of the `dartR` package visit the github page for instructions (https://github.com/green-striped-gecko/dartR). Then we make the package available into the current project via `library(dartR)`. A brief description on the installation procedure is given below. There are two ways to install `dartR`, either via CRAN (prefered for the standard user) or via Github (latest, but untested version).

1.1 Installing dartR from CRAN

The latest stable version of dartR can be installed from the CRAN repository. Simply type:

```
install.packages("dartR")
```

1.2 Manual installation via Github (with all dependencies)

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

1.3 Testing the installation

To test if your installation was successful include the library by typing:

```
 library(dartR)
```

```
## Loading required package: adegenet

## Loading required package: ade4

## Registered S3 method overwritten by 'spdep':
##   method   from
##   plot.mst ape

##
##    /// adegenet 2.1.3 is loaded ////////////
##
##    > overview: '?adegenet'
##    > tutorials/doc/questions: 'adegenetWeb()'
##    > bug reports/feature requests: adegenetIssues()

## Loading required package: ggplot2

## Registered S3 method overwritten by 'pegas':
##   method      from
##   print.amova ade4

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2

## Registered S3 method overwritten by 'genetics':
##   method      from
##   [.haplotype pegas
```

If some packages have not been loaded, an error message will be given and you will need to install those package manually (see a detailed description of the process at the Github repository. A common problem is also that a package from bioiconductor is not installed (SNPRelate or qvalue). To fix this problem type:

```
BiocManager::install(c("SNPRelate", "qvalue"))
```

1.4 Setting the working directory

Once installed and invoked into your current R session, we need to prepare our session, so it points to a working directory. The working directory is the location on your hard drive, that holds your data and also is where intermediate and final output files will be exported if not directed otherwise.

To set the default directory, use the following with the appropriate directory specified:

```
# Set the default working directory (change this to suit)
setwd("c:/your.working.directory/")
```

Note that the file specification uses forward slashes, irrespective of which operating system you use (Linux, Windows or MacOS).

1.5 Test datasets

The dartR library contains test datasets that form the basis of the commands and exercises below. We recommend that you rename the test data set to `gl` in case you want to make alterations to it. If you wish to run the vignette code with your own data, simply rename your data set to `gl` after you have loaded it into R.

```
# Rename the test genlight object to gl, something simple
gl <- testset.gl
```

When working through the vignette, you should try the commands to see if you can replicate the output.

2 Genlight Format

The R package `dartR` relies on the SNP data being stored in a compact form using a bit-level coding scheme. SNP data coded in this way are held in a genlight object that is defined in R package `adegenet` (Jombart, 2008; Jombart and Ahmed, 2011). Refer to the tutorial prepared by Jombart and Collinson (2015) called *Analysing genome-wide SNP data using adegenet 2.0.0*, if you require further information.

The complex storage arrangement of genlight objects is hidden from the user because it is accompanied by a number of accessors. These allow the data to be accessed in a way similar to the manipulation of standard objects in R, such as lists, vectors and matrices.

A genlight object can be considered to be a matrix containing the SNP data encoded in a particular way. The matrix entities (rows) are the individuals, and the attributes (columns) are the SNP loci. In the body of this individual x locus matrix are the SNP data, coded as 0 for homozygous reference state, 1 for heterozygous, and 2 for homozygous alternate (or SNP) state. You can access these data by converting to a standard matrix using

```
m <- as.matrix(gl)
```

This function allows normal R approaches for examination and manipulation. For example,

```
as.matrix(gl)[1:5,1:3]
```

```
##           100049687-12-C/T 100049698-16-G/A 100049728-23-A/G
## AA010915                 2               NA                 0
## UC_00126                 2               NA                 0
## AA032760                NA               NA                 0
## AA013214                 2               NA                 0
## AA011723                 2               NA                 0
```

displays the SNP states for 5 rows [individuals] and 3 columns [loci]. Associated with the SNP genotypes in the genlight object is a `data.frame` of locus metadata (e.g. CloneID, CallRate, TrimmedSequence, etc). Each item in this list (loc.metrics) is a vector of length equal to the number of loci. Also associated with the SNP genotypes is a `data.frame` of individual metadata (individual id, population, sex, lat(itude), lon(gitude), etc) Each item in this `data.frame` (ind.metrics) is a vector of length equal to the number of individuals.

The information in the genlight object can be accessed using the following `adegenet` accessors:

- nInd(gl): | returns the number of individuals in the genlight object.
- **nLoc(gl)**: returns the number of loci.
- **nPop(gl)**: returns the number of populations to which the individuals are assigned.
- **indNames(gl)**: returns or sets labels for individuals.
- **locNames(gl)**: returns or sets labels for loci.
- **alleles(gl)**: returns or sets allelic states of each locus for each individual (e.g. "A/T").
- **ploidy(gl)**: returns or sets the ploidy of the individuals (normally diploid or 2).
- **pop(gl)**: returns or sets the population to which each individual belongs. Try also levels(pop(gl)) for a list of unique population names.
- **NA.posi**: returns the loci with missing values, that is, loci for which a sequence tag failed to amplify for each individual.
- **chr**: returns or sets the chromosome for each locus.
- **position**: returns or sets the position of each SNP in the sequence tag of each locus.
- **other(gl)**: returns of sets miscellaneous information stored as a list.


Task

Try some of these using commands on the R console

An alternative way to write these accessors is to use the form of gl@pop or gl@other, for example.

Some simple operations such as computing allele frequencies or diagnosing missing values can be problematic without representing the full dataset in memory. The package {adegenet} implements a few procedures that perform such basic tasks on genlight objects, processing one individual at a time, thereby minimizing memory requirements.

- **glSum**: counts the frequency of the alternate allele for each locus.
- **glNA**: counts the number of missing values for each locus.
- **glMean**: computes the relative frequency (a proportion in the range 0-1) of the second allele for each locus.
- **glVar**: computes the variance of the allele frequency distribution for each locus.
- **glDotProd**: computes the dot products between all pairs of individuals, with centering and scaling.

in each case taking advantage of the 0, 1, 2 coding of the SNP states for each locus - the coding essentially corresponds to the frequency of the alternate allele.

 Task

Try some of these using commands on the R console

 Hint

For further information on using genlight objects for SNP datasets, refer to the tutorial prepared by Thibaut Jombart and Caitlin Collinson (2015) entitled Analysing genome-wide SNP data using adegenet 2.0.0.

3 DArT Input Data Formats

Diversity Arrays Technology Pty Ltd (DArT™) supply 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.

 Hint

Refer to the DArT documentation provided with your report for further information.

4 Reading DArT Files into a Genlight Object

SNP data can be read into a genlight object using 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 cloneID 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. An example of the function used to input data is as follows:

```
gl <- gl.read.dart(filename = "testset.csv", ind.metafile = " ind_metrics.csv")
```

The `filename` specifies the csv file provided by Diversity Arrays Technology, and the `covfilename` specifies the csv file which contains metrics associated with each individual (id, pop, sex, etc).

Using the example data set provided in the package (accessed via an internal path to the files)

```
dartfile <- system.file("extdata","testset_SNPs_2Row.csv", package="dartR")

metafile <- system.file("extdata","testset_metadata.csv", package="dartR")
gl <- gl.read.dart(filename=dartfile, ind.metafile = metafile, probar=FALSE)
```

```
## Starting gl.read.dart
## Starting utils.read.dart
##   Topskip not provided. Setting topskip to 3 .
##   Reading in the SNP data
##   Detected 2 row format.
## Added the following locus metrics:
## AlleleID CloneID AlleleSequence SNP SnpPosition CallRate OneRatioRef OneRatioSnp TrimmedSequence 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:/Users/s425824/AppData/Local/Te
## Completed: utils.read.dart
## Starting utils.dart2genlight
## Starting 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")
## Adding individual metrics: C:/Users/s425824/AppData/Local/Temp/Rtmpk5JfRw/Rinst2b6c348a8f/dartR/extda
##   Ids for individual metadata (at least a subset of) are matching!
##   Found  250 matching ids out of 250 ids provided in the ind.metadata file.
##  Added population assignments.
##   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.
## Completed: utils.dart2genlight
##  Data read in. Please check carefully the output above
##   Read depth calculated and added to the locus metrics
##   Minor Allele Frequency (maf) calculated and added to the locus metrics
## Completed: gl.read.dart
```

The resultant genlight object, `gl`, can be interrogated to determine if the data have been input correctly.

To dispay (parts of) the genlight object we have the following options:

Display the structure of the genlight object

```
gl
```

```
##  /// GENLIGHT OBJECT /////////
##
##  // 250 genotypes,  255 binary SNPs, size: 749.5 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
##    @pop: population of each individual (group size range: 1-11)
##    @other: a list containing: loc.metrics  latlong  ind.metrics  loc.metrics.flags  verbose  history
```

Display the SNP genotypes for the first 3 individuals and 5 loci

```
as.matrix(gl)[1:3,1:3]
```

```
##          100049687-12-C/T 100049698-16-G/A 100049728-23-A/G
## AA010915                2               NA                0
## UC_00126                2               NA                0
## AA032760               NA               NA                0
```

Report the number of loci, individuals and populations

```
nLoc(gl)
```

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

```
nInd(gl)
```

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

```
nPop(gl)
```

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

List the population labels (only the first 5)

```
levels(pop(gl))[1:5]
```

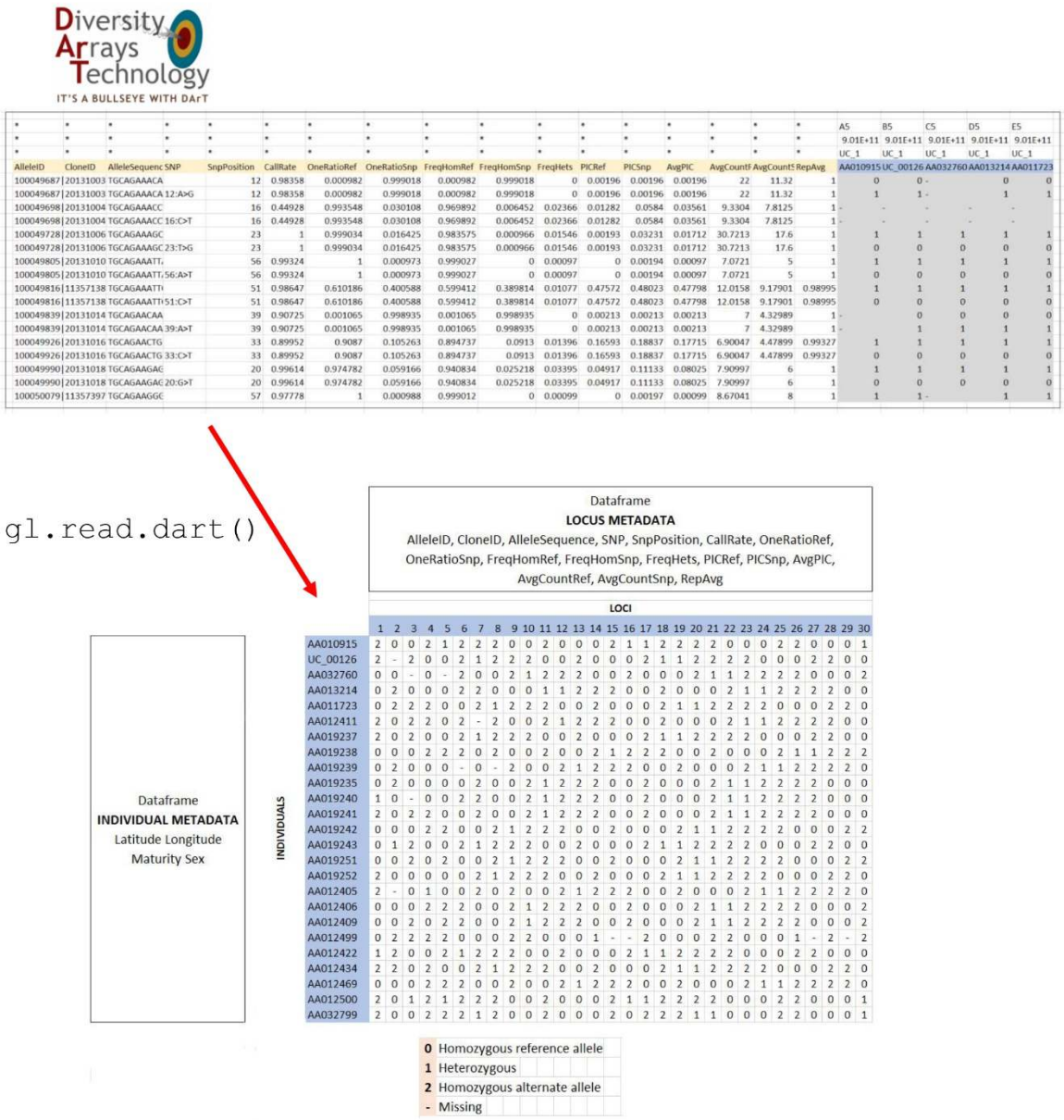
```
## [1] "EmmacBrisWive" "EmmacBurdMist" "EmmacBurnBara" "EmmacClarJack"
## [5] "EmmacClarYate"
```

To load your own data in, you need the csv file provided by DArT (e.g. mydata.csv) and (optional) an individual metadata file (e.g. my.metadata.csv).

```
gl <- gl.read.dart(filename="mydata.csv", ind.metafile = "my.metadata.csv")
```

5 Working with DArT Genlight Objects

Genlight objects for working with DArT genotypes have all of the general attributes described above, but also have some specific characteristics that derive from the DArT preparatory filtering and associated quality statistics. This is shown diagrammatically below.



5.1 Locus metadata

The locus metadata included in the genlight object are those provided as part of your DArT report. These metadata are obtained from the DArT 1Row or 2Row csv file when it is read in to the genlight object. The locus metadata are held in an R `data.frame` that is associated with the SNP data as part of the genlight object.

The locus metadata would typically include:

identifier	explanation
CloneID:	Unique identifier for the sequence in which the SNP marker occurs

identifier	explanation
SNP:	In 2-row format, this column is blank in the Reference row, and contains the base position and base variant details in the SNP row. In 1-row format, this column contains the base position and base variant details
SnpPosition:	The position (zero is position 1) in the sequence tag at which the defined SNP variant base occurs
TrimmedSequence	(optional) The sequence containing the SNP or SNPs, trimmed of adaptors.
CallRate:	The proportion of samples for which the genotype call is non-missing (that is, not "-")
OneRatioRef:	The proportion of samples for which the genotype score is "1", in the Reference allele row of the 2-row format.
OneRatioSnp:	The proportion of samples for which the genotype score is "1", in the SNP allele row of the 2-row format
FreqHomRef:	The proportion of samples homozygous for the Reference allele
FreqHomSnp:	The proportion of samples homozygous for the Alternate (SNP) allele
FreqHets:	The proportion of samples which score as heterozygous.
PICRef:	The polymorphism information content (PIC) for the Reference allele row
PICSnp:	The polymorphism information content (PIC) for the SNP allele row
AvgPIC:	The average of the polymorphism information content (PIC) of the Reference and SNP allele rows
AvgCountRef:	The sum of the tag read counts for all samples, divided by the number of samples with non-zero tag read counts, for the Reference allele row
AvgCountSnp:	The sum of the tag read counts for all samples, divided by the number of samples with non-zero tag read counts, for the Alternate (SNP) allele row
RepAvg:	The proportion of technical replicate assay pairs for which the marker score is consistent.

These metadata variables are held in the genlight object as an associated data.frame called loc.metrics, which can be accessed in the following form:

```
#Only the entries for the first ten individuals are shown
gl@other$loc.metrics$RepAvg[1:10]
```

```
##  [1] 1.000000 1.000000 1.000000 1.000000 0.989950 1.000000 0.993274 1.000000
##  [9] 1.000000 0.980000
```

You can check the names of all available loc.metrics via:

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

```
##  [1] "AlleleID"        "CloneID"         "AlleleSequence"  "SNP"
##  [5] "SnpPosition"     "CallRate"        "OneRatioRef"     "OneRatioSnp"
##  [9] "TrimmedSequence" "FreqHomRef"      "FreqHomSnp"      "FreqHets"
## [13] "PICRef"          "PICSnp"          "AvgPIC"          "AvgCountRef"
## [17] "AvgCountSnp"     "RepAvg"          "clone"           "uid"
## [21] "rdepth"
```

Depending on the report from DarT you may have additional (fewer) loc.metrics (e.g. Trimmed Sequence is available on request).

These metadata are used by the {dartR} package for various purposes, so if any are missing from your dataset, then there will be some analyses that will not be possible. For example, TrimmedSequence is

used to generate output for subsequent phylogenetic analyses that require estimates of base frequencies and transition and transversion ratios.

CloneID is essential (with its very special format), and `dartR` scripts for loading your data sets will terminate with an error message if this is not present.

5.2 Individual Metadata

Individual (=specimen or sample) metadata are user specified, and do not come from DArT. Individual metadata are held in a second dataframe associated with the SNP data in the genlight object. See the figure above.

Two special individual metrics are:

individual metric	explanation
id	Unique identifier for the individual or specimen that links back to the physical sample
pop	A label for the biological population from which the individual was drawn

These metrics are supplied by the user by way of a metafile, provided at the time of inputting the SNP data to the genlight object. A metafile is a comma-delimited file, usually named ind_metrics.csv or similar, that contains labelled columns. The file must have a column headed id, which contains the individual (=specimen or sample labels) and a column headed pop, which contains the populations to which individuals are assigned.

These special metrics can be accessed a:

`gl@pop` or `pop(gl)`

and

`gl@ind.names` or `indNames(gl)`

A number of other user-defined metrics can be included in the metadata file. Examples of user-defined metadata for individuals include:

metric	explanation
sex	Sex of the individual (Male, Female)
maturity	Maturity of the individual (Adult, Subadult, juvenile)
lat	Latitude of the location of collection
long	Longitude of the location of collection

These optional data are provided by the user in the same metafile used to assign id labels and assign individuals to populations. It is the excel csv file referred to above.

The individual metadata are held in the genlight object as a dataframe named ind.metrics and can be accessed using the following form:

```
#only first 10 entries shoums
gl@other$ind.metrics$sex[1:10]
```

```
##  [1] Male    Male    Male    Male    Unknown Male    Female  Female  Male
## [10] Female
## Levels: Female Male Unknown
```

5.3 Importing data from other formats/sources

In the table below we list possible avenues to load data from other formats and most importantly also from text simple files, which opens the package in principle to any other source.

Import path	Package	Pathway*	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) GENETIX
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 & Gruber (2014) (?read.genetable)

*Pathway provides the order of functions needed to convert data to genlight, — indicates that the function direclty converts to a `genlight` object

Below we exemplify an import from a text file. Provided with the package is a simple data set in a text file format (editable via Excel or Calc) or any text editor. To convert your data into a genlight object you can arrange your data set in the same format and also attach meta data for loci such as call rates, sequences and individuals (will be stored in the @other slot under), such as coordinates, lat/lons and population structure.

To have a look at the format in the provided file, type:

```
read.csv( paste(.libPaths()[1],"/dartR/extdata/platy.csv",sep="" ))
```

```
##     ind   pop       lat     long  group    age loci1 loci2 loci3 loci4 loci5
## 1  T158 Black -40.86642 145.2836 Female    juv   A/A   G/C   A/T   A/A   G/C
## 2  T306 Black -40.85589 145.2764   Male     Ad   A/A   G/G   A/A   A/A   G/C
## 3  T305 Black -40.87889 145.2885 Female     Ad   A/A   G/G   A/T   A/A   G/C
## 4  T148 Black -40.99193 145.3757   Male     Ad   A/A   G/G   A/A   A/A   G/C
## 5  T149 Black -40.99193 145.3757 Female     Ad   A/A   G/C   A/T   A/A   G/C
## 6  T106  Brid -41.23205 147.4597   Male     Ad   A/A   G/G   A/A   A/A   G/G
## 7  T107  Brid -41.23205 147.4597   Male     Ad   A/A   G/G   A/T   A/A   G/G
## 8  T110  Brid -41.23205 147.4597 Female     Ad   A/A   G/G   A/A   A/A   G/G
## 9  T111  Brid -41.23205 147.4597 Female     Ad   A/T   G/C   A/T   A/A   G/G
## 10 T308   Cam -41.09567 145.7958   Male Sub-Ad   A/T   C/C   A/A   A/A   G/G
## 11 T307   Cam -41.06975 145.8152   Male     Ad   A/T   C/C   A/T   A/A   C/C
## 12 T302   Cam -41.05121 145.8280   Male     Ad   T/T   G/C   A/A   A/A   C/C
## 13 T303   Cam -41.04764 145.8230 Female    Juv   T/T   C/C   A/T   A/A   C/C
##    loci6
## 1    T/A
```

```
## 2    T/A
## 3    T/A
## 4    T/A
## 5    T/A
## 6    T/A
## 7    T/A
## 8    T/A
## 9    T/A
## 10   T/A
## 11   T/A
## 12   T/T
## 13   A/A
```

We now have to specify each of the columns to be able to read it into R as a genind format using `read.genetable` from the `PopGenReport` package. For a detailed explanation see `?read.genetable`.

```
#you might need to install PopGenReport via
#install.packages("PopGenReport")
library(PopGenReport)
```

```
## Loading required package: knitr
```

```
platy <- read.genetable( paste(.libPaths()[1],"/dartR/extdata/platy.csv",
sep="" ), ind=1, pop=2, lat=3, long=4, other.min=5, other.max=6, oneColPerAll=FALSE,sep="/")

platy
```

```
## /// GENIND OBJECT /////////
##
##  // 13 individuals; 6 loci; 11 alleles; size: 13.1 Kb
##
##  // Basic content
##    @tab:  13 x 11 matrix of allele counts
##    @loc.n.all: number of alleles per locus (range: 1-2)
##    @loc.fac: locus factor for the 11 columns of @tab
##    @all.names: list of allele names for each locus
##    @ploidy: ploidy of each individual  (range: 2-2)
##    @type:  codom
##    @call: df2genind(X = genes, sep = sep, ncode = ncode, ind.names = as.character(inds),
##     loc.names = colnames(genes), pop = as.character(pops), NA.char = NA.char,
##     ploidy = ploidy)
##
##  // Optional content
##    @pop: population of each individual (group size range: 4-5)
##    @other: a list containing: latlong  data
```

To convert the genind to a genlight object simply type:

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

```
## Starting gi2gl
## Completed: gi2gl
```

The resulting genlight object has already some of the meta data as provided in the table but not in the right place (pop, indNames ploidy, loc.n.all and latlong in the "@other"' slot are already there).

For example to put the individual meta data (group, age) into the right slot in a genlight object we need to move it into the `@other$ind.metrcis` slot.

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

If there are additional data available for the loci (e.g. via a csv file) we can also attach that via the `@other$loc.metrics` slot using `read.csv`. For demonstratation we will create some random loci meta data with the code below and store it into a `data.frame` called `df.loc`. We create two columns: Trimmed-Sequnce (containing the sequence data for each loci, trimmed to a consistent length, and removing the adaptors), and a quality index (ranging from zero to one indicating the reproducibility of repeated runs for that loci).

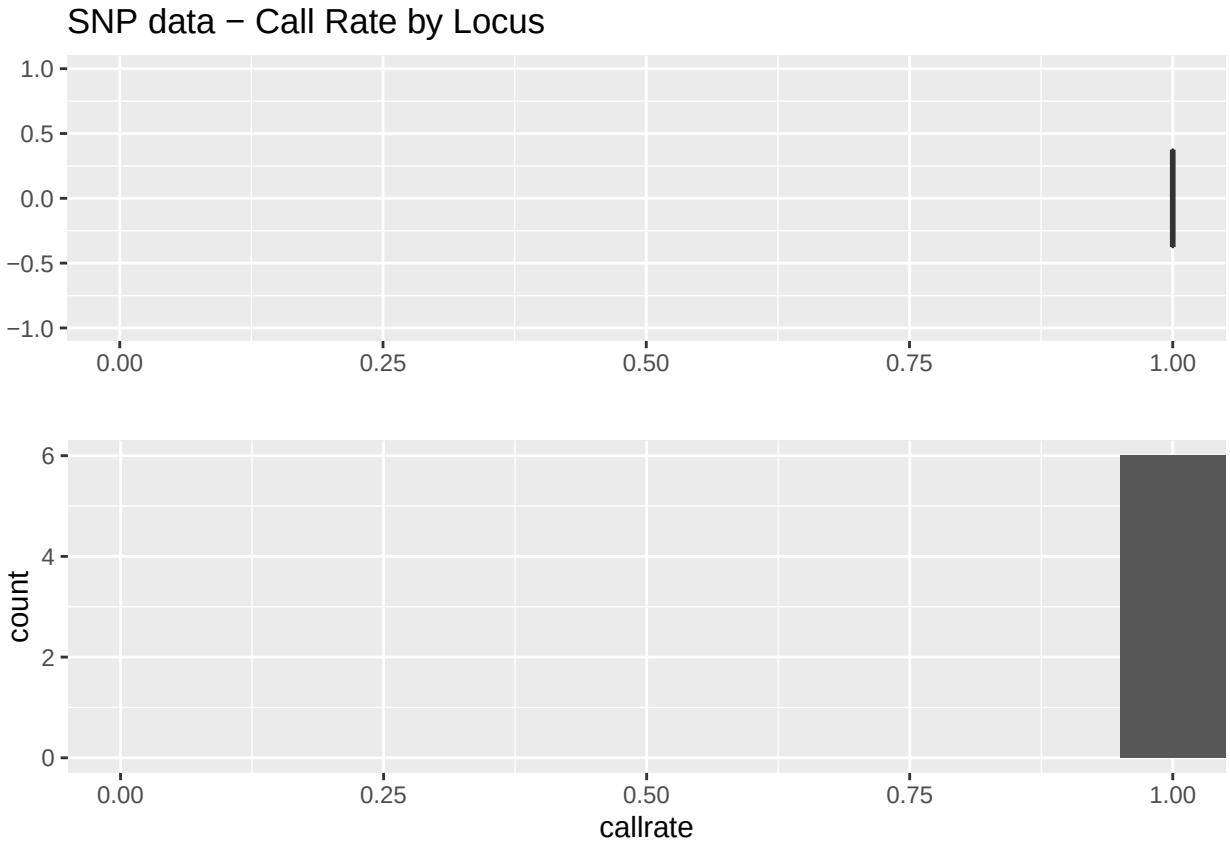
```
ts <- sapply(1:nLoc(platy.gl), function(x) paste(sample(c("A","T","G","C"), 50, replace = T),
                                                 collapse = ""))
df.loc <- data.frame(RepAvg = runif(nLoc(platy.gl)),  TrimmedSequence=ts)

platy.gl@other$loc.metrics <- df.loc
```

The last line of code stores the loci metadata in the right location and we can use our filter and report function now on the genlight object as before.

```
gl.report.callrate(platy.gl)
```

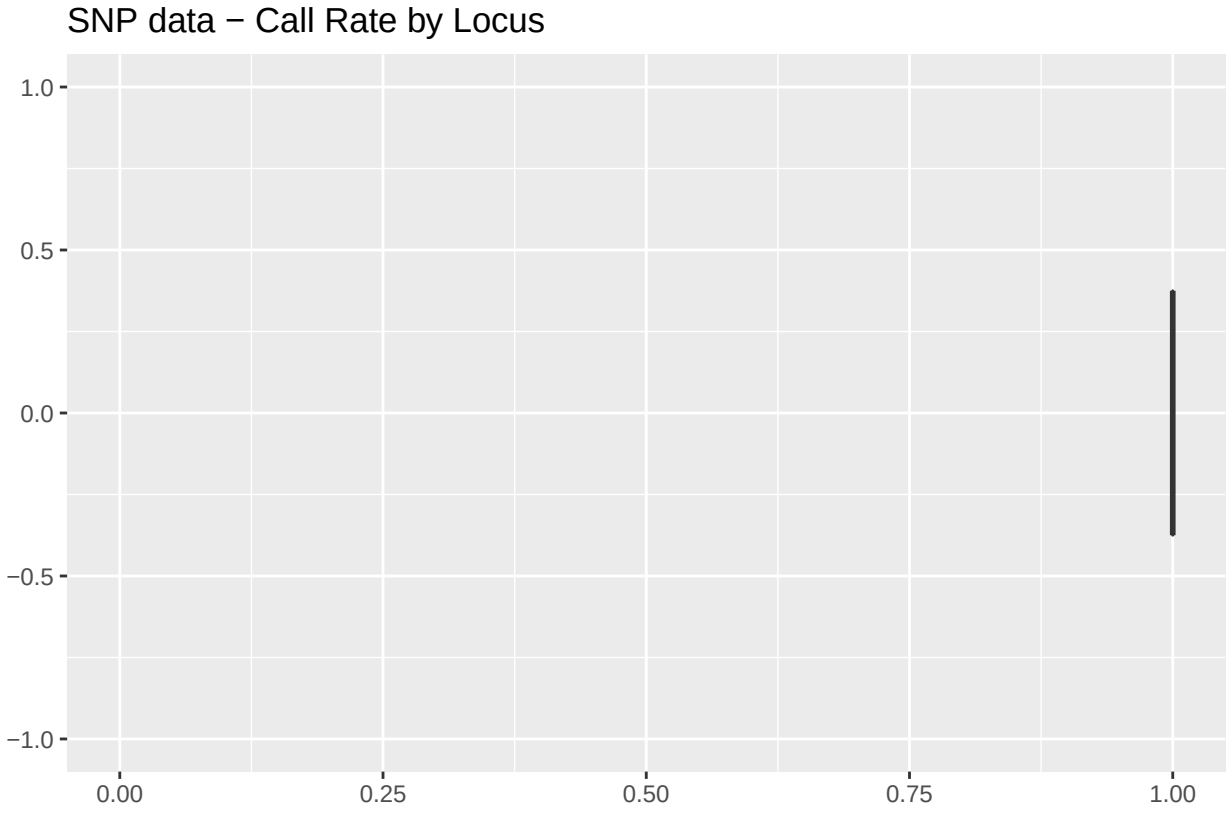
```
## Starting gl.report.callrate
##   Processing a SNP dataset
##   Warning: genlight object contains monomorphic loci which will be factored into Callrate calculatio
```



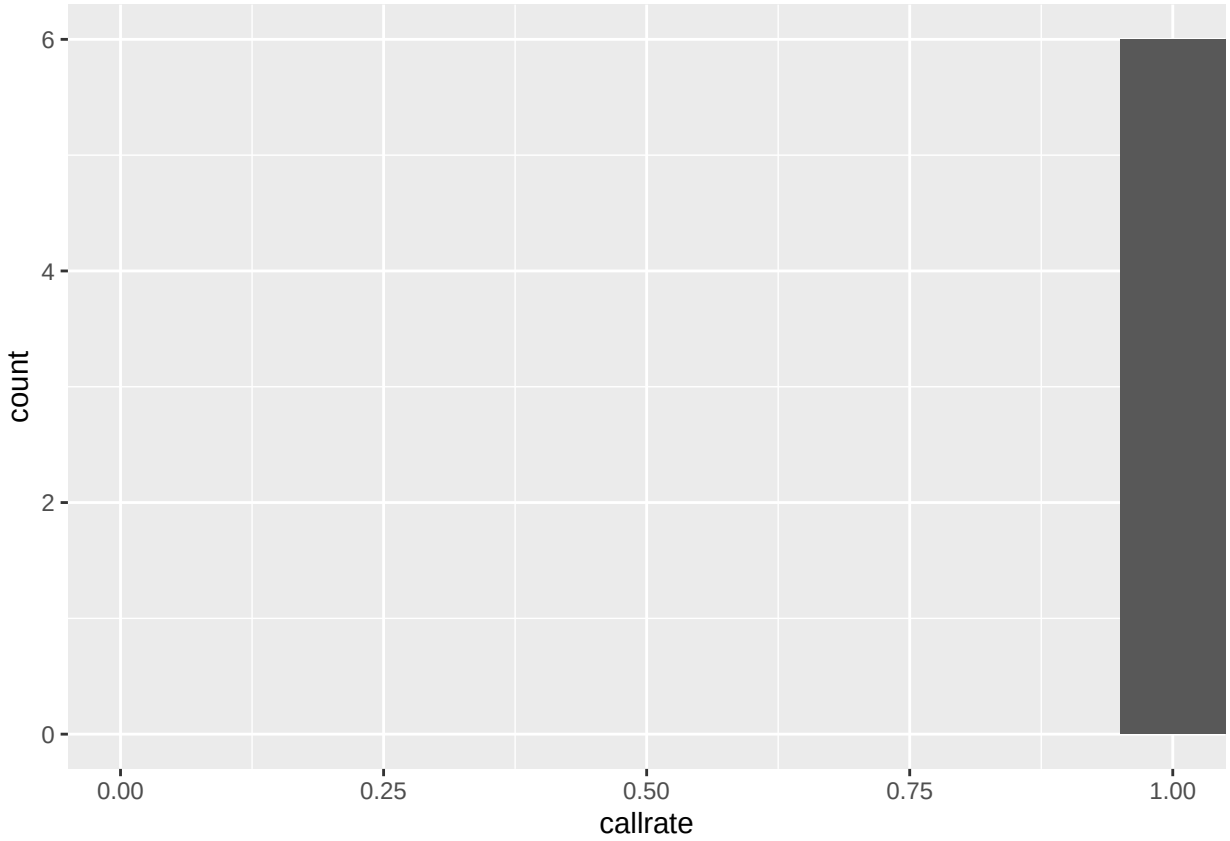

```
##   Reporting Call Rate by Locus
##   No. of loci = 6
##   No. of individuals = 13
##     Miniumum Call Rate:  1
##     Maximum Call Rate:  1
##     Average Call Rate:  1
##     Missing Rate Overall:  0
## Completed: gl.report.callrate
```

```
## $callrates
##    Threshold Retained Percent Filtered Percent
## 1       1.00        6     100        0       0
## 2       0.95        6     100        0       0
## 3       0.90        6     100        0       0
## 4       0.85        6     100        0       0
## 5       0.80        6     100        0       0
## 6       0.75        6     100        0       0
## 7       0.70        6     100        0       0
## 8       0.65        6     100        0       0
## 9       0.60        6     100        0       0
## 10      0.55        6     100        0       0
## 11      0.50        6     100        0       0
## 12      0.45        6     100        0       0
## 13      0.40        6     100        0       0
## 14      0.35        6     100        0       0
```

```
## 15      0.30        6     100        0       0
## 16      0.25        6     100        0       0
## 17      0.20        6     100        0       0
## 18      0.15        6     100        0       0
## 19      0.10        6     100        0       0
## 20      0.05        6     100        0       0
## 21      0.00        6     100        0       0
##
## $boxplot
```




```
##
## $hist
```




```
gl2 <- gl.filter.reproducibility(platy.gl, t=0.5)
```

```
## Starting gl.filter.reproducibility
##   Processing a SNP dataset
##   Identifying loci with repeatability below : 0.5
##   Removing loci with repeatability less than 0.5
## Completed: gl.filter.reproducibility
```

Some of the functions require additional meta data, e.g. the gl2fasta need sequence data (TrimmedSequnce), the position of the SNP in the sequence for each loci `@position` and the type of allelels for each loci `@loc.all`. If such data is available the information needs to be stored in the correct slots to work with the function. The code below exports our genlight object to a fasta format (again we create random meta data via the code below).

```
platy.gl@position <- as.integer(runif(nLoc(platy.gl),2,49))
platy.gl@loc.all <- testset.gl@loc.all[1:6]
```

And finally we can use the `gl2fasta` function (using method 1, see explanation in ?gl2fasta):

```
gl2fasta(platy.gl)
```

6 Subsetting and Recoding Data

6.1 Filtering

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

Refer to the help on each function for details of the parameters taken by each of these functions using `?nameoffunction`.

6.2 Examples of dartR code to filter a `gl` dataset

Filtering of data is often necessary to make sure only high quality loci (few missing data) and with a consistent quality are retained and "noise" in the data set is minimised. In addition by filtering you reduce the number of loci, which often speeds up the analysis considerably. The kind and order of filtering that someone applies depends very much on the intended analysis. For example for classical calculation of indices of population structure, loci and individuals with lots of missing data might be discarded, though for other kind of analysis the amount of missing data may hint to "hidden" subspecies in the populations. Therefore a general adivse on the order and amount of filtering cannot be given. As an example if the focus is towards studying population structure, where only a limited number of individuals are sampled, a valid strategy is to filter in such a way that the number of individuals per population are maintained, but the number of loci can be reduced. So a suggested order here is:

1. Filter by reproducibility (`gl.filter.reproducibility` 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"`)

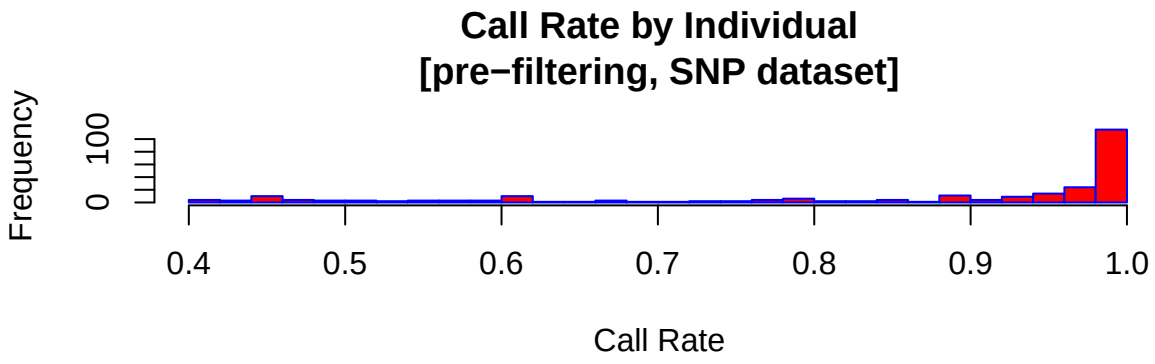
Additional filters to apply could be for excluding possible loci under selection (`gl.outflank`), checking loci for linkage disequilibrium (`gl.report.ld`) or filtering for loci out of Hardy-Weinberg-Equilibrium (`gl.filter.hwe`)

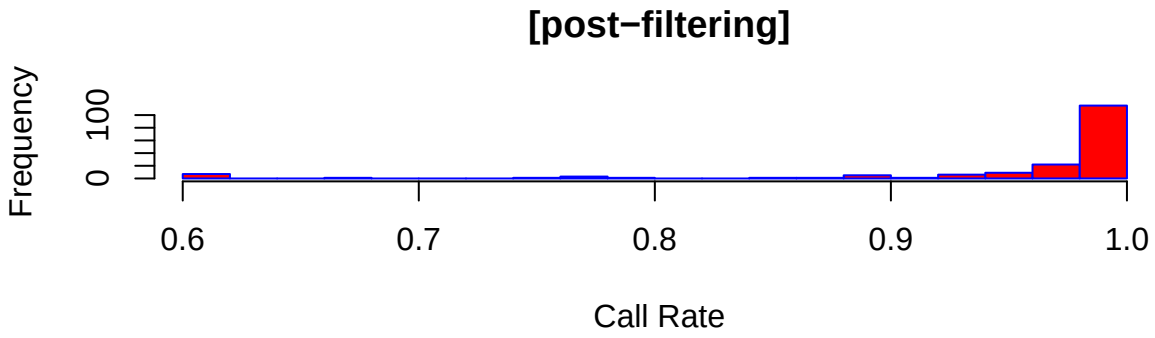
Simple examples how to apply some of the filters are provided below.

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
##   Processing a SNP dataset
##   Warning: Dataset contains monomorphic loci which will be included in the Call Rate calculations fo
##   Removing loci based on Call Rate, threshold = 0.95
```



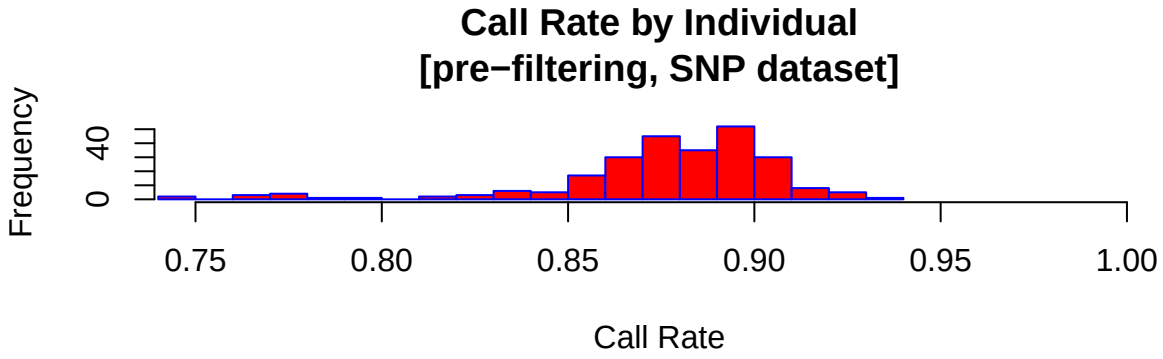


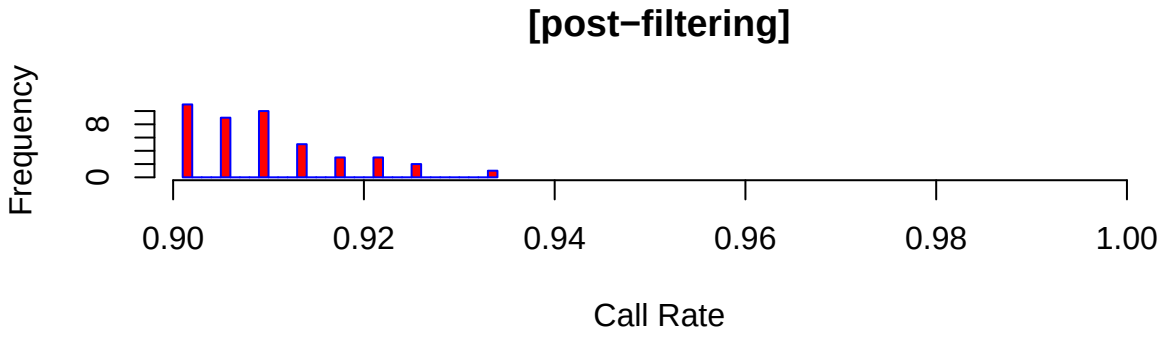

```
## Completed: gl.filter.callrate
```

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

```
gl2 <- gl.filter.callrate(gl, method="ind", threshold = 0.90)
```

```
## Starting gl.filter.callrate
##   Processing a SNP dataset
##   Warning: Dataset contains monomorphic loci which will be included in the Call Rate calculations fo
##   Removing individuals based on Call Rate, threshold = 0.9
##   No. of individuals deleted (CallRate <=  0.9 :
## AA010915[EmmacMDBForb], AA032760[EmmacMDBMaci], AA011723[EmmacBurnBara], AA012411[EmmacCoopEulb], AA
```






```
##   Note: Locus metrics not recalculated
##   Note: Resultant monomorphic loci not deleted
## Completed: gl.filter.callrate
```

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

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

```
## Starting gl.filter.reproducibility
##   Processing a SNP dataset
##   Identifying loci with repeatability below : 1
##   Removing loci with repeatability less than 1
## Completed: gl.filter.reproducibility
```

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
##   Processing a SNP dataset
##   Calculating Hamming distances between sequence tags
##   Filtering loci with Hamming Distance is less than 0.25
## Completed: gl.filter.hamming
```

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 = 0.90)
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.

7 Subsetting and Recoding Data

7.1 Population (=higher level grouping) reassignment

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"    "EmmacBurnBara"    "EmmacClarJack"
##  [5] "EmmacClarYate"    "EmmacCoopAvin"    "EmmacCoopCully"   "EmmacCoopEulb"
##  [9] "EmmacFitzAllig"   "EmmacJohnWari"    "EmmacMDBBowm"     "EmmacMDBCond"
## [13] "EmmacMDBCudg"     "EmmacMDBForb"     "EmmacMDBGwyd"     "EmmacMDBMaci"
```

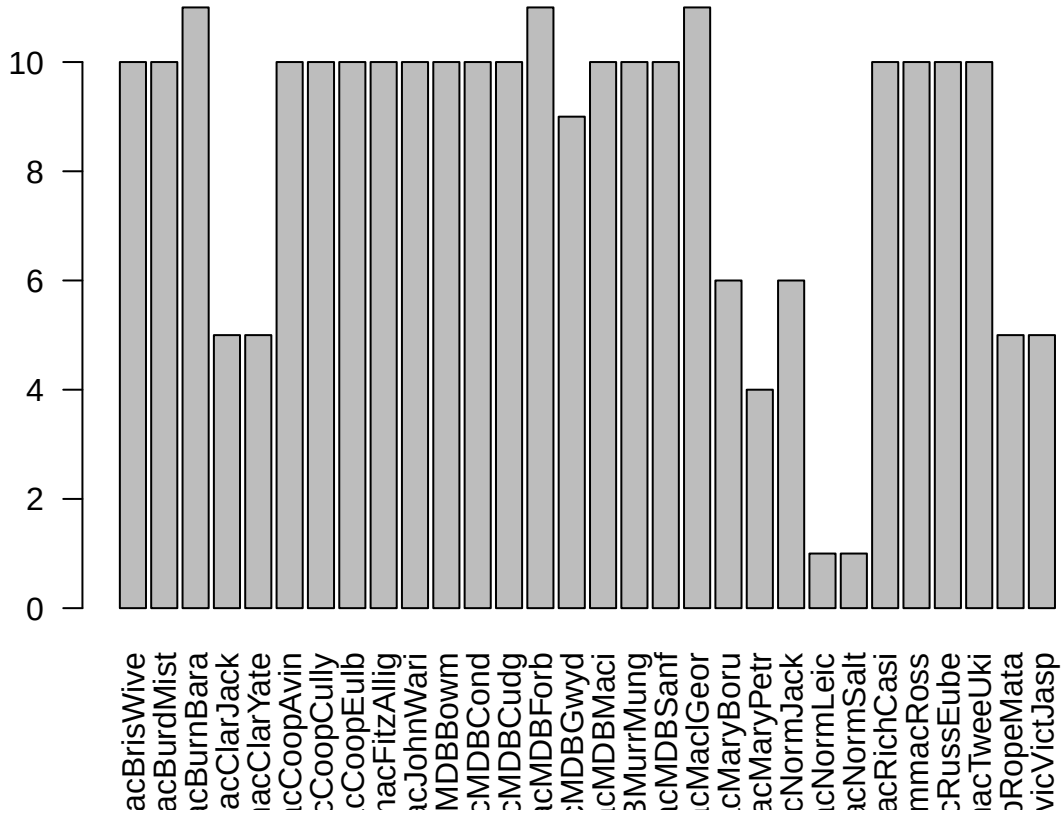
```
## [17] "EmmacMDBMurrMung" "EmmacMDBSanf"     "EmmacMaclGeor"    "EmmacMaryBoru"
## [21] "EmmacMaryPetr"    "EmmacNormJack"    "EmmacNormLeic"    "EmmacNormSalt"
## [25] "EmmacRichCasi"    "EmmacRoss"        "EmmacRussEube"    "EmmacTweeUki"
## [29] "EmsubRopeMata"    "EmvicVictJasp"
```

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

```
##
##    EmmacBrisWive    EmmacBurdMist    EmmacBurnBara    EmmacClarJack
##               10               10               11                5
##    EmmacClarYate    EmmacCoopAvin   EmmacCoopCully    EmmacCoopEulb
##                5               10               10               10
##   EmmacFitzAllig    EmmacJohnWari     EmmacMDBBowm     EmmacMDBCond
##               10               10               10               10
##     EmmacMDBCudg     EmmacMDBForb     EmmacMDBGwyd     EmmacMDBMaci
##               10               11                9               10
## EmmacMDBMurrMung     EmmacMDBSanf    EmmacMaclGeor    EmmacMaryBoru
##               10               10               11                6
##    EmmacMaryPetr    EmmacNormJack    EmmacNormLeic    EmmacNormSalt
##                4                6                1                1
##    EmmacRichCasi        EmmacRoss    EmmacRussEube     EmmacTweeUki
##               10               10               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.

7.2 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:

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

```
##   [1] "AA010915"  "UC_00126"  "AA032760"  "AA013214"  "AA011723"  "AA012411"
##   [7] "AA019237"  "AA019238"  "AA019239"  "AA019235"  "AA019240"  "AA019241"
##  [13] "AA019242"  "AA019243"  "AA019251"  "AA019252"  "AA012405"  "AA012406"
##  [19] "AA012409"  "AA012499"  "AA012422"  "AA012434"  "AA012469"  "AA012500"
##  [25] "AA032799"  "AA032826"  "AA010795"  "AA010796"  "AA032800"  "AA032801"
##  [31] "AA032808"  "AA032809"  "AA032811"  "AA032812"  "AA032822"  "AA032825"
##  [37] "AA010797"  "AA010752"  "AA010754"  "AA010756"  "AA010798"  "AA010799"
##  [43] "AA010800"  "AA010802"  "AA010803"  "AA010804"  "AA010809"  "AA010749"
##  [49] "AA010758"  "AA010763"  "AA010765"  "AA010771"  "AA010772"  "AA010781"
##  [55] "AA032762"  "AA032763"  "AA032756"  "AA032757"  "AA032758"  "AA032761"
##  [61] "AA032765"  "AA010931"  "AA010937"  "AA010940"  "AA032764"  "AA032768"
##  [67] "AA010936"  "AA010909"  "AA010916"  "AA010917"  "AA010920"  "AA010921"
##  [73] "AA020651"  "AA020652"  "AA020667"  "AA020669"  "AA020655"  "AA020656"
##  [79] "AA020644"  "AA020645"  "AA020646"  "AA020649"  "AA013203"  "AA013217"
##  [85] "AA013220"  "AA013202"  "AA013225"  "AA018496"  "AA018497"  "AA018513"
##  [91] "AA013231"  "AA013261"  "AA013265"  "AA013270"  "AA018492"  "AA018493"
##  [97] "AA018494"  "AA018495"  "AA018514"  "AA018515"  "AA018516"  "UC_00125"
## [103] "UC_00126a" "UC_00146"  "UC_00149"  "AA018640"  "AA018658"  "AA011729"
## [109] "UC_00132"  "UC_00137"  "UC_00143"  "UC_00157"  "UC_00161"  "AA018637"
## [115] "AA018638"  "AA018639"  "AA011731"  "AA033576"  "AA033577"  "AA011732"
## [121] "AA011737"  "AA011741"  "AA011744"  "AA011745"  "AA011746"  "AA011749"
## [127] "AA033575"  "AA033578"  "AA012411a" "AA033579"  "AA033582"  "AA033593"
## [133] "AA033602"  "AA033609"  "AA033617"  "AA010915a" "AA011723a" "AA019158"
## [139] "AA020379"  "UC_01044"  "AA018380"  "AA018371"  "AA004553"  "AA000328"
## [145] "AA000311"  "AA019159"  "AA020378"  "UC_01060"  "AA018379"  "AA018365"
## [151] "AA004554"  "AA000303"  "AA000320"  "AA019160"  "AA020377"  "UC_01053"
## [157] "AA018375"  "AA004555"  "AA000304"  "AA019165"  "AA019161"  "AA020376"
## [163] "UC_01062"  "AA018374"  "AA04523"   "AA000305"  "AA019164"  "AA020375"
## [169] "AA018373"  "AA032875"  "AA000309"  "AA019163"  "AA020374"  "AA018368"
## [175] "AA032878"  "AA000302"  "AA019162"  "AA020365"  "UC_00150"  "AA018369"
## [181] "AA004551"  "AA032880"  "AA000307"  "AA019156"  "AA020371"  "UC_01051"
## [187] "AA018370"  "AA004552"  "AA032882"  "AA000310"  "AA019157"  "AA019075"
## [193] "AA004864"  "AA019071"  "AA004868"  "AA019083"  "AA019072"  "AA004858"
## [199] "AA004869"  "AA019082"  "AA019073"  "AA004859"  "AA004866"  "AA019077"
## [205] "AA004860"  "AA019080"  "AA004861"  "AA019079"  "AA004862"  "AA019078"
```

```
## [211] "AA004863"  "UC_00267"  "UC_00205"  "UC_00206"  "UC_00208"  "UC_00243"
## [217] "UC_00209"  "UC_00254"  "UC_00210"  "UC_00259"  "UC_00126c" "AA063718"
## [223] "AA063720"  "AA063722"  "AA063726"  "AA063732"  "AA063708"  "AA063710"
## [229] "AA063712"  "AA063714"  "AA063716"  "AA020735"  "AA032442"  "AA032441"
## [235] "AA020749"  "AA020746"  "AA020744"  "AA020743"  "AA020739"  "AA020738"
## [241] "AA001451"  "AA01452"   "AA001454"  "AA001455"  "AA001446"  "AA001456"
## [247] "AA001447"  "AA001448"  "AA001449"  "AA001450"
```

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.

7.3 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"    "EmmacBurnBara"    "EmmacClarJack"
##  [5] "EmmacClarYate"    "EmmacCoopAvin"    "EmmacCoopCully"   "EmmacCoopEulb"
##  [9] "EmmacFitzAllig"   "EmmacJohnWari"    "EmmacMDBBowm"     "EmmacMDBCond"
## [13] "EmmacMDBCudg"     "EmmacMDBForb"     "EmmacMDBGwyd"     "EmmacMDBMaci"
## [17] "EmmacMDBMurrMung" "EmmacMDBSanf"     "EmmacMaclGeor"    "EmmacMaryBoru"
## [21] "EmmacMaryPetr"    "EmmacNormJack"    "EmmacNormLeic"    "EmmacNormSalt"
## [25] "EmmacRichCasi"    "EmmacRoss"        "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))`

7.3.1 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" "AA013214" "AA011723" "AA012411"
##  [7] "AA019237" "AA019238" "AA019239" "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)`.

7.4 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.

7.5 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.

7.6 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: 240.4 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  loc.metrics.flags  verbose  history
```

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  21
```

you see that the subsetting of the meta data for individuals worked fine (we have seven indivduals (=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
##   EmmacFitzAllig      10    0
##   EmmacJohnWari       10    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
##   EmmacMaclGeor       11    0
##   EmmacMaryBoru        6    0
##   EmmacMaryPetr        4    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] 255  11 103 173  62  14 110 144  58  80 127 107 248  73 104 236 116 170  92
## [20] 254 132 185  25  85 141 133 243   6  56  70
```

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: 197.7 Kb
##  53 (11.78 %) 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  loc.metrics.flags  verbose  history
```

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

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

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

```
## [1] 30 21
```

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.

8 Genetic Distance

SNP data are multivariable data, in the sense that each individual (entity) has an allele profile (state) for each of several loci (attributes). It is a simple data matrix because SNPs are bi-allelic, that is, each locus can have one of three allelic states - aa, ab or bb, coded in a genlight object as 0, 1 or 2 respectively. One allele (the most common allele) is assigned to the reference allele, and the other to the alternate (or in DArT documentation, the SNP allele).

An obvious first choice in exploring the data is to construct a distance matrix between individuals based on the genetic profiles, or to construct a distance matrix between populations based on their allele frequency profiles.

There are a very many measures of genetic distance, but they collapse in number when applied to bi-allelic SNP data. For example, Rogers D and Euclidean D differ only by a constant multiplier when the data are biallelic. Distance matrices can be generated by a number of R packages, the most popular of which are `dist()` from package `stats` and `vegdist` from package `vegan`. The function `gl.dist.pop` is a wrapper for those two functions, applying them to allele frequencies calculated for each locus at each population defined in the genlight object.

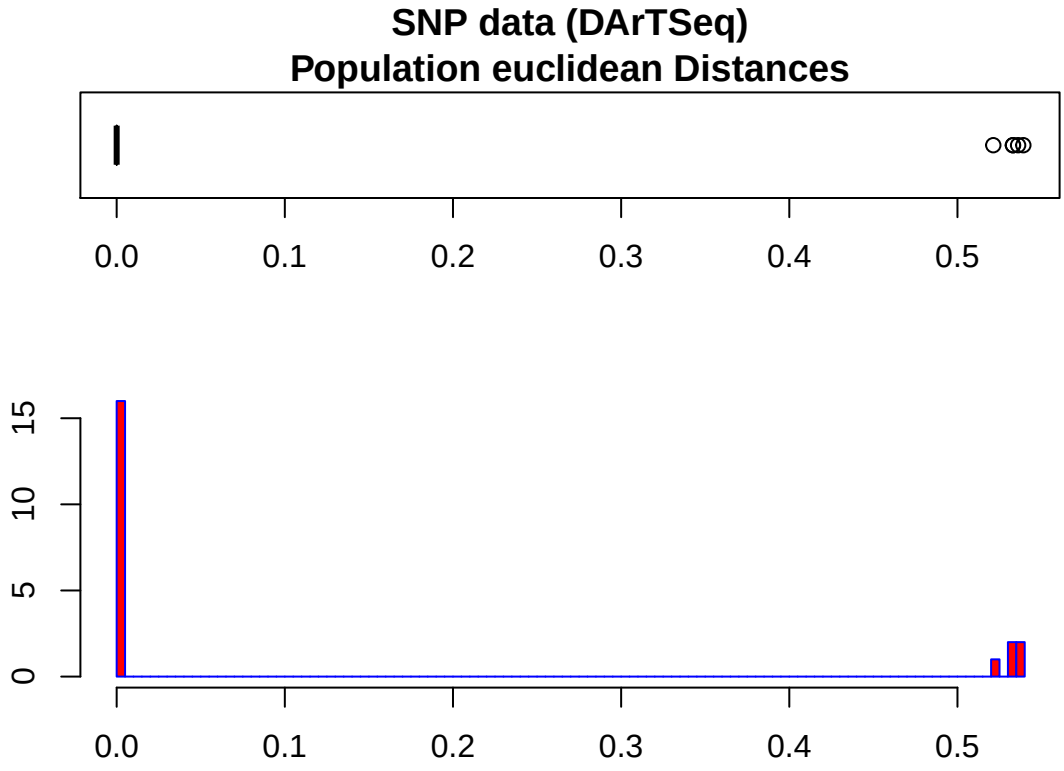
To calculate Euclidean distances (only for the first seven individuals and the first 100 loci, due to runtime reasons):

```
gg <- gl[1:7,1:100]
#need to subset loc.metrics as well!!!
gg@other$loc.metrics <- gg@other$loc.metrics[1:100,]

gl.dist.pop(gg, method="euclidean")
```

```
## Starting gl.dist.pop
##   Processing a SNP dataset
##   Calculating distances:  euclidean
##   Refer to dist {stats} documentation for algorithm

##   Standard boxplot, no adjustment for skewness
```



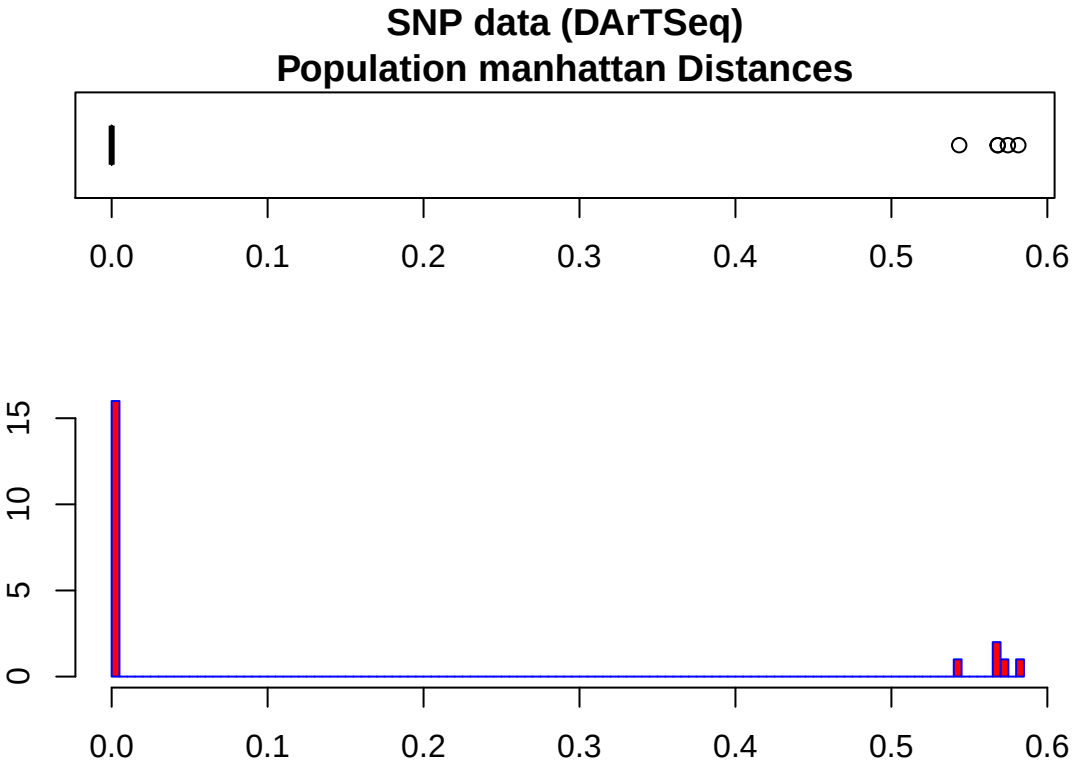
```
##
##   Reporting inter-population distances
##   Distance measure: euclidean
##     No. of populations = 7
##     Average no. of individuals per population = 1
##     No. of loci = 100
##     Miniumum Distance:  0
##     Maximum Distance:  0.54
##     Average Distance:  0.127
##
## Completed: gl.dist.pop
```

```
##               EmmacBurdMist EmmacBurnBara EmmacCoopEulb EmmacMDBForb
## EmmacBurnBara     0.0000000
## EmmacCoopEulb     0.0000000     0.0000000
## EmmacMDBForb      0.0000000     0.0000000     0.0000000
## EmmacMDBMaci      0.0000000     0.0000000     0.0000000    0.0000000
## EmmacMDBSanf      0.0000000     0.0000000     0.0000000    0.0000000
## EmmacMaclGeor     0.5330018     0.5360563     0.5330018    0.5391639
##               EmmacMDBMaci EmmacMDBSanf
## EmmacBurnBara
## EmmacCoopEulb
## EmmacMDBForb
## EmmacMDBMaci
## EmmacMDBSanf     0.0000000
## EmmacMaclGeor    0.0000000    0.5212860
```

```
gl.dist.pop(gg, method="manhattan")
```

```
## Starting gl.dist.pop
##   Processing a SNP dataset
##   Calculating distances:  manhattan
##   Refer to dist {stats} documentation for algorithm

##   Standard boxplot, no adjustment for skewness
```




```
##
##   Reporting inter-population distances
##   Distance measure: manhattan
##     No. of populations = 7
##     Average no. of individuals per population = 1
##     No. of loci = 100
##     Miniumum Distance:  0
##     Maximum Distance:  0.58
##     Average Distance:  0.135
##
## Completed: gl.dist.pop
```

```
##               EmmacBurdMist EmmacBurnBara EmmacCoopEulb EmmacMDBForb
## EmmacBurnBara     0.0000000
## EmmacCoopEulb     0.0000000     0.0000000
## EmmacMDBForb      0.0000000     0.0000000     0.0000000
## EmmacMDBMaci      0.0000000     0.0000000     0.0000000    0.0000000
## EmmacMDBSanf      0.0000000     0.0000000     0.0000000    0.0000000
## EmmacMaclGeor     0.5681818     0.5747126     0.5681818    0.5813953
##               EmmacMDBMaci EmmacMDBSanf
## EmmacBurnBara
## EmmacCoopEulb
## EmmacMDBForb
## EmmacMDBMaci
## EmmacMDBSanf     0.0000000
## EmmacMaclGeor    0.0000000    0.5434783
```

Distances available for computation include:

method = “manhattan”, “euclidean”, “canberra”, “bray”, “kulczynski”, “jaccard”, “gower”, “altGower”, “morisita”, “horn”, “mountford”, “raup” , “binomial”, “chao”, “cao”, “mahalanobis”, “maximum”, “binary” or “minkowski”.

Refer to the documentation for functions ?dist and ?vegdist for computational formulae.

YOu have been wondering why there are only 3 pairs of distances calculated. This is because gl.dist.pop calculates distances using the population information in the genlight object. If someone wants to calculate distances between individuals we need to redefine the population information in the genlight object. A convinient way is to use the individuals identifier which is stored in the indNames slot.

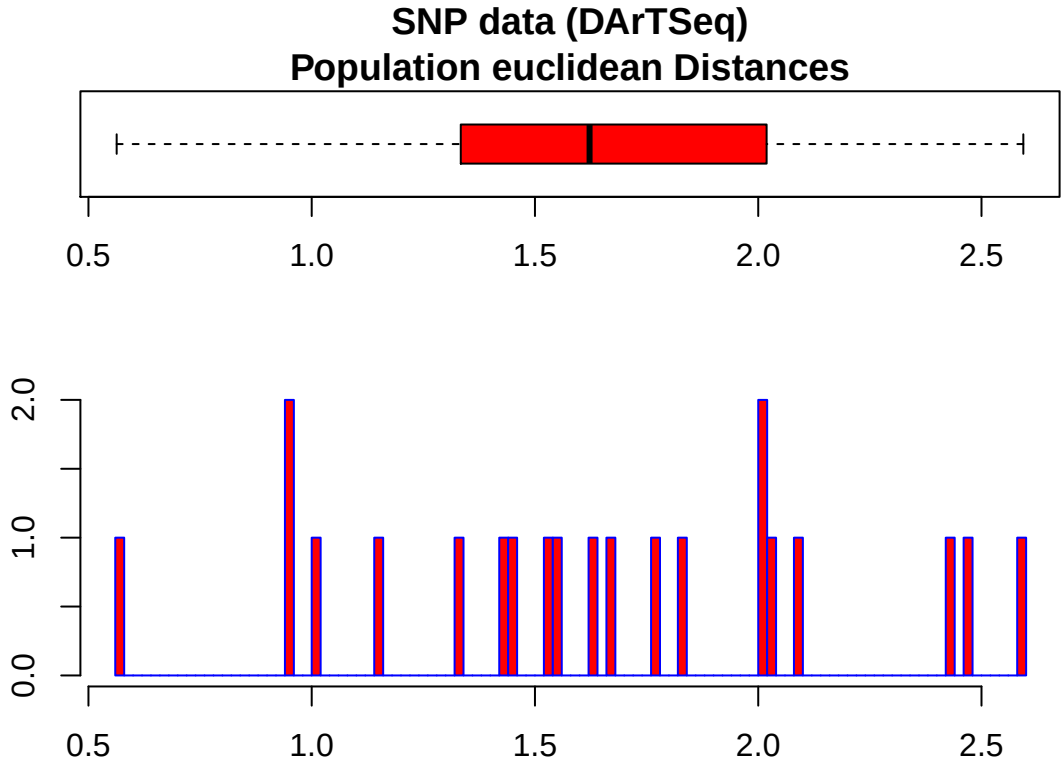
Here is an example how to calculate individuals based distances for the first 7 individuals based on all loci:

```
glind7 <- gl[1:7,]  #copy and store the original dataset in glind
pop(glind7) <- indNames(glind7)# redefine the population information

gl.dist.pop(glind7[1:7,], method="euclidean")
```

```
## Starting gl.dist.pop
##   Processing a SNP dataset
##   Calculating distances:  euclidean
##   Refer to dist {stats} documentation for algorithm

##   Standard boxplot, no adjustment for skewness
```



```
##
##   Reporting inter-population distances
##   Distance measure: euclidean
##     No. of populations = 7
##     Average no. of individuals per population = 1
##     No. of loci = 255
##     Miniumum Distance:  0.56
##     Maximum Distance:  2.59
##     Average Distance:  1.639
##
## Completed: gl.dist.pop
```

```
##           AA010915  AA011723  AA012411  AA013214  AA019237  AA032760
## AA011723 1.8317377
## AA012411 2.5937207 2.0255179
## AA013214 1.3338177 2.0187477 2.4640269
## AA019237 1.7634501 0.9498014 2.0009056 1.6223072
## AA032760 1.0140147 1.5325417 2.4335429 0.5631733 1.4226066
## UC_00126 1.4577380 1.5473740 2.0814717 1.6758192 0.9475692 1.1464872
```

```
data.frame(ind=1:7, indNames=indNames(gl)[1:7], pop=pop(gl)[1:7])
```

```
##   ind indNames           pop
## 1   1 AA010915  EmmacMDBForb
## 2   2 UC_00126 EmmacMaclGeor
## 3   3 AA032760  EmmacMDBMaci
## 4   4 AA013214  EmmacMDBSanf
## 5   5 AA011723 EmmacBurnBara
## 6   6 AA012411 EmmacCoopEulb
## 7   7 AA019237 EmmacBurdMist
```

In the section below you will see that individuals from the area around the Cooper (individual 6, AA012411, EmmacCoopEulb) are genetically quite different from the other populations. This is also reflected in the largest euclidean pairwise distance for this individuals with every other individual (>2).

8.1 F Statistics

Several packages have already implemented the calculations of F statistics. Therefore we just provide different ways how to convert our data sets and use already existing packages:

For Fst and Neis Gst we recommend the use of functions of the StAMPP package as they allow for the use of parallel computing, which is much faster. It also works on genlight objects directly and therefore no conversion is necessary. For a more detailed explanation on options to achieve also confidence intervals via bootstrap refer to the StAMPP help pages) stamppFst calculates pairwise Fst values between populations. (due to performance reasons we use only the first 30 individuals, which result in 8 populations :

```
library(StAMPP) #you may need to install the package
pwfst <-stamppFst(gl[1:20,], nboots=1, percent=95, nclusters=1)
round(pwfst,3)
```

For Neis Gst use

```
pwGst <-stamppNeisD(gl[1:20,]) #no parallel version :-()
round(pwGst,3)
```

For Jost's D and G'st we suggest to use functions from the mmod package. The disadvantage here is that it is much slower as we need to convert our data set from a genlight to a genind object and then use a none-parallel function from mmod (depending on your computer the example will take some time).

```
library(mmod) #you may need to install the package first
#for performance reason use only a subset (and recode the populations)
recpops<- factor(rep(LETTERS[1:5],50))
glsub <- testset.gl
pop(glsub)<-recpops

gi <- gl2gi(glsub, v=0) #v=0 suppresses output
round(pairwise_D(gi),4)
```

```
##        A      B      C      D
## B -1e-04
## C  2e-04 -4e-04
## D  5e-04 -1e-04  2e-04
## E  0e+00 -4e-04 -1e-04  0e+00
```

```
round(pairwise_Gst_Hedrick(gi),4)
```

```
##         A       B       C       D
## B -0.0037
## C  0.0064 -0.0113
## D  0.0133 -0.0015  0.0064
## E -0.0011 -0.0118 -0.0028 -0.0014
```

```
round(pairwise_Gst_Nei(gi),4)
```

```
##         A       B       C       D
## B -0.0018
## C  0.0031 -0.0054
## D  0.0065 -0.0007  0.0031
## E -0.0005 -0.0057 -0.0014 -0.0007
```

9 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.

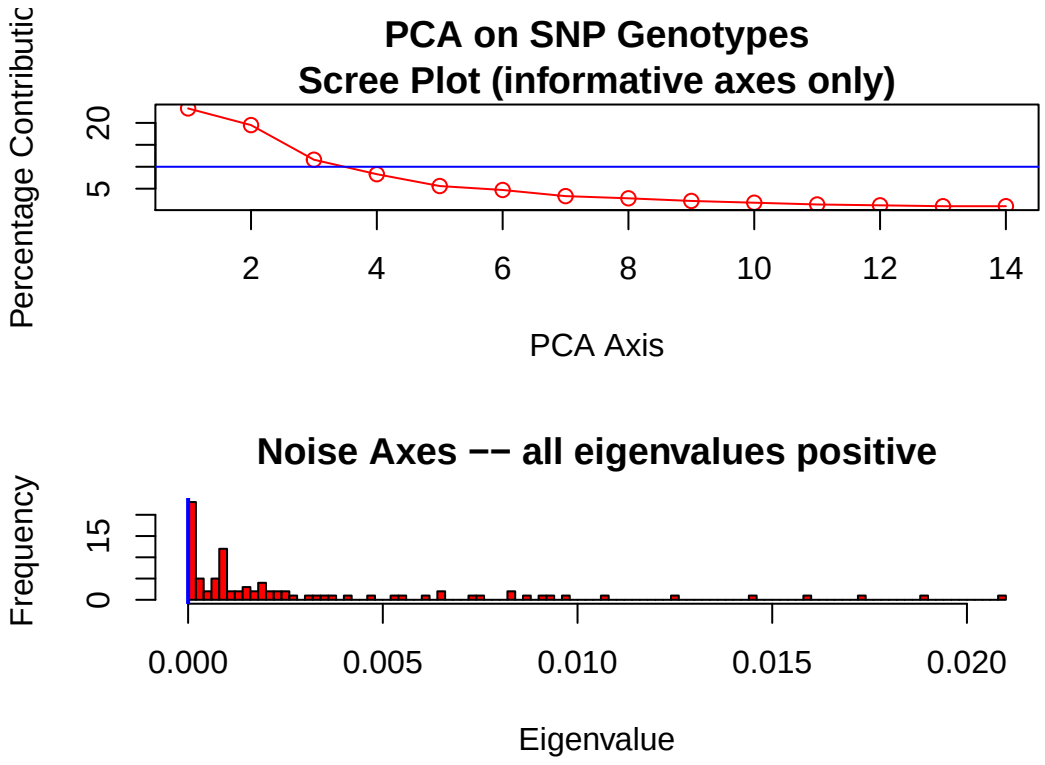
Orthogonal linear combinations of the original axes are calculated and ordinated such that the first PCoA axis explains the most variation, PCoA-2 is orthogonal to PCoA-1 and explains the most residual variation, and so on. A scree plot of eigenvalues provides an indication of the number of informative 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.

9.1 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)
```

```
## Starting gl.pcoa
##   Processing a SNP dataset
##   Performing a PCA, individuals as entities, loci as attributes, SNP genotype as state
```




```
## Completed: gl.pcoa
```

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)
```

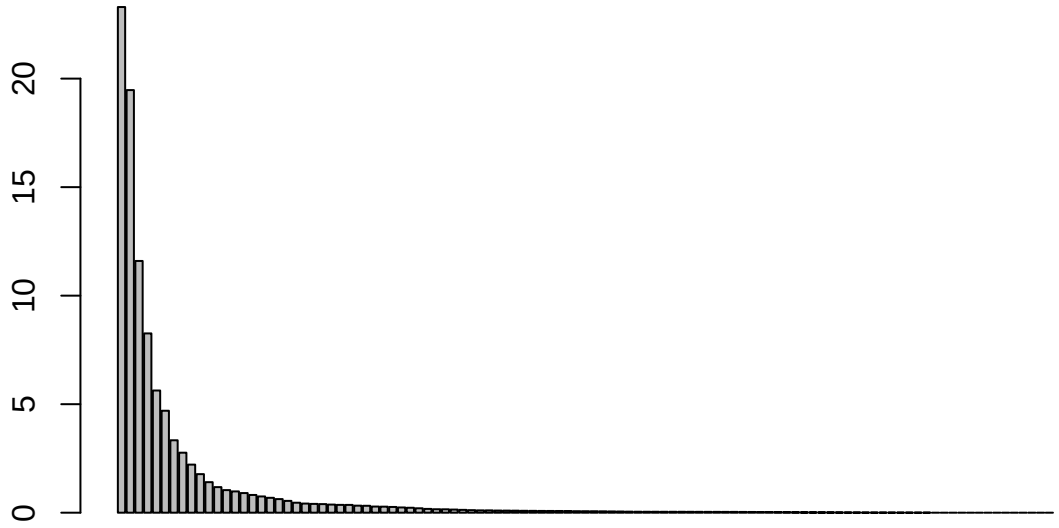
```
## [1] "scores"   "eig"      "loadings" "call"
```

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, )
```



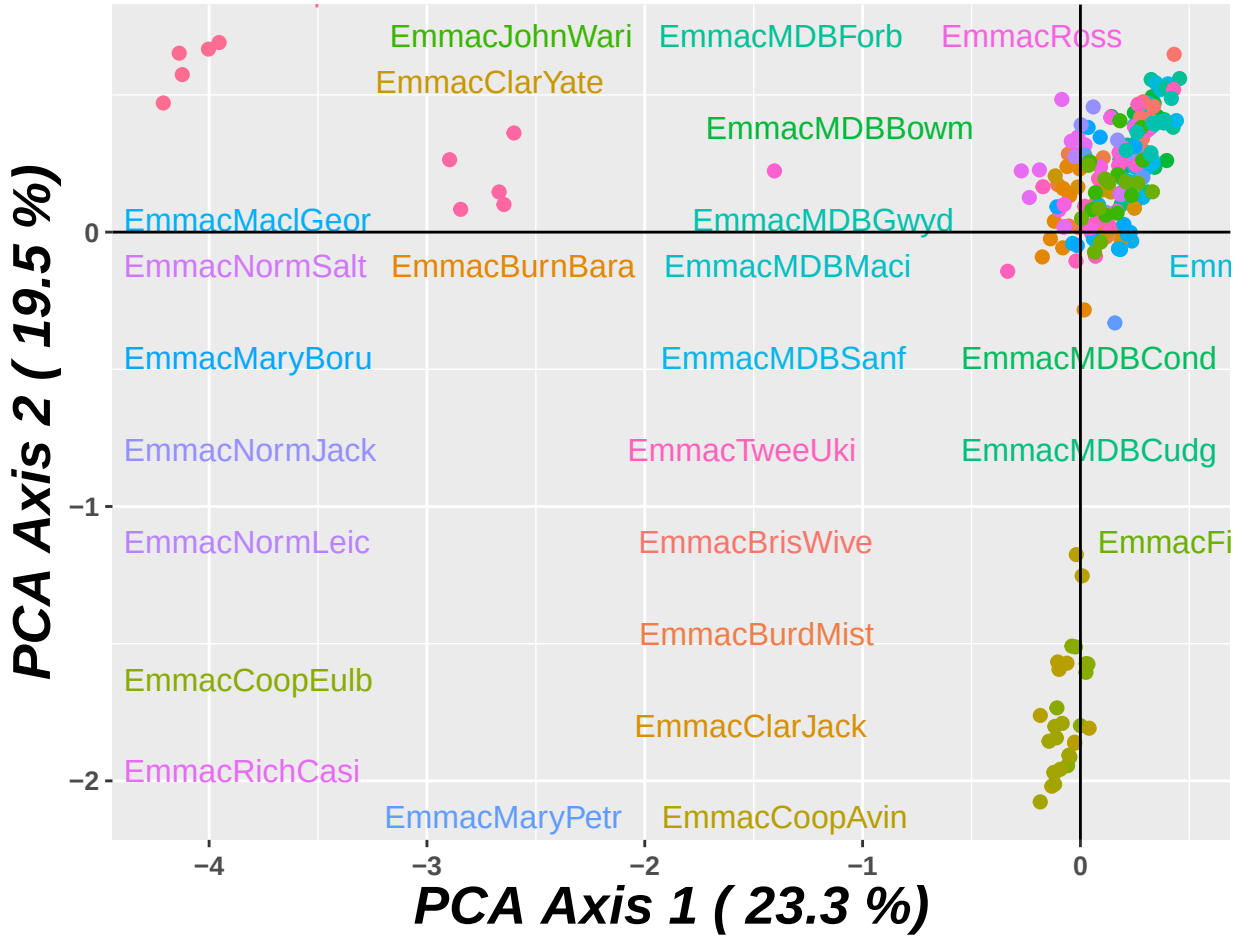
9.2 Plotting the results of PCoA

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}.

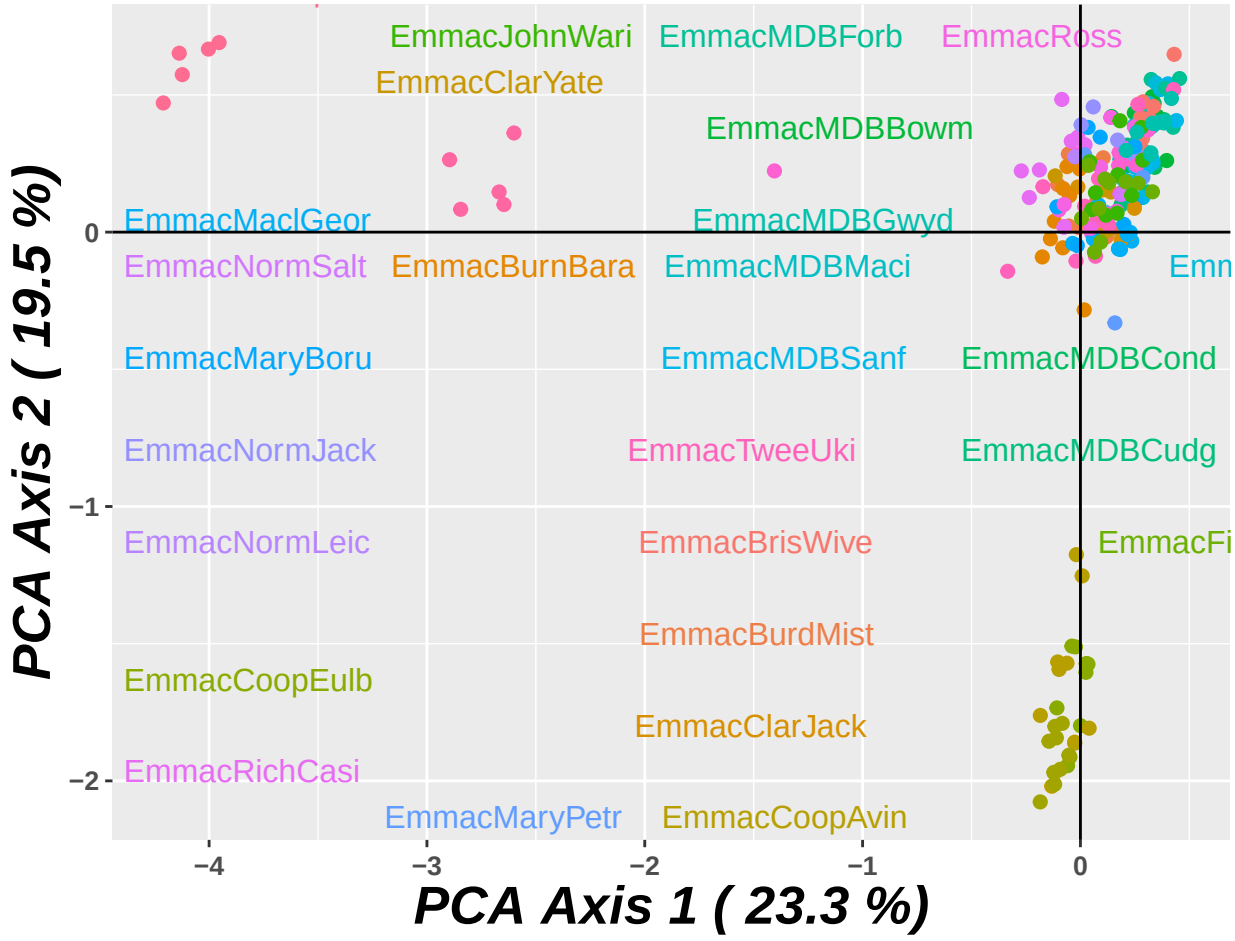
The plotting script is not intended to produce publication quality plots, but should form a basis for importing the plots to illustrator for subsequent amendment. The command

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

```
## Starting gl.pcoa.plot
##   Plotting populations
##   Preparing plot .... please wait
```




```
##   While waiting, returning ggplot compliant object
## Completed: gl.pcoa.plot
```



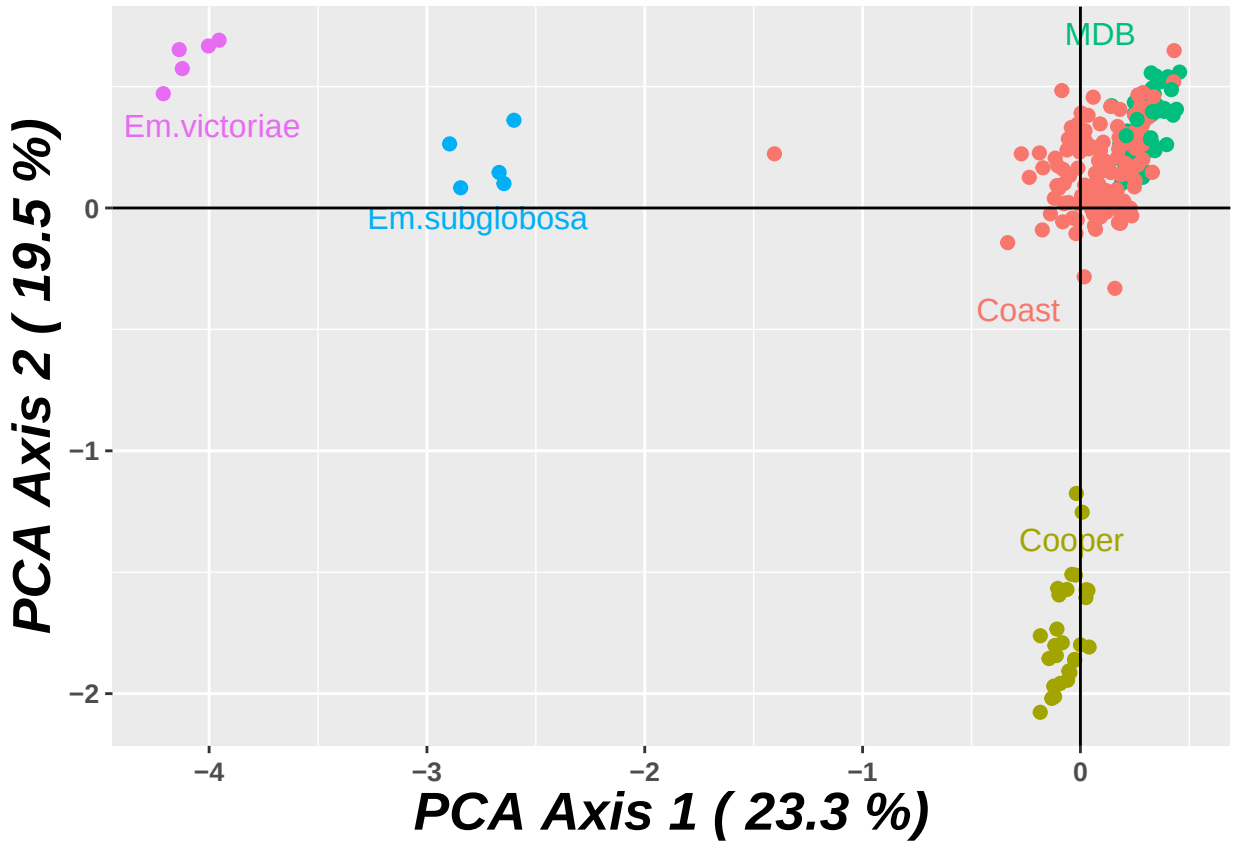
You can see that this plot is very busy, and that the many labels are displaced quite some distance from their associated points. This is because there is a tradeoff between avoiding overlap of the labels and proximity of the labels - you can use colour to identify which labels go with which points.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

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

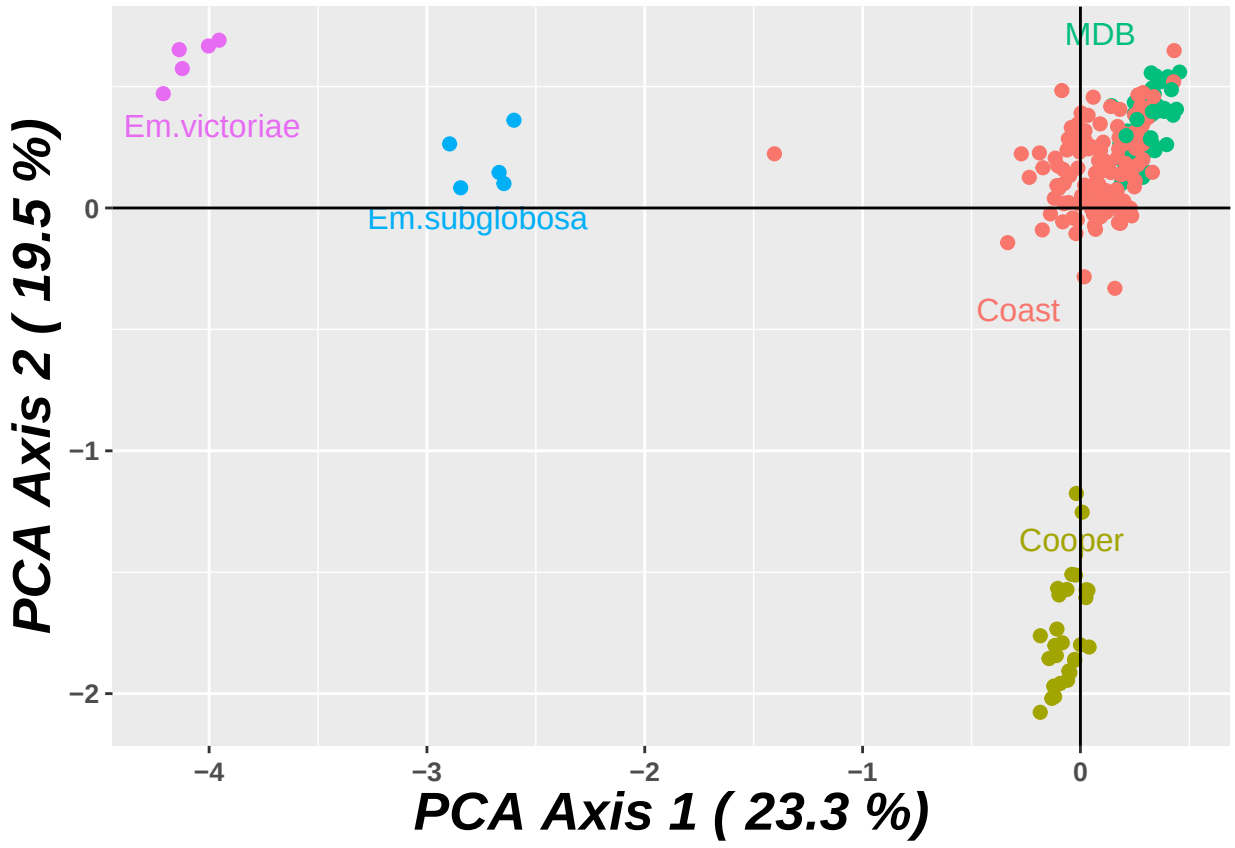
or using R

```
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)
```

```
## Starting gl.pcoa.plot
##   Plotting populations
##   Preparing plot .... please wait
```




```
##   While waiting, returning ggplot compliant object
## Completed: gl.pcoa.plot
```



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}:

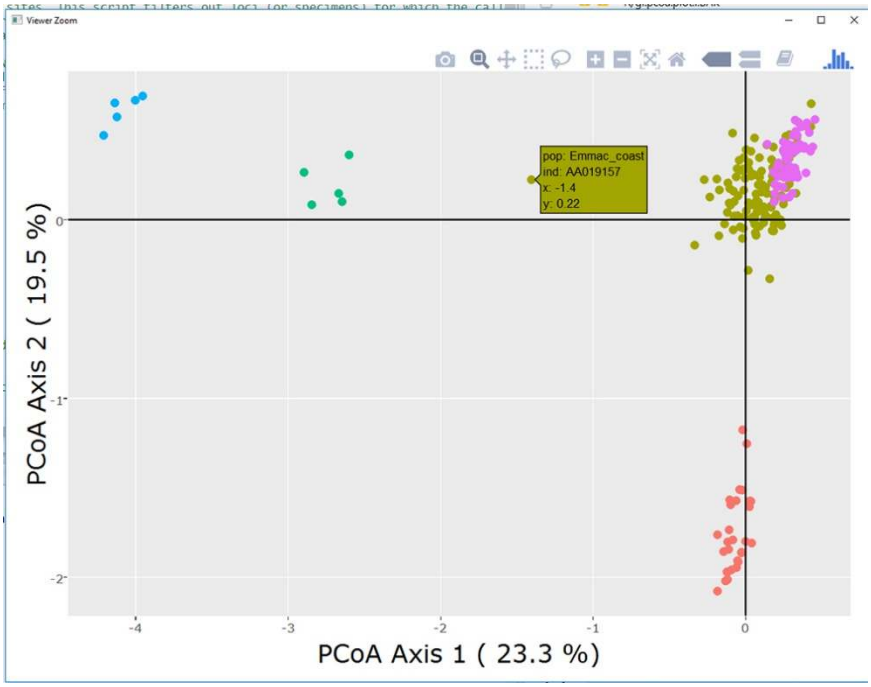
In case ggplotly is not installed, please type:

```
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.

9.3 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)
```

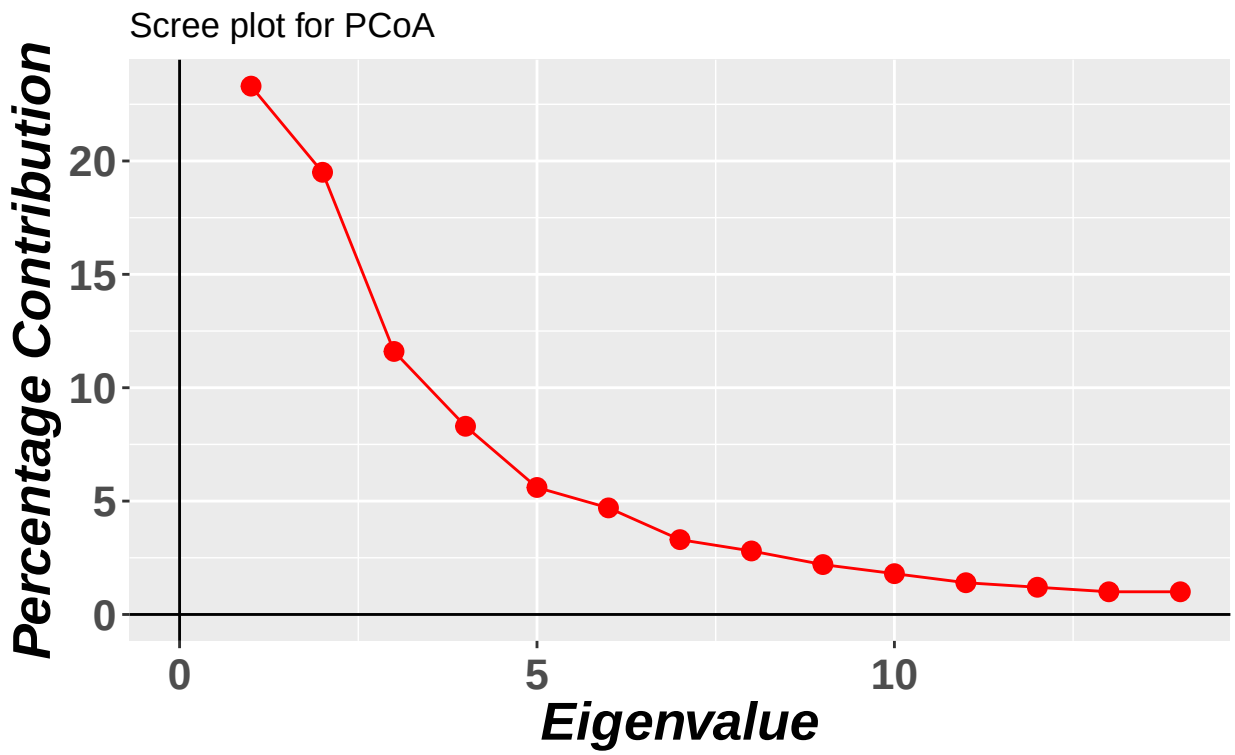
```
##   Note: Only eigenvalues for dimensions that explain more that the average of the original variables
##   No. of axes each explaining 10% or more of total variation: 3

## Warning: Use of `df$eigenvalue` is discouraged. Use `eigenvalue` instead.

## Warning: Use of `df$percent` is discouraged. Use `percent` instead.

## Warning: Use of `df$eigenvalue` is discouraged. Use `eigenvalue` instead.

## Warning: Use of `df$percent` is discouraged. Use `percent` instead.
```

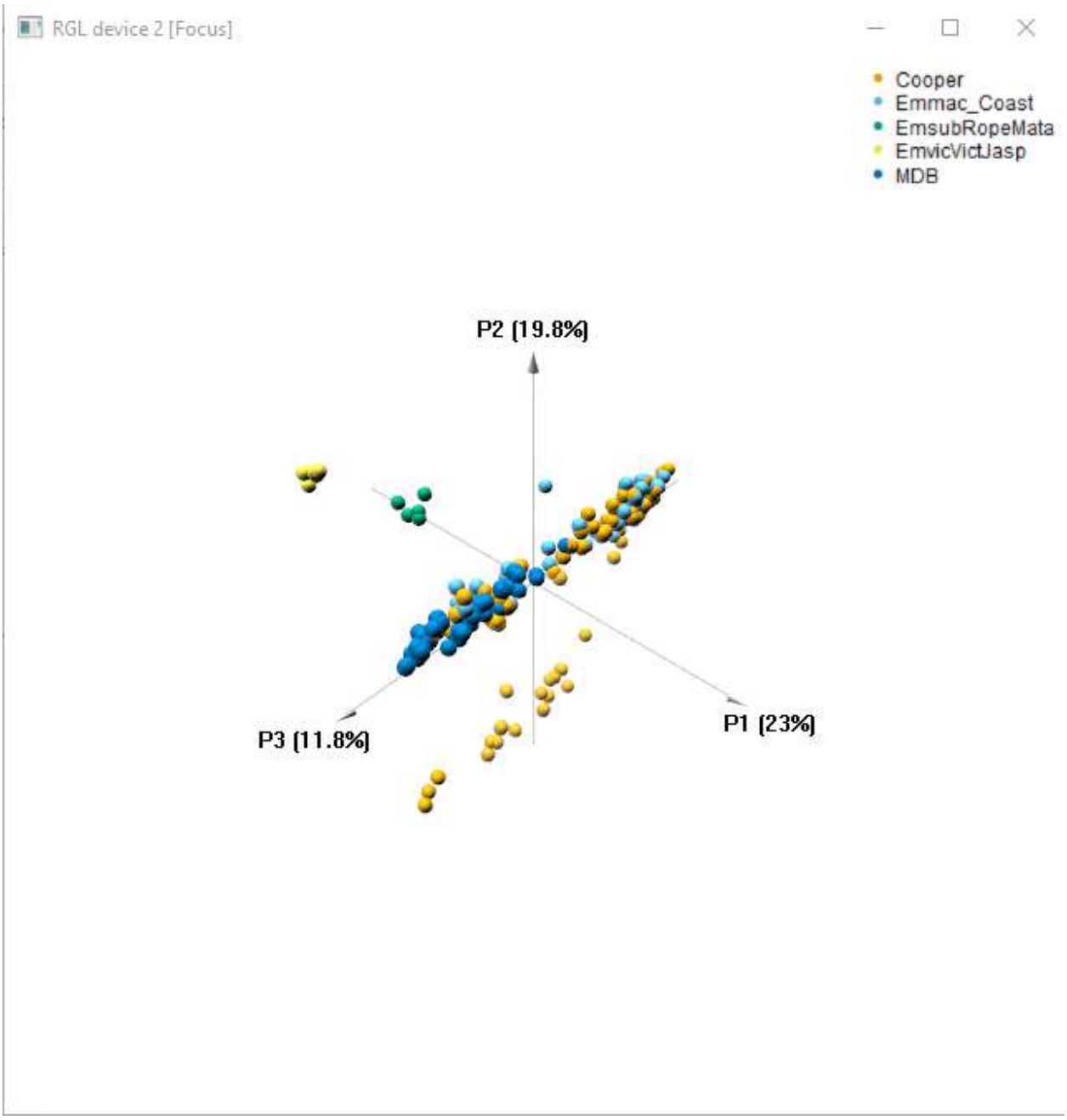


This plot, by default, will show the percentage variation in the data explained by each axis successively where the amount of variation is substantive. By substantive, I mean explaining more than the original variables did on average. As a rule of thumb, one should examine all dimensions that explain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

9.4 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)
```



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.

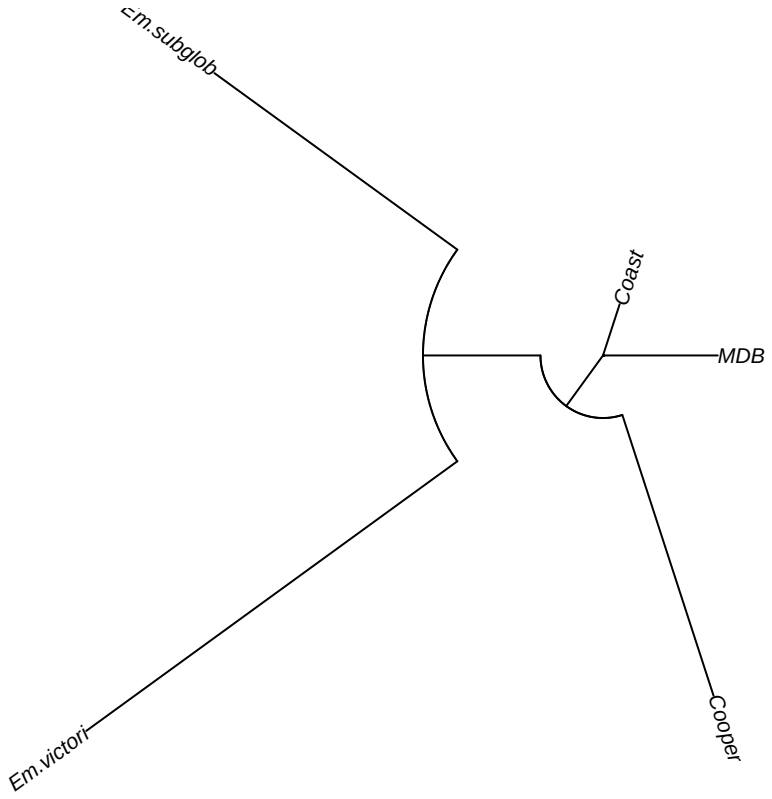
This function is essentially a wrapper for the corresponding function in {pca3d}, adding percentage variation explained to each axis and fixing some parameters.

9.5 Neighbour-joining trees

Another way of visualizing genetic similarity among entities is to construct a phenetic tree using a neighbor-joining approach or UPGMA. This script is essentially a wrapper for nj {ape} applied to Euclidean distances between OTUs.

```
gl.tree.nj(glnew, type="fan")
```

```
## Starting gl.tree.nj
##   Processing a SNP dataset
##   Converting to a matrix of frequencies, locus by populations
##   Computing Euclidean distances
```




```
## Completed: gl.tree.nj
```

```
##
## Phylogenetic tree with 5 tips and 3 internal nodes.
##
## Tip labels:
## [1] "Coast     " "Cooper    " "MDB       " "Em.subglob" "Em.victori"
##
## Unrooted; includes branch lengths.
```

10 Fixed Difference Analysis

The PCoA approach outlined above considers allele frequency differences between individuals or populations as a basis for constructing a distance matrix which is then used by the PCoA algorithms to execute the ordination.

There is some advantage in considering only fixed differences between populations to examine structure, that is, to use allelic differences where the alleles have come to fixation to alternative states in populations taken pairwise.

A fixed difference between two populations at a specific locus occurs when the population share no alleles at that locus. Allele frequencies may ebb and wane, but once a locus becomes fixed for an allele or suite of alleles, there is no returning save a convergent mutation (rare) or gene flow. The acquistion of a fixed difference between two populations is a significant biological event. Because it takes only one individual to disperse between two populations per generation to retard divergence, the accumulation of fixed differences between two populations is considered a robust indication of lack of gene flow.

In a nutshell, fixed differences are summed over populations taken pairwise, and when two populations have no fixed differences (or insubstantial fixed differences), the populations are amalgamated and the process repeated until there is no further reduction (Georges and Adams, 1996). The final set of taxa are diagnosable by the presence or absence of a set of alleles at multiple loci.

As samples taken from populations for sequencing are finite, there are sampling issued to consider. There is an asymmetry in these considerations. Populations can be amalgamated on the basis of shared alleles at all loci with confidence, because it only takes one individual from one population to share alleles at a locus with the other population to conlude that there is no fixed difference at that locus. So the process of amalgamating populations based on shared alleles at all loci is robust.

This is not the case when interpeting fixed differences between two populations as a basis for their diagnosability. Because of the finite sample sizes, an observed fixed difference may have arisen because the populations from which the samples were drawn do not share alleles at a particular locus, or because there is a non-trivial probability that the observed fixed difference arose through sampling error (a false positive).

Distinguishing between real fixed differences and false positives is difficult, because we do not have knowledge of the true allele fequencies for each of the two populations being compared. Our approach to dealing with this is to turn to simulation. Briefly, we take the observed allele frequencies for the two populations under consideration, take random samples of the allele profiles for each of the observed n1 and n2 individuals, count the number of fixed differences, and repeat this, say 1000 times. This provides an estimate of the expected number of fixed differences for the two populations.

Because the two populations may have true fixed differences, this estimate of expected fixed differences conflates true fixed differences with false positives. We accommodate this by asking you to make a judgement. How extreme does the divergence between the two populations have to be (say 99:1%, 1:99%) for you to regard a sample fixed difference (100:0, 0:100) as a positive rather than a false positive? A value of delta = 0.02 (98:2, 2:98) might be a reasonable compromise. Then, any fixed differences that arise in the samples from true allele frequencies of less extreme than 98:2, 2:98 are considered false positives. This decision accommodates the possiblity of true fixed differences in the populations from which we draw the simulated samples.

The script gl.collapse.recursive() optionally calculates the expected number of false positives (and its standard error), and compares it with the observed number of fixed differences to yield a p value for the statistical significance of the the result.

The script gl.collapse.pval() takes the final set amalgamated populations arising from gl.collapse.recursive(), and further amalgamates populations based on lack of statistical significance of the observed fixed differences.

The resulting set of aggregated populations can be regarded as diagnostic OTUs for which you can be confident there is strong support for lack of recent or contemporary gene flow between them, such that phylogenetic approaches to extracting a bifrucating pattern of ancestry and descent are appropriate.

The script

```
fd <- gl.collapse.recursive(gl, t=0)
```

will ultimately yield a grouping of aggregate populations that are diagnosable from each other by one or more fixed allelic differences. Here is the output, abridged: Let's refresh the data again, to bring back in the outgroups, then run the analysis.

The output comprises a list of matricies.

fd$gl -- input genlight object; fd$fd – raw fixed differences; fd$pcfd -- percent fixed differences fd$nobs – mean no. of individuals used in each comparison fd$nloc – -total number of loci used in each comparison fd$expobs – if test=TRUE, the expected count of false positives for each comparison [by simulation] fd$prob – if test=TRUE, the significance of the count of fixed differences [by simulation]

A full series of distance matrices and associated recode tables have been filed to disk for later interrogation and use

An option to statistically compare the number of observed fixed differences with those expected to arise by chance given the finite sample sizes is available using test=TRUE. Populations can then be further amalgamated based on lack of significance using the script gl.collapse.pval().

The output is a list of matricies as above, but for the aggregated populations.

The outcome is unequivocal. All of the coastal Emydura macquarii and those from the Murray-Darling basin amalgamate into one OTU on the basis of no fixed differences between them (Group 1.1). The Cooper Creek animals form a separate diagnosable unit (OTU) (Group 1.2). The populations that did not amalgamate outgroup taxa - Emydura victoriae and Emydura subglobosa - remain well supported.

You can see the power of this approach for defining diagnosable taxa, taxa that are free of contemporary or recent geneflow and so on independent evolutionary trajectories.

##Distance Phylogeny on resultant OTUs

The script for distance phylogeny is gl2phylip() which calculates Euclidean distances using dist {stats} then outputs the data in a form suitable for input to the Phylip package written by Joseph Felsenstein (http://evolution.genetics.washington.edu/phylip.html) (Felsenstein, 1989). The input file can include replicated distance matrices for the purpose of bootstrapping.

```
gl <- fd.sig$gl
phy <- gl2phylip(gl, outfile="turtle.phy", bstrap=1000)
```

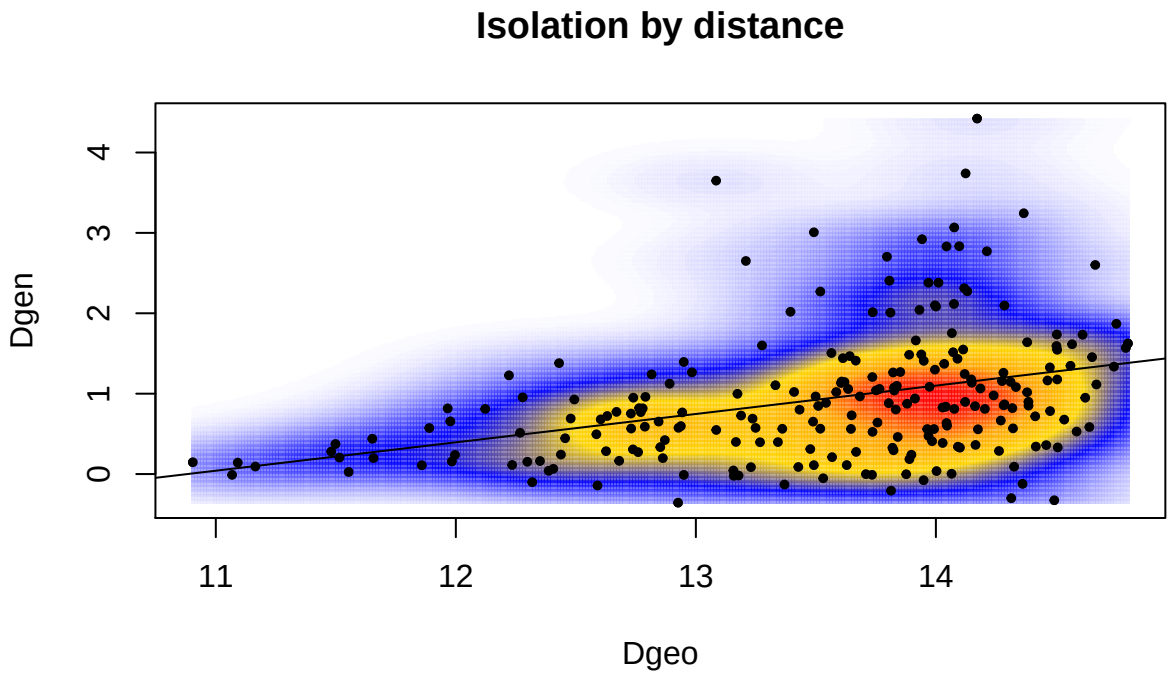
11 Isolation by distance plot

This functions performs an isolation by distance analysis based on a mantel test and also produces an isolation by distance plot. If a genlight object with coordinates (in the @other$latlong slot) is provided, then a Euclidean and genetic distance matrix are calculated (currently only pairwise Fst between population is implemented by default). Coordinates are expected as lat long and converted to Google Earth Mercator projections to avoid distortions. If coordinates are already projected in a suitable coordinate system, you can set projected=TRUE and no projection is applied. An isolation by distance analysis will be carried out and a plot will be returned, based on log(Euclidean distance) against between population pairwise Fst/1-Fst (see Rousseau's distance measure. Genetics April 1, 1997 vol. 145 no. 4 1219-1228) You can also provide your own genetic and Euclidean distance matrix, then simply the mantel and isolation by distance plot will be returned. The function is based on code provided by the adegenet tutorial (http://adegenet.r-forge.r-project.org/files/tutorial-basics.pdf), using the functions mantel (package vegan), stamppFst (package StAMPP) and Mercator (package dismo).

An example run (using only a subset of the test data):

```
gl <- gl.ibd(gl=testset.gl[1:180,])
```

```
## Standard analysis performed on the genlight object. Mantel test and plot will be Fst/1-Fst versus log
## Coordinates transformed to Mercator (google) projection to calculate distances in meters.
##
## Mantel statistic based on Pearson's product-moment correlation
##
## Call:
## mantel(xdis = Dgen, ydis = Dgeo, permutations = 999, na.rm = TRUE)
##
## Mantel statistic r: 0.3655
##       Significance: 0.001
##
## Upper quantiles of permutations (null model):
##   90%   95% 97.5%   99%
## 0.145 0.175 0.197 0.221
## Permutation: free
## Number of permutations: 999
```



12 Phylogenetic Analysis

The objective of phylogenetic analysis is to extract relationships between taxonomic entities, be they species, evolutionarily significant units (ESUs) or other diagnosable units (Felsenstein, 2004; Swofford and Berlocher, 1987). The goal is thus to extract the pattern of ancestry and descent among such taxonomic entities (call them operational taxonomic units or OTUs). The OTUs need to be diagnosable because one assumes that differences among them reflect divergence through time, unobscured by contemporary or recent tokogenic exchange. That is, they are considered to be on evolutionary trajectories that are independent by virtue of reproductive or long-standing geographic isolation.

The true evolutionary history of the OTUs is in the form of a bifurcating tree, that is, the true divergences among OTUs satisfy both the conditions of a metric and the four-point condition (Buneman, 1973). Metric is used here in the sense that, given the positions of two OTUs to represent the distance between them, the distances to a third OTU uniquely defines its position. The four- point condition is used here in the sense that for any four OTUs, there exists a simple tree accurately depicting the distances between them (that is, there exists a non-negative internal branch). A set of OTUs and pairwise distances among them that satisfy the metric and four-point conditions will define a unique bifurcating tree.

In practice, homoplasy obscures the true phylogeny by distorting measures of genetic distance between taxa so that they no longer meet the metric and four-point criteria. The challenge becomes to estimate the most parsimonious (least steps), most likely (maximum likelihood) or the best-bet (Bayesian estimate) tree in terms of consistency with available data. Because information contained in the genetic code can be over-written by new mutations, the true evolutionary history may not be recoverable even with the most comprehensive of contemporary data. Instead, the exercise is to generate the phylogeny most consistent with available data, and use this solution as the best available hypotheses for future testing, in whole or in part, as new data or new methods of analyzing those data come to hand.

There are a number of approaches and a large range of software packages for recovering phylogenies which are not covered here (Felsenstein, 1989; Swofford, 2002). The objective of {dartR} is to generate files in a format that can be read into the packages of choice. We export to fastA format.

12.1 Distance Methods

Under ideal conditions, the true phylogeny can be uniquely recovered from the true measures of divergence between OTUs (that is, tree distances define a unique tree). So one approach to recovering phylogenies is to extract the phylogeny that is most consistent with the estimated distances between OTUs. To avoid introducing artificial departure of the distances from the underlying tree, a metric distance needs to be chosen, and for SNP datasets we choose Euclidean Distance. This differs from Rogers D (Rogers, 1972) by a constant multiplier, and so the two measures are essentially the same.

The script for distance phylogeny is gl2phylip() which calculates Euclidean distances using dist {stats} then outputs the data in a form suitable for input to the Phylip package written by Joseph Felsenstein (http://evolution.genetics.washington.edu/phylip.html) (Felsenstein, 1989). The input file can include replicated distance matrices for the purpose of bootstrapping.

Assuming the data have been appropriately filtered before saving, the commands to do this are

```
gl <- testset.gl
phy <- gl2phylip(gl, outfile="turtle.phy", bstrap=1000)
```

The output file, turtle.phy or whatever you decide to call it, is available for input to Phylip and the variety of commands for handling distance matricies (e.g. fitch). Refer to the Phylip documentation.

12.2 Character-based Methods

As with allozyme data sets of the past, SNP datasets present particular challenges for phylogenetic analysis. The challenges arise because of difficulty in handling heterozygotes. Individuals homozygous for the reference state (0) or homozygous for the alternate state (2) are unambiguous in their character state, but heterozygotes (1) present both character states.

12.3 Converting Diploid to Haploid

One way of overcoming this is to randomly allocate the SNP state for heterozygous loci to one or the other homozygous states, that is, to add background noise to the data from which the phylogenetic signal can

still be extracted without introducing a systematic bias. We convert diplotypes to haplotypes amenable to phylogenetic analysis with `gl2fasta()` using method 1 to 4 (see ?gl2fasta for an explanation or the various methods availalbe)

The way to do this is to use the command:

```
gl2fasta(gl, method=2, outfile="nohets.fasta")
```

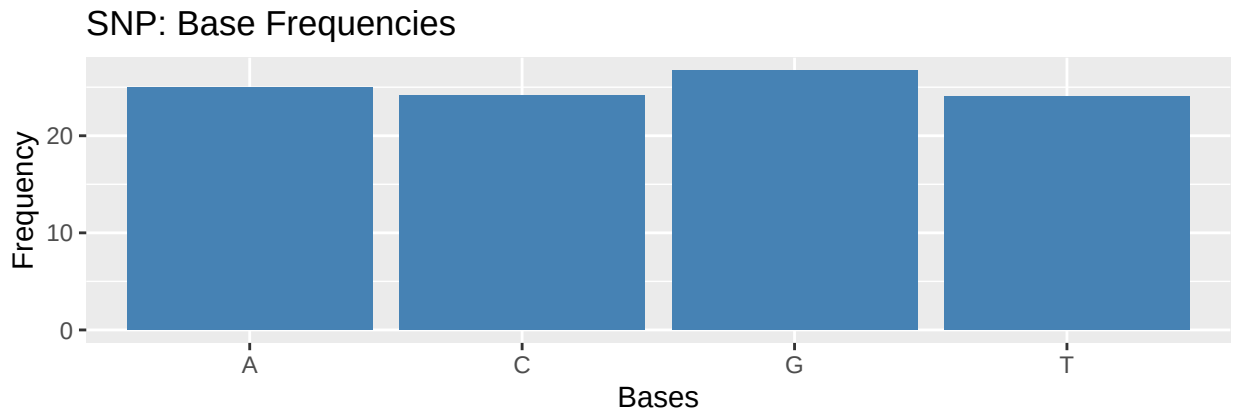
which concatenates the sequences of the DNA fragments, trimmed of adaptors, into a fastA file, having first randomly allocated the heterozygotic states. This set of "composite haplotypes" is then suitable for analysis using the diverse range of phylogenetic packages available.

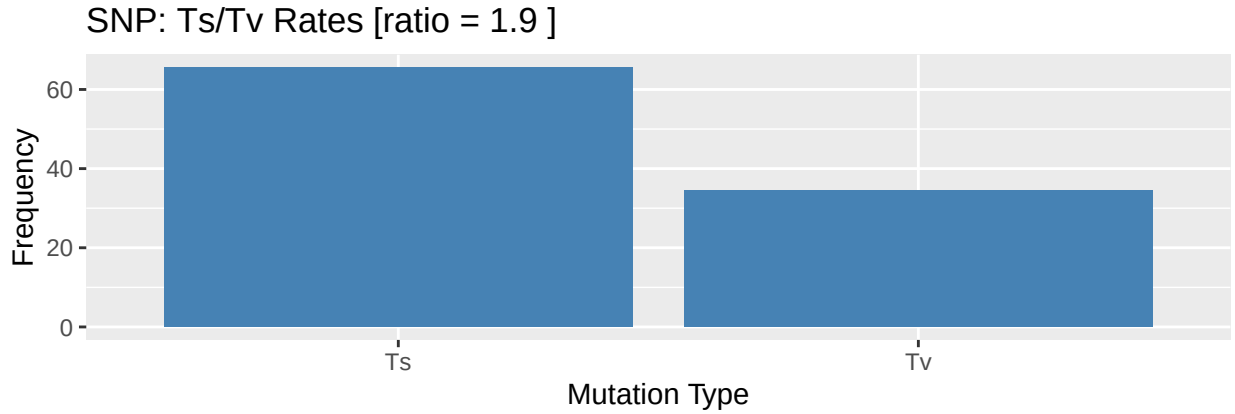
The reason for including the full trimmed sequences, most of the bases of which are monomorphic across individuals, is to allow for application of various mutational models in likelihood analysis. These require estimates of base frequencies and the frequency of transitions and transversions and the software to generate these estimates and factor them in to the phylogenetic analysis e.g. ModelTest (Posada and Crandall 1998) require the full sequence information.

The base frequencies and transition and transversion ratios can in any case be obtained using

```
gl.report.bases(testset.gl)
```

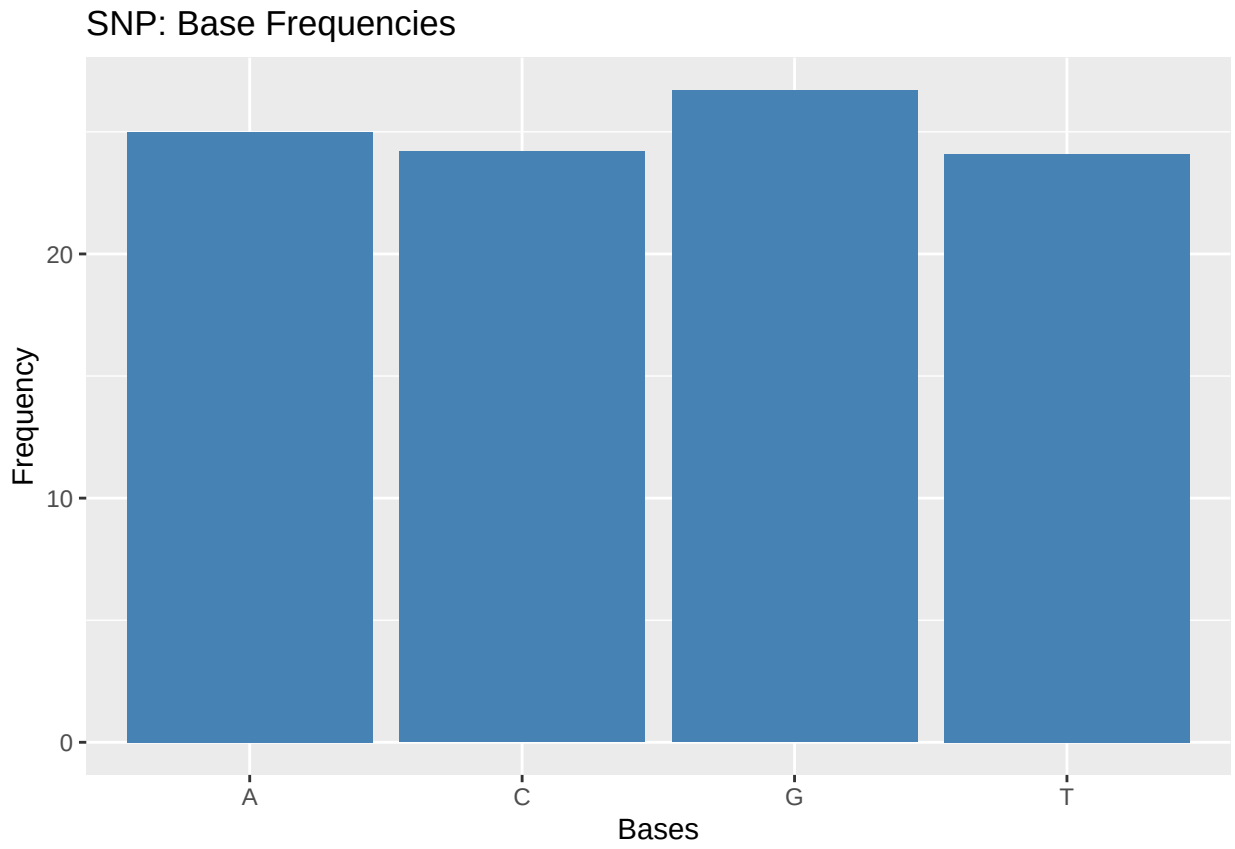
```
## Starting gl.report.bases
##   Processing a SNP dataset
##   Counting the bases
##   Counting Transitions and Transversions
##   Average trimmed sequence length: 60.7 ( 20 to 69 )
##   Total number of trimmed sequences: 255
##   Base frequencies (%)
##     A: 25.01
##     G: 26.67
##     T: 24.12
##     C: 24.2
##
##   Transitions  : 65.49
##   Transversions: 34.51
##   tv/ts ratio: 1.8977
```



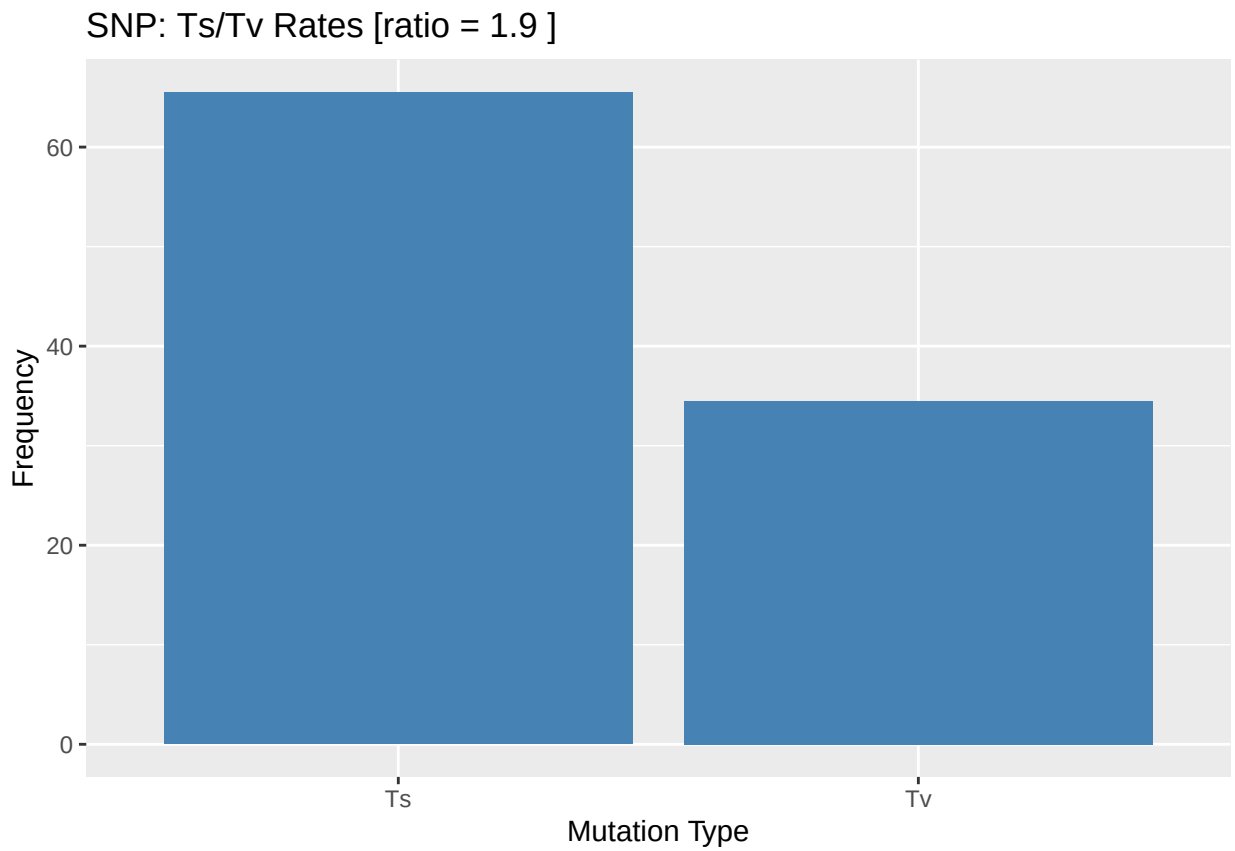



```
##   Creating an output vector to return
## Completed: gl.report.bases

## $freq
##     A     G     T     C    tv    ts
## 25.01 26.67 24.12 24.20 34.51 65.49
##
## $plotbases
```




```
##
## $plottstv
```



Monomorphic sites are parsimony uninformative, so a more compact output fastA file, for parsimony analyses, can be generated with

```
gl2fasta(gl, method=4, outfile="nohets.fasta")
```

12.4 Using Ambiguity Codes

Another common approach is to output the concatenated sequences using ambiguity codes for the heterozygous SNP bases. This can be done with

```
gl2fasta(gl, method=1, outfile="ambcodes.fasta")
```

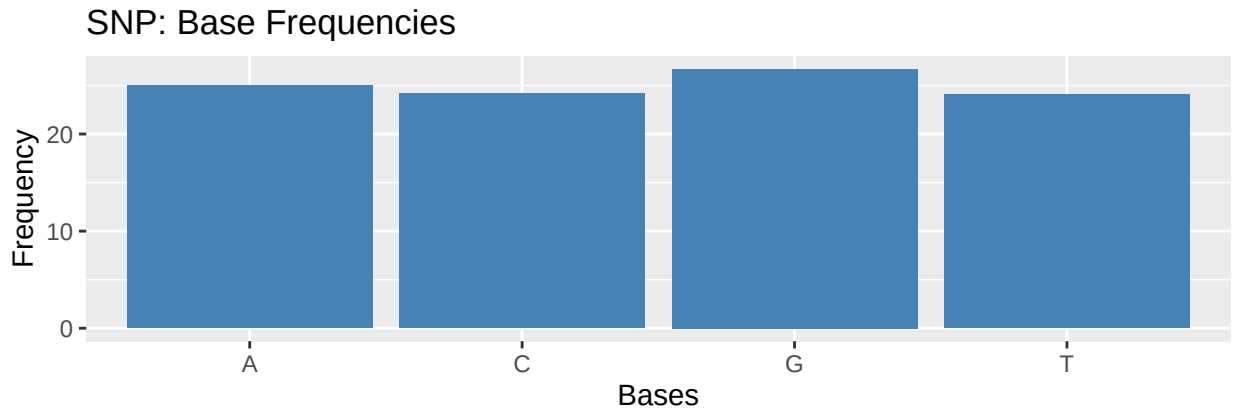
Maximum likelihood packages like RAxML Stamatakis (2014) cater for the ambiguities by adjusting tip likelihoods as described by Felsenstein (2004:255). Again, it is possible to output only the variable sites to the fastA file.

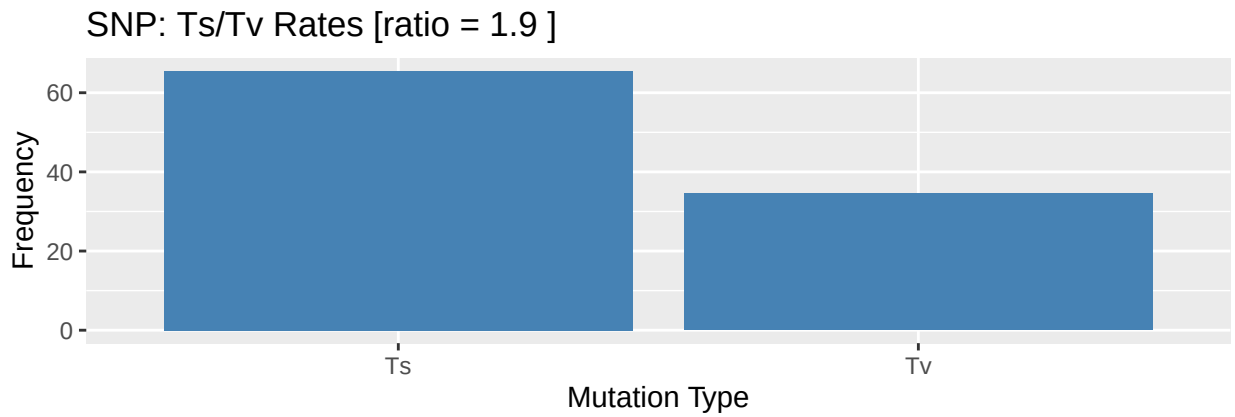
```
gl2fasta(gl, method=3, outfile="ambcodes.fasta")
```

These data can be used in some Maximum Likelihood software by providing base frequencies and transition and transversion ratios separately. They can be calculated with

```
gl.report.bases(testset.gl)
```

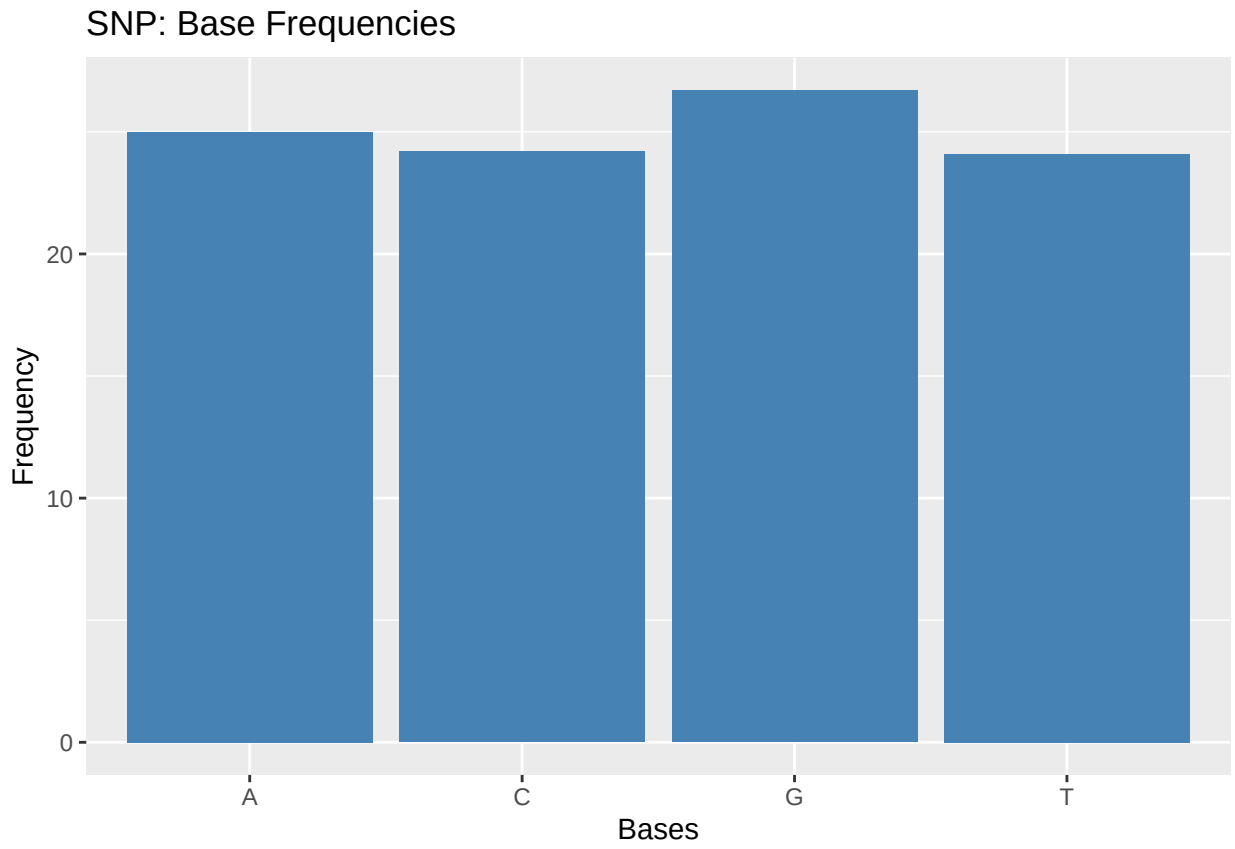
```
## Starting gl.report.bases
##   Processing a SNP dataset
##   Counting the bases
##   Counting Transitions and Transversions
##   Average trimmed sequence length: 60.7 ( 20 to 69 )
##   Total number of trimmed sequences: 255
##   Base frequencies (%)
##     A: 25.01
##     G: 26.67
##     T: 24.12
##     C: 24.2
##
##   Transitions  : 65.49
##   Transversions: 34.51
##   tv/ts ratio: 1.8977
```



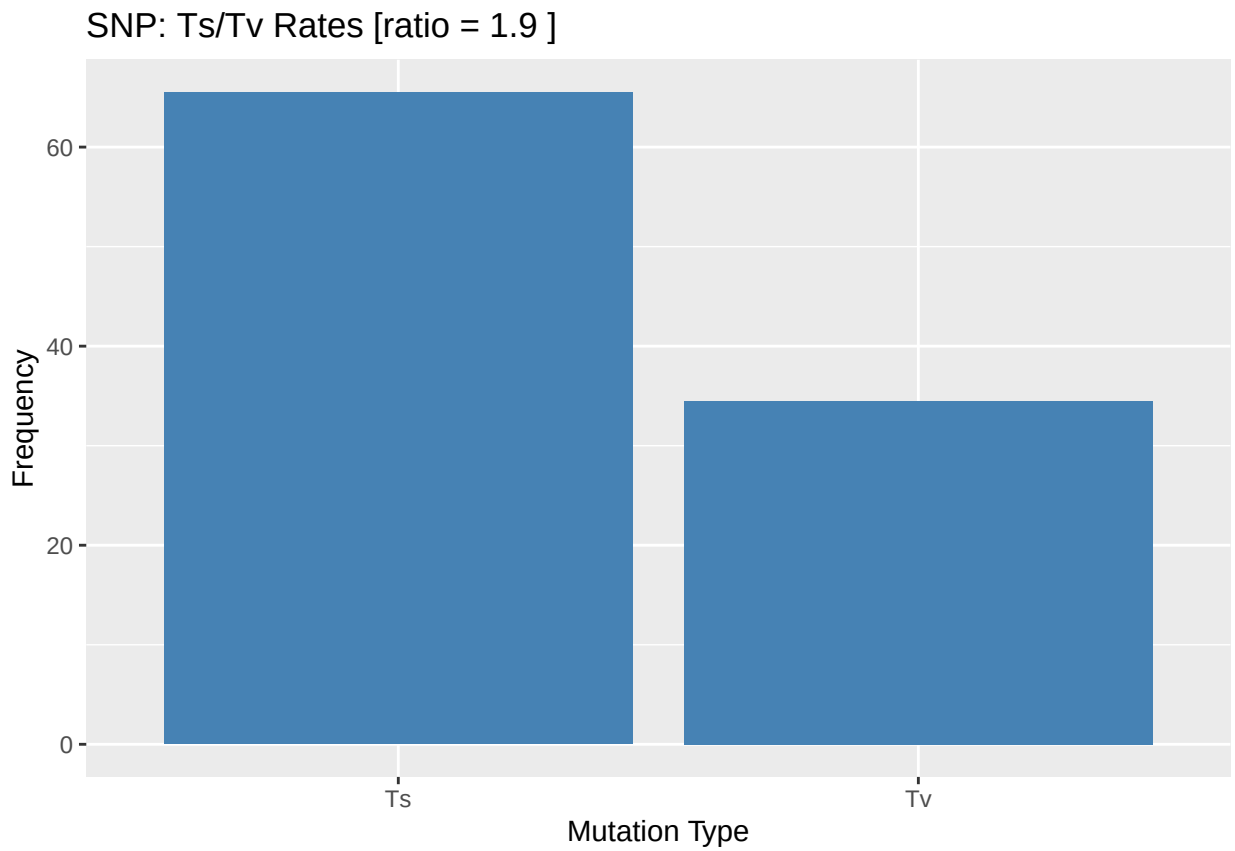



```
##   Creating an output vector to return
## Completed: gl.report.bases

## $freq
##     A     G     T     C    tv    ts
## 25.01 26.67 24.12 24.20 34.51 65.49
##
## $plotbases
```




```
##
## $plottstv
```



##SVD Quartets

Concatenating the sequence tags or using data comprising of the SNP locus alone has been heavily criticised in some quarters because the approach does not accommodate the possibility that the phylogeny for each SNP may depart from the species tree. Gene trees need to be estimated separately and then combined to estimate the species tree.

One such approach is to use SVD Quartets, that is, to estimate the relationships between taxa taken four at a time, then combine these to estimate the species tree.

This is implemented in dartR using script gl2svdquartets(), which generates the nexus file for input to PAUP for the analysis.

There are two approaches to preparing the data. The first enters the genotypes using two lines, one for reference state, one for alternate state. The second approach enters one genotype per line, accommodating the heterozygosity by using standard ambiguity codes. These are given as options in gl2svdquartets().

If using either of these approaches, it is important to filter out secondary SNPs from sequence tags using gl.filter.secondaries(), prior to analysis.

13 Population Assignment

Assigning individuals of unknown provenance to populations of known provenance is a challenging exercise, and several approaches have been suggested. Perhaps the simplest is to calculate the probabilities of yielding the observed genotype of the unknown individual given the observed allele frequencies in each the target population. Using this approach, the individual is assigned notionally to those populations for which this probability is highest; populations for which the probability is lower than some level of significance are

eliminated from further consideration. This approach was first applied in a study of microsatellite markers in bear populations (Paetkau et al. 2004) and subsequently applied using classical and Bayesian approaches to estimating probabilities (Goetz and Thaller 1998; Blanchong et al. 2002).

Unfortunately, the sheer number of SNPs generated by next generation sequencing technologies, often in the 10s or 100s of thousands, makes the assumption of independence of the loci untenable. Linkage is the problem and for most organisms, there is insufficient genomic knowledge to overcome this directly. Similarly, the sample sizes used in studies of population assignment are typically small, and inappropriate for tests of sufficient power to identify loci that can be regarded as independent. Lack of independence is a problem computationally because it is a prerequisite for combining the probabilities (or likelihoods) of the observed genotype at each locus as a product (or by summation of log values) to yield an overall probability of assignment for the unknown genotype. We have taken an alternate approach.

The approach taken here is to first eliminate from consideration those target populations where a SNP allele is present in the unknown individual but not in the target. When the unknown individual possesses such a private allele, the target population is unlikely to be the source population. This analysis can be done with script `gl.report.pa`. The analysis is also undertaken by `gl.assign`.

In many cases, examining private alleles will narrow down the possible source populations considerably, and depending on the spatial resolution required for the assignment (say, Australia or New Guinea), may provide a satisfactory answer.

A second approach is to examine the position of the unknown individual relative to the target populations in a reduced ordinated locus space using PCoA. This graphic representation is provided by gl.assign(). Addition of confidence ellipses then allows a decision to eliminate some populations from consideration as the source of the unknown individual.

The converse is not true. This approach does not allow assignment of the unknown to populations that contain the unknown within their confidence ellipse. The overall confidence envelope is multidimensional, and separation of the unknown from a target population may occur in deeper dimensions. Hence, as with the private alleles approach, this graphical approach serves to narrow down the candidates for the source of the unknown, and may in that sense, provide a satisfactory answer.

A third approach is to address the issue of non-independence (linkage) among the SNP loci by ordinating the space defined by those loci. The resultant axes, linear combinations of the information contained in each locus, are orthogonal and so can be regarded as independent. Subsequent standardization can achieve independent and identically distributed variates, which simplifies analysis of probabilities and likelihoods.

The script `gl.assign` first eliminates populations on the basis of private alleles. It then ordinates the space defined in locus space for the remaining populations.

A limited number of dimensions is retained in the final solution, the decision based on consideration of (a) the number of substantive eigenvalues (greater in explanatory power than the original variables before ordination), (b) the number of populations including the unknown, (c) an operational maximum number of dimensions specified in the code (dim=7) or (d) a user specified value. The script selects the minimum of these values to set the dimension of the reduced ordination space used subsequently.

A 95% confidence envelope (or some other level of confidence specified by the user) is defined in the reduced ordinated space, and the likelihood of the unknown genotype occurring is estimated for each dimension under Normal distribution assumptions. These likelihoods are logged for computational reasons, weighted by the eigenvalue for their respective dimension, and summed to yield an Assignment Index for the unknown against each population. Summing the weighted logged likelihoods is supported by the independence of each of the ordinated axes, but the result should be nevertheless regarded as an assignment index rather than an accurate likelihood.

An Assignment Index is calculated in the same way for a notional individual residing on the boundary of the confidence envelope. Comparing the assignment index for each population with that of the notional boundary individual provides a basis for a decision on assignment. If the Assignment Index for the unknown is less than the critical value for the Assignment Index (that of the boundary individual), then the unknown

is assigned to that population. Where more than one population is selected, the population with the greatest Assignment Index is the most likely. This will become evident in the examples that follow.

```
x <- gl.assign(testset.gl, unknown = "UC_00146", nmin=10, t=0)
```

```
## Starting gl.assign
##   Processing a SNP dataset
## Starting utils.pa.ind
##   Processing a SNP dataset
##   Retaining 20 populations with sample size greater than or equal to 10 : EmmacBrisWive EmmacBurdMis
##
##   Discarding 10 populations with sample size less than 10 : EmmacClarJack EmmacClarYate EmmacMDBGwyd
##
##   Assigning 1 unknown individual(s) to 20 target populations
##   Unknown individual: UC_00146
##   Total number of SNP loci:  255
##
##   Table showing number of loci with private alleles
##     0 EmmacMaclGeor
##     1 EmmacBurnBara EmmacRichCasi EmmacTweeUki
##     2 EmmacBrisWive EmmacMDBBowm EmmacMDBCudg EmmacMDBForb EmmacMDBMaci EmmacMDBMurrMung
##     3 EmmacFitzAllig EmmacMDBCond EmmacMDBSanf EmmacRussEube
##     4 EmmacBurdMist EmmacJohnWari EmmacRoss
##     6 EmmacCoopEulb
##     7 EmmacCoopCully
##     16 EmmacCoopAvin
##
##   Data retained for the unknown individual and remaining candidate source populations (zero loci wit
##    EmmacMaclGeor unknown
##
## Completed: utils.pa.ind
## There are no further populations to compare for assignment. EmmacMaclGeor is the best assignment
```

In this example, the unknown individual, UC_00146, is assigned to only one population, EmmaclGeor, so the assignment problem is resolved. This individual does indeed come from the Georges Creek, a tributary of the Macleay River.

If we are worried that the 1 locus with a private allele out of 255 loci for populations EmmacBurnBara, EmmacRichCasi, and EmmacTweeUki is less than substantial evidence, we can proceed to analyse the data further using `gl.assign`. We would take this step also if there were more than one population scoring 0 private alleles. The code below yields the results of the private alleles analysis again, but in addition computes assignment based on confidence envelopes:

```
x <- gl.assign(testset.gl, unknown ="UC_00146", nmin=10, alpha=0.95, t=1)
```

```
## Starting gl.assign
##   Processing a SNP dataset
## Starting utils.pa.ind
##   Processing a SNP dataset
##   Retaining 20 populations with sample size greater than or equal to 10 : EmmacBrisWive EmmacBurdMis
##
##   Discarding 10 populations with sample size less than 10 : EmmacClarJack EmmacClarYate EmmacMDBGwyd
##
```

```
##   Assigning 1 unknown individual(s) to 20 target populations
##   Unknown individual: UC_00146
##   Total number of SNP loci:  255
##
##   Table showing number of loci with private alleles
##     0 EmmacMaclGeor
##     1 EmmacBurnBara EmmacRichCasi EmmacTweeUki
##     2 EmmacBrisWive EmmacMDBBowm EmmacMDBCudg EmmacMDBForb EmmacMDBMaci EmmacMDBMurrMung
##     3 EmmacFitzAllig EmmacMDBCond EmmacMDBSanf EmmacRussEube
##     4 EmmacBurdMist EmmacJohnWari EmmacRoss
##     6 EmmacCoopEulb
##     7 EmmacCoopCully
##     16 EmmacCoopAvin
##
##   Data retained for the unknown individual and remaining candidate source populations (zero loci wit
##    EmmacMaclGeor unknown
##
## Completed: utils.pa.ind
## There are no further populations to compare for assignment. EmmacMaclGeor is the best assignment
```

The plot allows us to eliminate populations EmmacRichCasi and EmmacBurnBara from consideration as the unknown individual does not fall within or near their confidence ellipses. EmmacTweeUki and EmmacMaclGeor remain in contention.

Output from further analysis using the ordination and assignment index yields the following. Both EmmacTweeUki and EmmacMaclGeor are potential sources for the unknown individual, but the most likely source is EmmacMaclGeor, as supported by the private allele analysis.

A critical issue is whether the number of individuals in the target populations are sufficient to characterize them in the critical considerations made here. Is the sample size sufficient to support the identification of a private allele in the unknown? Is it sufficient to confidently construct confidence ellipses in the PCoA plot? Is it sufficient to provide a robust estimate of the distribution of individuals along each axis of the ordination in order to adequately estimate the likelihood of the unknown on that axis? We have set a default of nmin=10, but this is a matter of judgement that needs to be considered when planning a study.

14 Sex Linkage

Sex linked SNP markers can obtained from genotypic profiles for known males and known females. Be aware that the filters used by DArT Pty Ltd to generate the most reliable set of markers can apply tests that eliminate sex linked markers and other markers that show abberrant patterns of inheritance. It is necessary to ask DArT to generate a dataset fit for purpose, where they relax some of their filters.

Putative sex linked markers in a SNP dataset are those that are universally homozygous in the homogametic sex and universally heterozygous in the heterogametic sex. The most promising putative sex linked markers will have an AvgCount for the reference or alternate SNP of 10 or more, approximately one half in the other SNP state.

The function gl.sexlinkage(gl) will identify these markers and provide diagnostics in the form of AvgCount. The putative markers can then be tested more thoroughly by examining where they physically occur on the chromosomes and by developing appropriate PCR tests that can be applied to the panel of known sex individuals.

15 Conversion to formats for other packages

The genlight objects for managing SNP data are a relatively recent development, and the analysis options are limited. Conversion to {adegenet} genind object opens up a greater range of analysis options, and this conversion can be achieved with

```
gl <- testset.gl
gi <- gl2gi(gl, v=0)
```

Conversion in the opposite direction can be achieved with

```
gl2 <- gi2gl(gi)
```

15.1 Interfaces to Other Software

Package `dartR` does not pretend to provide a comprehensive range of analyses, but rather to provide avenues from SNP data stored as a genlight object to other available software packages. The following conversion functions are available:

function	explanation
gl2fasta()	Outputs the concatenated trimmed sequences to fastA format after first converting heterozygous SNPs to ambiguity codes or randomly assigning the heterozygous state to one or the other homozygous states (diplotypes to haplotypes).
gl2genind()	Converts a genlight object to a genind object as defined by the {adegenet} package.
genind2gl()	Converts a genind object as defined by the {adegenet} package to a genlight object.
gl2nhyb()	Outputs 200 loci selected according to user specified criteria for input to the package NewHybrids (Anderson and Thompson, 2002).

gl2faststructure() gl2 gdsfmt() | Outputs a gl object to a file in gds format that can subsequently be used with {SNPRelate}.

15.2 NewHybrids

NewHybrids (Anderson and Thompson, 2002) employs a statistical method for identifying species hybrids using data on multiple, unlinked markers which does not require that allele frequencies be known in the parental species. The probability model used is one in which parentals and various classes of hybrids (F1s, F2s, and various backcrosses) form a mixture from which the sample is drawn. Using the framework of Bayesian model-based clustering, NewHybrids computes, using Markov chain Monte Carlo, the relative likelihood (posterior probability) that each individual belongs to each of the distinct hybrid classes. NewHybrids is limited to ca 200 loci because of memory constraints, so the best 200 available loci were selected on the basis of their being different and fixed in each of the two nominated parental populations, and then on avgPIC.

To run a NewHybrids analysis you need to have installed NewHybrids (http://ib.berkeley.edu/labs/slatkin/eriq/software/software.htm#NewHybs)[http://ib.berkeley.edu/labs/slatkin/eriq/software/software.htm#NewHybs].

```
glnew <- gl.nhybrids(testset.gl)
```

Refer to help (?gl.nhybrids) for further information. This function compares two sets of parental populations to identify loci that exhibit a fixed difference, returns an genlight object with the reduced data, and creates an input file for the program NewHybrids using the top 200 loci. In the absence of two identified parental populations, the script will select a random set 200 loci only (method=random) or the first 200 loci ranked on information content (AvgPIC).

The resultant NewHybrids input file is then passed to the program NewHybrids outside the R environment. Refer to the NewHybrids documentation to continue.

If you have New Hybrids installed on your machine, then providing the directory with the New Hybrids executable (e.g. C:/workspace/R_analysis/NewHybsPC) will result in the analysis being executed. The advantage of this is that the final output is a little more elaborative.

15.3 Phylip

PHYLIP (Felsenstein, 1989) is a package of programs for inferring phylogenies (evolutionary trees). Methods that are available in the package include parsimony, distance matrix, and likelihood methods, including bootstrapping and consensus trees. The {dartR} scripts produce Euclidean distance matrices in a form that can be input to Phylip distance analyses (e.g. program Fitch).

To generate an input matrix for Phylip, use the function

```
glnew <- gl2phylip(outfile = "phyinput.txt")
```

To generate an input file containing resampled input matrices for the purposes of bootstrapping, use:

```
gl.new <- gl2phylip(outfile = "phyinput.txt", bstrap = 1000)
```

The resultant output file can be passed to Phylip programs for execution. Refer to the Phylip documentation.

15.4 SNPRelate

R package SNPRelate is available to undertake principal components analysis and relatedness analysis (Zheng et al., 2012). {SNPRelate} expects the data to be in gds format. As with the genlight format of package {adegenet}, the gds format supports efficient memory management and storage of SNP genotypes and associated metadata. In this format each byte encodes up to four SNP genotypes thereby reducing file size and access time. The GDS format supports data blocking so that only the subset of data that is being processed needs to reside in memory. GDS formatted data is also designed for efficient random access to large data sets.

Data conversion from a genlight object to gds format is via

```
gl2gds(gl, outfile="test.gds")
```

Once this file is created, you can open it for analysis with {SNPRelate} using

```
gds <- snpgdsOpen("gl2gds.gds")
```

and undertake the range of analyses available in {SNPRelate}, including

option	explanation
LD-pruning	A pruned set of loci which are in approximate linkage equilibrium will avoid the strong influence of locus clusters on relatedness analysis
PCoA	with some nice diagnostic plots
Relatedness	Identity-by-descent estimation can be done by either the method of moments (MoM) or maximum likelihood estimation (MLE)
IBS	Identity by State analysis and Multidimensional Scaling (MDS) for displaying relationships

15.5 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="myfile.fs", probar = FALSE)
```

```
## Starting gl2faststructure
##   Processing a SNP dataset
## Saved faststructure file: C:\Users\s425824\AppData\Local\Temp\Rtmp61wFlf/myfile.fs
## Completed: gl2faststructure
```

```
## NULL
```

16 References

Anderson, E.C., Thompson, E.A., 2002. A Model-Based Method for Identifying Species Hybrids Using Multilocus Genetic Data. Genetics 160, 1217-1229.

Blanchong, J. A., Scribner, K. T., and Winterstein, S. R. (2002). Assignment of Individuals to Populations: Bayesian Methods and Multi-Locus Genotypes. The Journal of Wildlife Management 66, 321. doi:10.2307/3803164

Buneman, P., 1973. A note on the metric properies of trees. J Combinarorial Theory 17B, 48-50.

Cattell, R.B., 1966. The scree test for the number of factors. Multivariate Behavioral Research 1, 245-276. doi:10.1207/s15327906mbr0102_10

Felsenstein, J., 2004. Inferring Phylogenies. Sinauer Associates, Sunderdland, MA USA.

Felsenstein, J., 1989. PHYLIP - Phylogeny Inference Package (Version 3.2). Cladistics 5, 164-166.

Georges, A., Adams, M., 1996. Electrophoretic delineation of species boundaries within the short-necked freshwater turtles of Australia (Testudines: Chelidae). Zoological Journal of the Linnean Society 118, 241-260.

Georges, A., Adams, M., 1992. A phylogeny for Australian chelid turtles based on allozyme electrophoresis. Australian Journal of Zoology 40, 453-476.

Götz, K.-U., and Thaller, G. (1998). Assignment of individuals to populations using microsatellites. Journal of Animal Breeding and Genetics 115, 53–61.

Gower, J.C., 1966. Some distance properties of latent root and vector methods used in multivariate analysis. Biometrika 53, 325–338. doi:10.1093/biomet/53.3-4.325

Jombart, T., 2008. adegenet: a R package for the multivariate analysis of genetic markers. Bioinformatics 24, 1403–1405. doi:10.1093/bioinformatics/btn129

Jombart, T., Ahmed, I., 2011. adegenet 1.3-1: new tools for the analysis of genome-wide SNP data. Bioinformatics 27, 3070–3071. doi:10.1093/bioinformatics/btr521

Paetkau, D., Slade, R., Burden, M., and Estoup, A. (2004). Genetic assignment methods for the direct, real-time estimation of migration rate: a simulation-based exploration of accuracy and power. Molecular Ecology 13, 55–65.

Porras-Hurtado, L., Ruiz, Y., Santos, C., Phillips, C., Carracedo, Á., Lareu, M.V., 2013. An overview of STRUCTURE: applications, parameter settings, and supporting software. Frontiers in Genetics 4. doi: 10.3389/fgene.2013.00098

Pritchard, J.K., Stephens, M., Donnelly, P., 2000. Inference of population structure using multilocus genotype data. Genetics 155, 945–959.

Porras-Hurtado, L., Ruiz, Y., Santos, C., Phillips, C., Carracedo, L., Lareu, M.V., 2013. An overview of STRUCTURE: applications, parameter settings, and supporting software. Frontiers in Genetics 4. doi: 10.3389/fgene.2013.00098

Raj, A., Stephens, M., Pritchard, J.K., 2014. fastSTRUCTURE: Variational Inference of Population Structure in Large SNP Data Sets. Genetics 197, 573–589. doi:10.1534/genetics.114.164350

Rogers, J.S., 1972. Measures of similarity and genetic distance. Studies in Genetics 7, 145–153.

Swofford, D.L., 2002. Phylogenetic Analysis Using Parsimony (*and Other Methods). Version 4. Sinauer Associates, Sunderland, Massachusetts, USA.

Stamatakis, A., 2014. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. Bioinformatics, 30(9), 1312-1313.

Swofford, D.L., Berlocher, S.H., 1987. Inferring evolutionary trees from gene frequency data under the principle of maximum parsimony. Systematic Zoology 36, 293–325.

Zheng, X., Levine, D., Shen, J., Gogarten, S.M., Laurie, C., Weir, B.S., 2012. A high-performance computing toolset for relatedness and principal component analysis of SNP data. Bioinformatics 28, 3326–3328.