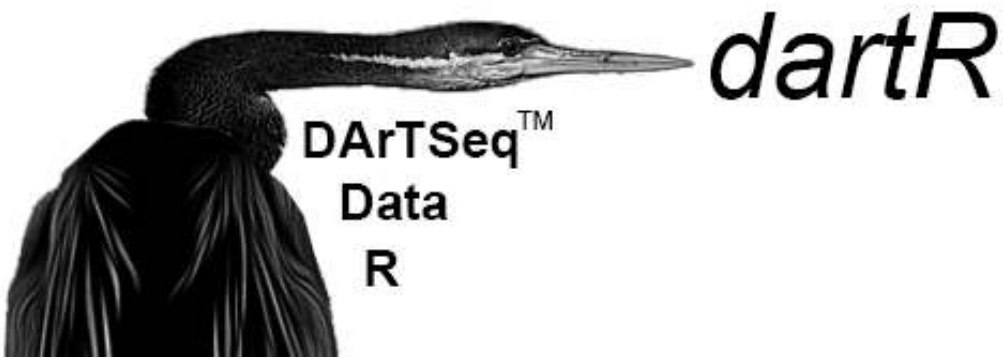


Introduction to dartR

Bernd Gruber, Peter Unmack, Oliver Berry & Arthur Georges

2017-06-24

Contents

1 What is dartR?

Package `dartR` is an R package for a) loading DArT ^{TM} 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. Though the majority of functions are.

In most cases, the scripts in {dartR} are wrappers for scripts included in other already available packages, to provide transparent 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 DArT datafiles in preparation for analysis by the vast suite of packages available in R through the CRAN repository, and make them easier accessible for users not deeply accustomed with the R language.

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) and Neighbour-joining trees.
- 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).

2 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 {dartR} package visit the github page for instructions (https://github.com/green-striped-gecko/dartR) and make it available into the current project via library(dartR). A brief description on the installation procedure is given below:

```
# Install and attach library dartR
  install.packages("devtools")
  library(devtools)
  source("http://bioconductor.org/biocLite.R")
  biocLite("qvalue", suppressUpdates=T)
  biocLite("SNPRelate", suppressUpdates=T)
  install_github("whitlock/OutFLANK")
  install_github("green-striped-gecko/dartR")
```

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

```
 library(dartR)
```

```
## Loading required package: adegenet

## Loading required package: ade4

## Warning: package 'ade4' was built under R version 3.3.3

##
##    /// adegenet 2.0.1 is loaded ////////////
```

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

[Please note library adegenet will output some lines as it is not possible to easily suppress those lines]

If some packages have not been loaded, an error message will be given and you will need to install the package manually.

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/")
```

The dartR library contains test datasets that form the basis of the commands and exercises below. We rename the test data set to `gl`, in case you want to run the code with your data, you simply can rename your data set into `gl`, once you 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 replicate the output.

3 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) on 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-A/G 100049698|16-C/T 100049728|23-T/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 list 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 list of individual metadata (individual id, population, sex, latitude, longitude, etc) Each item in this list (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 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/C").
- 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) on Analysing genome-wide SNP data using adegenet 2.0.0.

4 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.

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

```
## Topskip not provided. Try to guess topskip...
## Set topskip to  3 . Trying to proceed...
## Trying to determine if one row or two row format...
## Found  2  row(s) format. Proceed...
## Added the following covmetrics:
## AlleleID SNP SnpPosition CallRate OneRatioSnp FreqHomRef FreqHomSnp FreqHets AvgCountRef AvgCountSnp
## 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
## Start conversion....
## Format is 2 rows.
## Please note conversion of bigger data sets will take some time!
## Once finished, we recommend to save the object using save(object, file="object.rdata")
## Try to add covariate file: C:/Users/s425824/AppData/Local/Temp/RtmpMbk9lR/Rinst2e3454143603/dartR/ex
## Ids of covariate file (at least a subset of) are matching!
## Found  250 matching ids out of 250 ids provided in the covariate file. Subsetting snps now!.
```

```
##  Added pop factor.
## Added latlon data.
## Added  id  to the other$ind.metrics slot.
##  Added  pop  to the other$ind.metrics slot.
##  Added  lat  to the other$ind.metrics slot.
##  Added  lon  to the other$ind.metrics slot.
##  Added  sex  to the other$ind.metrics slot.
##  Added  maturity  to the other$ind.metrics slot.
```

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 thefollowing options:

Display the structure of the genlight object

```
gl
```

```
##  /// GENLIGHT OBJECT /////////
##
##  // 250 genotypes,  255 binary SNPs, size: 588.1 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
```

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

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

```
##          100049687-12-A/G 100049698-16-C/T 100049728-23-T/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
```

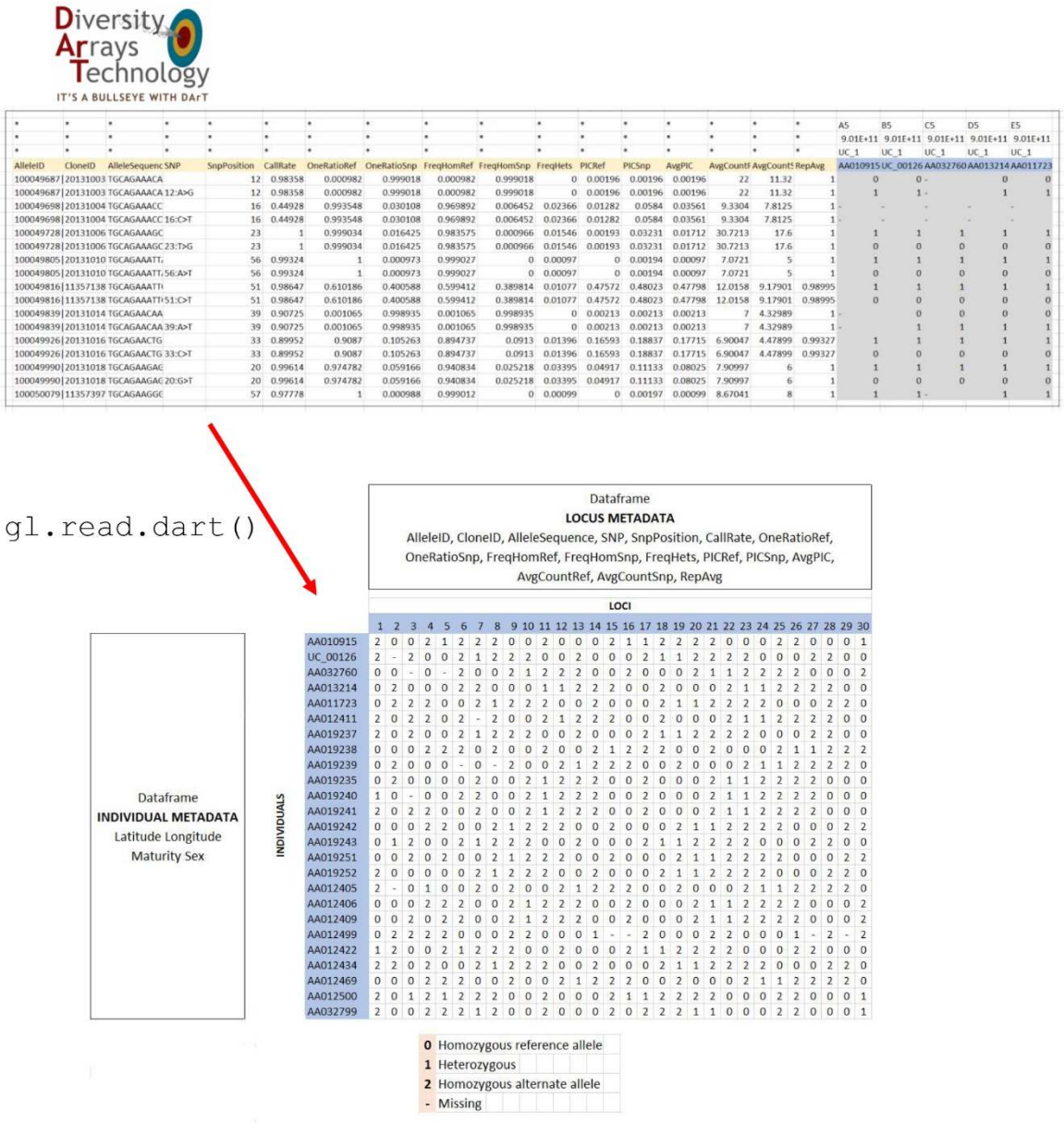
List the population labels (only the first 5)

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

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

6 Working with DArT Genlight Objects

Genelight 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.



6.1 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 csv file when it is read in to the genlight object. The locus metadata are held in an R dataframe 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
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	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 part of a dataframe 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
##  [8] 1.000000 1.000000 0.980000
```

You can check the names of all available loc.metrics via. Depending on the report from DarT you may have additional (fewer) loc.metrics (e.g. Trimmed Sequence is available on request).

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

```
##  [1] "AlleleID"    "SNP"         "SnpPosition" "CallRate"    "OneRatioSnp"
##  [6] "FreqHomRef"  "FreqHomSnp"  "FreqHets"    "AvgCountRef" "AvgCountSnp"
## [11] "RepAvg"      "clone"       "uid"
```

Task

Try this and other similar statements to interrogate the gl object by submitting commands from the R Editor.

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 maximum likelihood phylogeny 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.

6.2 Individual Metadata

Individual (=specimen) 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 shouns
gl@other$ind.metrics$sex[1:10]
```

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

7 Subsetting and Recoding Data

7.1 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.dups()	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.dups()	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.

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

7.2 Examples of dartR code to filter a gl dataset

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

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

```
## Reporting for a genlight object
## Note: Missing values most commonly arise from restriction site mutation.
##
## Initial no. of loci = 255
```

```
##   No. of loci deleted = 82
## Summary of filtered dataset
##   Call Rate > 0.95
##   No. of loci: 173
##   No. of individuals: 250
##   No. of populations:  30
```

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

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

```
## Reporting for a genlight object
## Note: Missing values most commonly arise from restriction site mutation.
##
## Initial no. of individuals = 250
## Filtering a genlight object
##   no. of individuals deleted = 206
## Individuals retained = 44
## List of individuals deleted because of low call rate
##  AA010915 AA032760 AA011723 AA012411 AA019237 AA019238 AA019239 AA019235 AA019240 AA019241 AA019243 A
##    from populations
##  EmmacMDBForb EmmacMDBMaci EmmacBurnBara EmmacCoopEulb EmmacBurdMist EmmacBurdMist EmmacBurdMist Emma
## Summary of filtered dataset
##   Call Rate > 0.9
##   No. of loci: 255
##   No. of individuals: 44
##   No. of populations:  17
```

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

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

```
## Reporting for a genlight object
## Note: RepAvg is a DArT statistic reporting reproducibility averaged across alleles for each locus.
##
## Initial no. of loci = 255
## No. of loci deleted = 41
## Summary of filtered dataset
##   Reproducibility >= 1
##   No. of loci: 214
##   No. of individuals: 250
##   No. of populations:  30
```

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

5. Filter out monomorphic loci

```
gl2 <- gl.filter.monomorphs(gl)
```

```
## Identifying monomorphic loci
##
## Breakdown of 255 loci
##   Polymorphic loci: 111 retained
##   Monomorphic loci: 144 deleted
##   Loci with no scores (all NA): 0 deleted
```

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

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

```
## Analysing a genlight object
## Hamming distance ranges from zero (sequence identity) to 1 (no bases shared at any position)
## Calculating pairwise Hamming distances between trimmed reference sequence tags
##
##
## Summary of filtered dataset
##   Initial No. of loci: 255
##   Hamming d > 0.25
##   Loci deleted 17
##   Final No. of loci: 238
##   No. of individuals: 250
##   No. of populations:  30
```

Note: This filter and its accompanying report function are very 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`. Though it is a bit of a waste in terms of memory, but avoids confusion which filter you have already applied to create a new object after each filter. 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)
```

7.3 Population (=higher level grouping) reassignment

Recall that the metafile contains information assigned to each individual including, often a minimum, population assignments. Population recode tables are csv files (comma delimited text files) that can be used to reassign individuals to different populations (groups), thus amalgamating populations or deleting populations from the genlight object.

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

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

```
##  [1] "EmmacBrisWive"    "EmmacBurdMist"    "EmmacBurnBara"
##  [4] "EmmacClarJack"    "EmmacClarYate"    "EmmacCoopAvin"
##  [7] "EmmacCoopCully"   "EmmacCoopEulb"    "EmmacFitzAllig"
## [10] "EmmacJohnWari"    "EmmacMDBBowm"     "EmmacMDBCond"
## [13] "EmmacMDBCudg"     "EmmacMDBForb"     "EmmacMDBGwyd"
## [16] "EmmacMDBMaci"     "EmmacMDBMurrMung" "EmmacMDBSanf"
## [19] "EmmacMaclGeor"    "EmmacMaryBoru"    "EmmacMaryPetr"
## [22] "EmmacNormJack"    "EmmacNormLeic"    "EmmacNormSalt"
## [25] "EmmacRichCasi"    "EmmacRoss"        "EmmacRussEube"
## [28] "EmmacTweeUki"     "EmsubRopeMata"    "EmvicVictJasp"
```

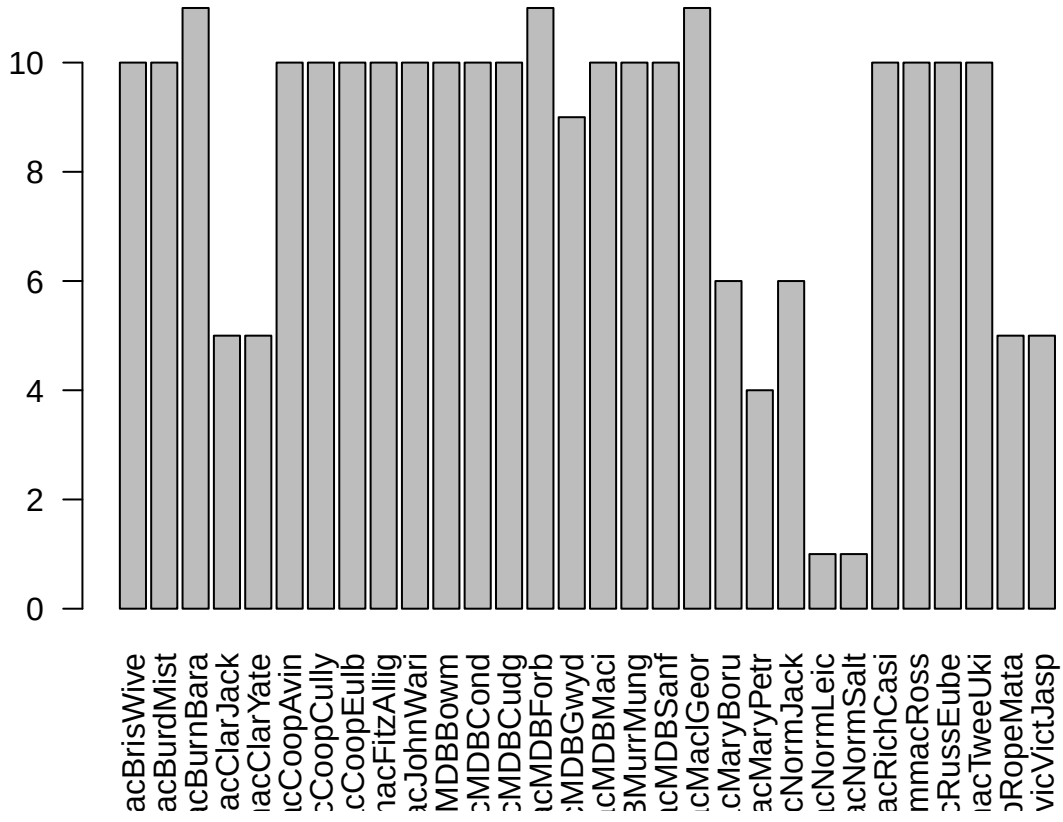
```
#table on individuals per population
table(pop(gl))
```

```
##
##    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
```

As a demonstration of the great ability of R it is easy to create a barplot on the number of individuals per population:

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



You can reassign individuals to new populations using a recode table. The quickest way to construct a recode table for an active genlight object is using

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

```
## Proforma recode table written to: C:\Users\s425824\AppData\Local\Temp\RtmpQRwb3B/new_pop_assignments
```

```
##  [1] "EmmacBrisWive"    "EmmacBurdMist"    "EmmacBurnBara"
##  [4] "EmmacClarJack"    "EmmacClarYate"    "EmmacCoopAvin"
##  [7] "EmmacCoopCully"   "EmmacCoopEulb"    "EmmacFitzAllig"
## [10] "EmmacJohnWari"    "EmmacMDBBowm"     "EmmacMDBCond"
## [13] "EmmacMDBCudg"     "EmmacMDBForb"     "EmmacMDBGwyd"
## [16] "EmmacMDBMaci"     "EmmacMDBMurrMung" "EmmacMDBSanf"
## [19] "EmmacMaclGeor"    "EmmacMaryBoru"    "EmmacMaryPetr"
## [22] "EmmacNormJack"    "EmmacNormLeic"    "EmmacNormSalt"
## [25] "EmmacRichCasi"    "EmmacRoss"        "EmmacRussEube"
## [28] "EmmacTweeUki"     "EmsubRopeMata"    "EmvicVictJasp"
```

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.

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. This editing is best done in Excel.

The population reassignments are then applied using:

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

```
## Processing genlight object
##   Reassigning entities to populations as per  C:\Users\s425824\AppData\Local\Temp\RtmpQRwb3B/new_pop
##   Removing entities flagged for deletion in  C:\Users\s425824\AppData\Local\Temp\RtmpQRwb3B/new_pop_a
## Identifying monomorphic loci
##
## Breakdown of 255 loci
##   Polymorphic loci: 111 retained
##   Monomorphic loci: 144 deleted
##   Loci with no scores (all NA): 0 deleted
## Summary of recoded dataset
##   No. of loci: 111
##   No. of individuals: 250
##   No. of populations:  30
```

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

```
levels(pop(gl))
```

```
##  [1] "EmmacBrisWive"    "EmmacBurdMist"    "EmmacBurnBara"
##  [4] "EmmacClarJack"    "EmmacClarYate"    "EmmacCoopAvin"
##  [7] "EmmacCoopCully"   "EmmacCoopEulb"    "EmmacFitzAllig"
## [10] "EmmacJohnWari"    "EmmacMDBBowm"     "EmmacMDBCond"
## [13] "EmmacMDBCudg"     "EmmacMDBForb"     "EmmacMDBGwyd"
## [16] "EmmacMDBMaci"     "EmmacMDBMurrMung" "EmmacMDBSanf"
## [19] "EmmacMaclGeor"    "EmmacMaryBoru"    "EmmacMaryPetr"
## [22] "EmmacNormJack"    "EmmacNormLeic"    "EmmacNormSalt"
## [25] "EmmacRichCasi"    "EmmacRoss"        "EmmacRussEube"
## [28] "EmmacTweeUki"     "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, and probably the easiest, is to use:

```
glnew2 <- gl.edit.recode.pop(gl, pop.recode = file.path(tempdir(),"new_pop_assingments.csv"))
```

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))`.

7.4 Deleting populations

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.

7.5 Relabeling individuals

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=file.path(tempdir(),"new_ind_assignments.csv"))
```

```
## Proforma recode table written to: C:\Users\s425824\AppData\Local\Temp\RtmpQRwb3B/new_ind_assignments
```

```
##   [1] "AA010915"  "UC_00126"  "AA032760"  "AA013214"  "AA011723"
##   [6] "AA012411"  "AA019237"  "AA019238"  "AA019239"  "AA019235"
##  [11] "AA019240"  "AA019241"  "AA019242"  "AA019243"  "AA019251"
##  [16] "AA019252"  "AA012405"  "AA012406"  "AA012409"  "AA012499"
##  [21] "AA012422"  "AA012434"  "AA012469"  "AA012500"  "AA032799"
##  [26] "AA032826"  "AA010795"  "AA010796"  "AA032800"  "AA032801"
##  [31] "AA032808"  "AA032809"  "AA032811"  "AA032812"  "AA032822"
##  [36] "AA032825"  "AA010797"  "AA010752"  "AA010754"  "AA010756"
##  [41] "AA010798"  "AA010799"  "AA010800"  "AA010802"  "AA010803"
##  [46] "AA010804"  "AA010809"  "AA010749"  "AA010758"  "AA010763"
##  [51] "AA010765"  "AA010771"  "AA010772"  "AA010781"  "AA032762"
##  [56] "AA032763"  "AA032756"  "AA032757"  "AA032758"  "AA032761"
##  [61] "AA032765"  "AA010931"  "AA010937"  "AA010940"  "AA032764"
##  [66] "AA032768"  "AA010936"  "AA010909"  "AA010916"  "AA010917"
##  [71] "AA010920"  "AA010921"  "AA020651"  "AA020652"  "AA020667"
##  [76] "AA020669"  "AA020655"  "AA020656"  "AA020644"  "AA020645"
##  [81] "AA020646"  "AA020649"  "AA013203"  "AA013217"  "AA013220"
##  [86] "AA013202"  "AA013225"  "AA018496"  "AA018497"  "AA018513"
##  [91] "AA013231"  "AA013261"  "AA013265"  "AA013270"  "AA018492"
##  [96] "AA018493"  "AA018494"  "AA018495"  "AA018514"  "AA018515"
## [101] "AA018516"  "UC_00125"  "UC_00126a" "UC_00146"  "UC_00149"
## [106] "AA018640"  "AA018658"  "AA011729"  "UC_00132"  "UC_00137"
## [111] "UC_00143"  "UC_00157"  "UC_00161"  "AA018637"  "AA018638"
## [116] "AA018639"  "AA011731"  "AA033576"  "AA033577"  "AA011732"
## [121] "AA011737"  "AA011741"  "AA011744"  "AA011745"  "AA011746"
## [126] "AA011749"  "AA033575"  "AA033578"  "AA012411a" "AA033579"
## [131] "AA033582"  "AA033593"  "AA033602"  "AA033609"  "AA033617"
## [136] "AA010915a" "AA011723a" "AA019158"  "AA020379"  "UC_01044"
## [141] "AA018380"  "AA018371"  "AA004553"  "AA000328"  "AA000311"
## [146] "AA019159"  "AA020378"  "UC_01060"  "AA018379"  "AA018365"
## [151] "AA004554"  "AA000303"  "AA000320"  "AA019160"  "AA020377"
## [156] "UC_01053"  "AA018375"  "AA004555"  "AA000304"  "AA019165"
## [161] "AA019161"  "AA020376"  "UC_01062"  "AA018374"  "AA04523"
## [166] "AA000305"  "AA019164"  "AA020375"  "AA018373"  "AA032875"
## [171] "AA000309"  "AA019163"  "AA020374"  "AA018368"  "AA032878"
## [176] "AA000302"  "AA019162"  "AA020365"  "UC_00150"  "AA018369"
## [181] "AA004551"  "AA032880"  "AA000307"  "AA019156"  "AA020371"
## [186] "UC_01051"  "AA018370"  "AA004552"  "AA032882"  "AA000310"
## [191] "AA019157"  "AA019075"  "AA004864"  "AA019071"  "AA004868"
## [196] "AA019083"  "AA019072"  "AA004858"  "AA004869"  "AA019082"
## [201] "AA019073"  "AA004859"  "AA004866"  "AA019077"  "AA004860"
## [206] "AA019080"  "AA004861"  "AA019079"  "AA004862"  "AA019078"
## [211] "AA004863"  "UC_00267"  "UC_00205"  "UC_00206"  "UC_00208"
## [216] "UC_00243"  "UC_00209"  "UC_00254"  "UC_00210"  "UC_00259"
## [221] "UC_00126c" "AA063718"  "AA063720"  "AA063722"  "AA063726"
## [226] "AA063732"  "AA063708"  "AA063710"  "AA063712"  "AA063714"
## [231] "AA063716"  "AA020735"  "AA032442"  "AA032441"  "AA020749"
## [236] "AA020746"  "AA020744"  "AA020743"  "AA020739"  "AA020738"
## [241] "AA001451"  "AA01452"   "AA001454"  "AA001455"  "AA001446"
## [246] "AA001456"  "AA001447"  "AA001448"  "AA001449"  "AA001450"
```

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=file.path(tempdir(),"new_ind_assignments.csv"))
```

```
## Processing genlight object
##   Relabelling individuals (=specimens) as per  C:\Users\s425824\AppData\Local\Temp\RtmpQRwb3B/new_in
##   Removing entities flagged for deletion in  C:\Users\s425824\AppData\Local\Temp\RtmpQRwb3B/new_ind_
## Identifying monomorphic loci
##
## Breakdown of 255 loci
##   Polymorphic loci: 111 retained
##   Monomorphic loci: 144 deleted
##   Loci with no scores (all NA): 0 deleted
## Summary of recoded dataset
##   No. of loci: 111
##   No. of individuals: 250
##   No. of populations:  30
```

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=file.path(tempdir(),"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)`.

7.6 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)`.

7.7 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. For example:

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

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

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

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

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

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

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

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

```
## [1] 255  13
```

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] 147 227 139 116 229 170  76 223 237 176  46  86  27 246 210  21 185
## [18]  85  75 238  98 103  91 205 158 253  77   7 241 168
```

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: 105.4 Kb
##  55 (12.22 %) missing data
##
##  // Basic content
##    @gen: list of 15 SNPbin
##    @ploidy: ploidy of each individual  (range: 2-2)
##
##  // Optional content
##    @ind.names:  15 individual labels
##    @loc.names:  30 locus labels
##    @loc.all:  30 alleles
##    @position: integer storing positions of the SNPs
##    @pop: population of each individual (group size range: 5-10)
##    @other: a list containing: loc.metrics  latlong  ind.metrics
```

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

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

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

```
## [1] 30 13
```

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` 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 five individuals and the first 10 loci):

```
d <- gl.dist(gl[1:5,1:10])
```

```
## Using SNP data from a genlight object
##   Tallying allele frequencies, this may take some time
##   Calculation of allele frequencies complete
##   Calculating distances:  euclidean
##   Refer to dist {stats} documentation for algorithm
```

```
d
```

```
##               EmmacMDBForb EmmacMaclGeor EmmacMDBMaci EmmacMDBSanf
## EmmacMDBForb             0
## EmmacMaclGeor            0             0
## EmmacMDBMaci             0             0            0
## EmmacMDBSanf             0             0            0            0
## EmmacBurnBara            0             0            0            0
##               EmmacBurnBara
## EmmacMDBForb
## EmmacMaclGeor
## EmmacMDBMaci
## EmmacMDBSanf
## EmmacBurnBara             0
```

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.

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 <- gl
pop(glsub)<-recpops

gi <- gl2gi(glsub, probar = FALSE)
```

```
## Start conversion....
## Please note conversion of bigger data sets will take some time!
## Once finished, we recommend to save the object using >save(object, file="object.rdata")
##
## Matrix converted.. Prepare genind object...
## Finished! Took 1 seconds.
```

```
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)
```

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

The resultant object pc contains the eigenvalues, factor scores and factor loadings that can be accessed for subsequent analyses.

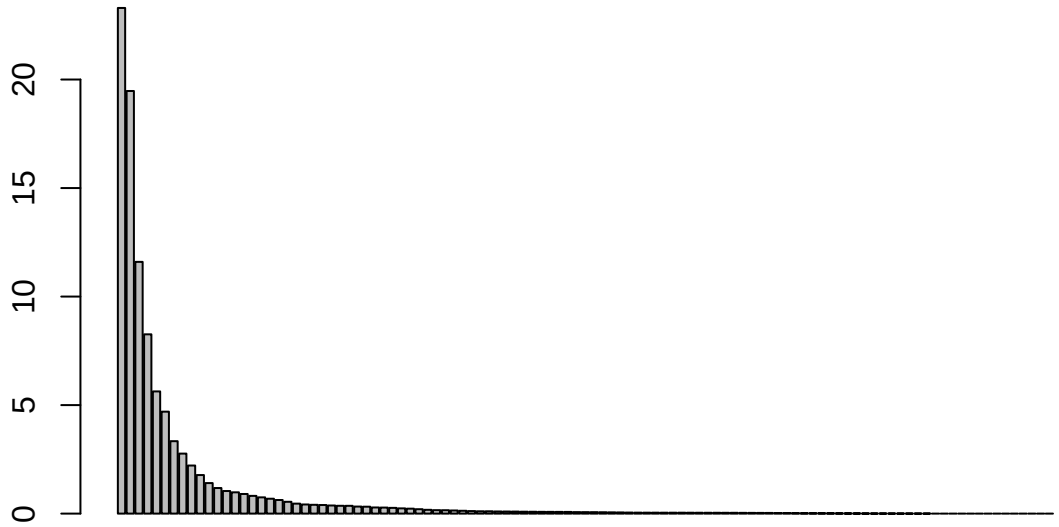
```
names(pc)
```

```
## [1] "eig"      "scores"   "loadings" "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, )
```



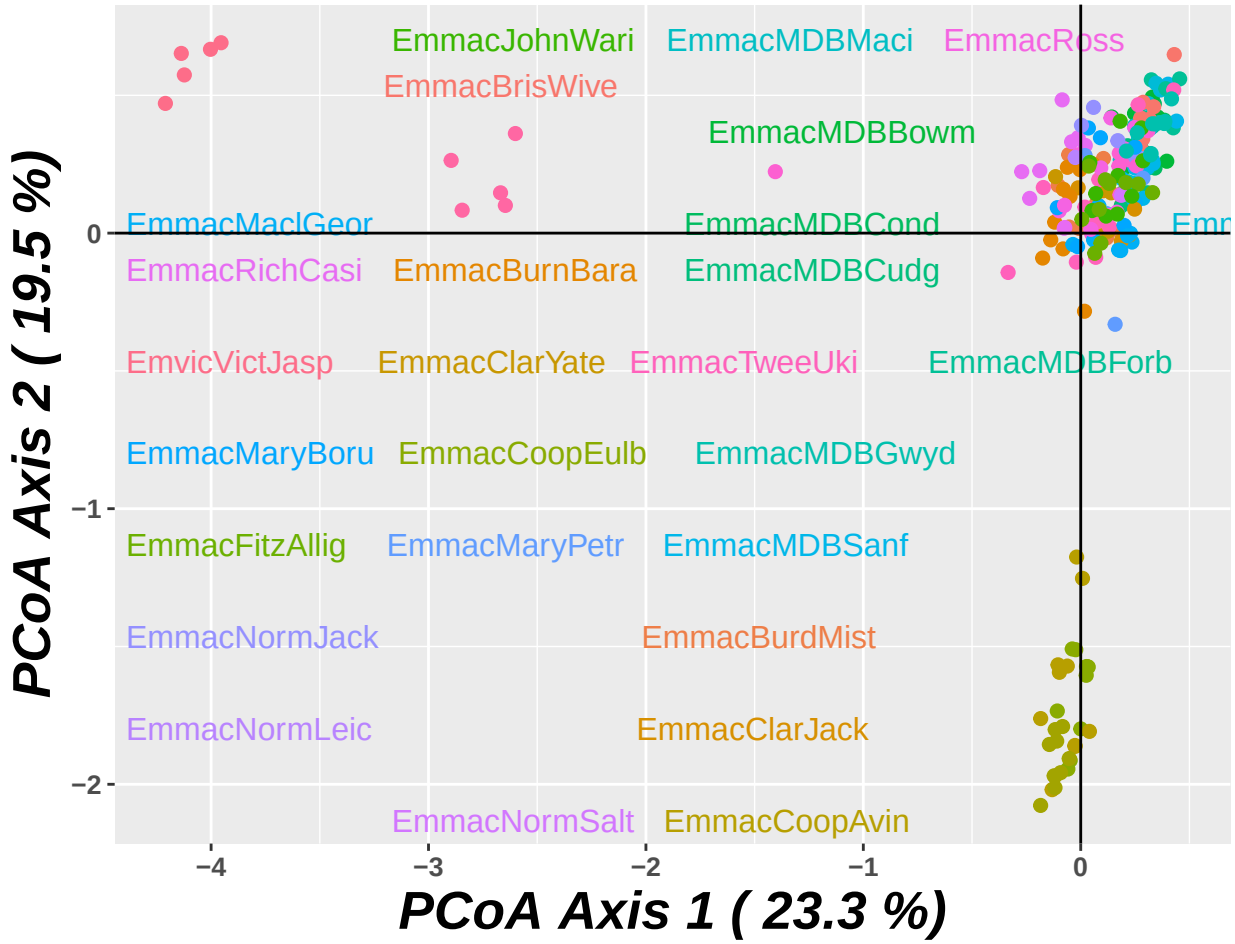
9.2 Plotting the results of PCoA

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)
```

```
## Plotting populations
```



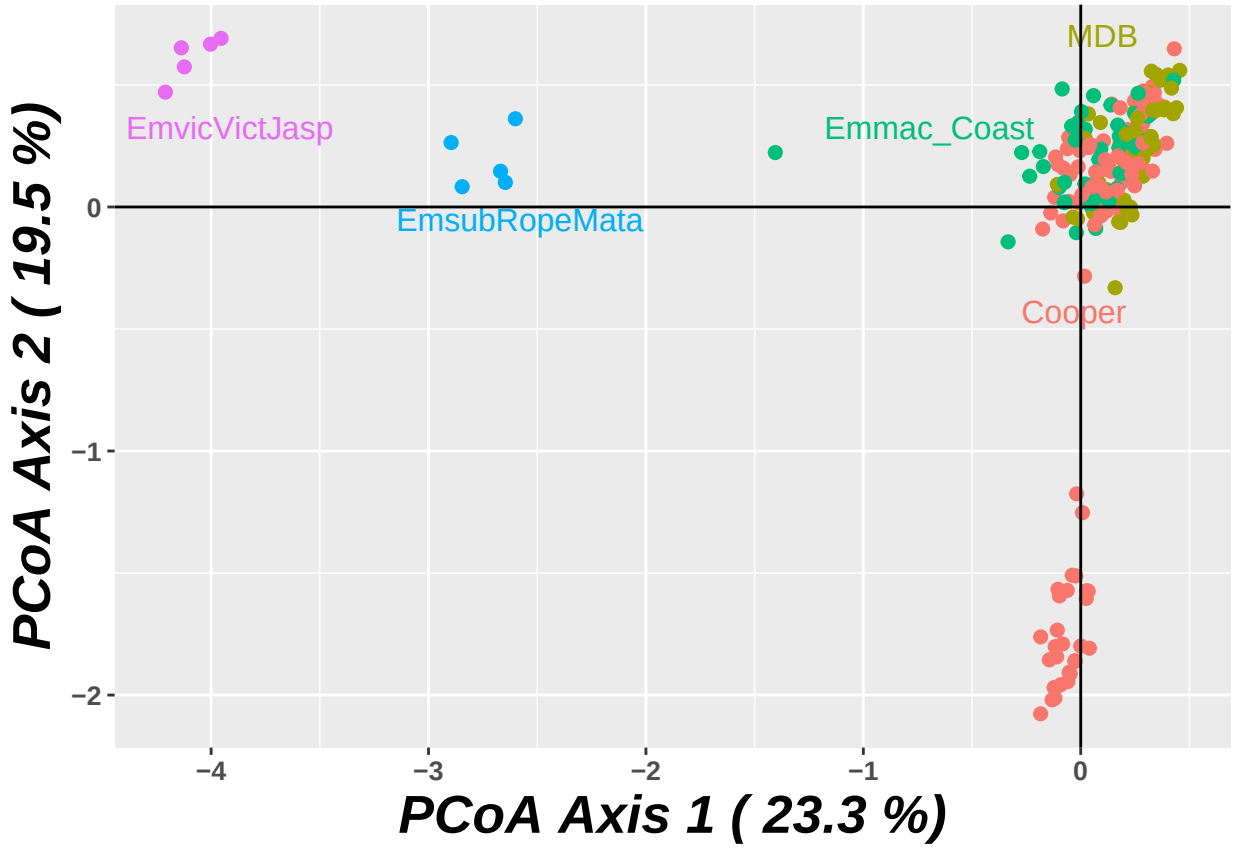
You can see that this plot is very busy, and that the many labels are displaced quite some distance from their associated points. This is because there is a tradeoff between avoiding overlap of the labels and proximity of the labels – you can use colour to identify which labels go with which points.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

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

or using R

```
glnew <- gl
levels(pop(glnew)) <- c(rep("Cooper",13), rep("MDB", 8 ), rep("Emmac_Coast",7),"EmsubRopeMata" ,  "Emvi
gl.pcoa.plot(pc, glnew, labels="pop", xaxis=1, yaxis=2)
```

```
## Plotting populations
```



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

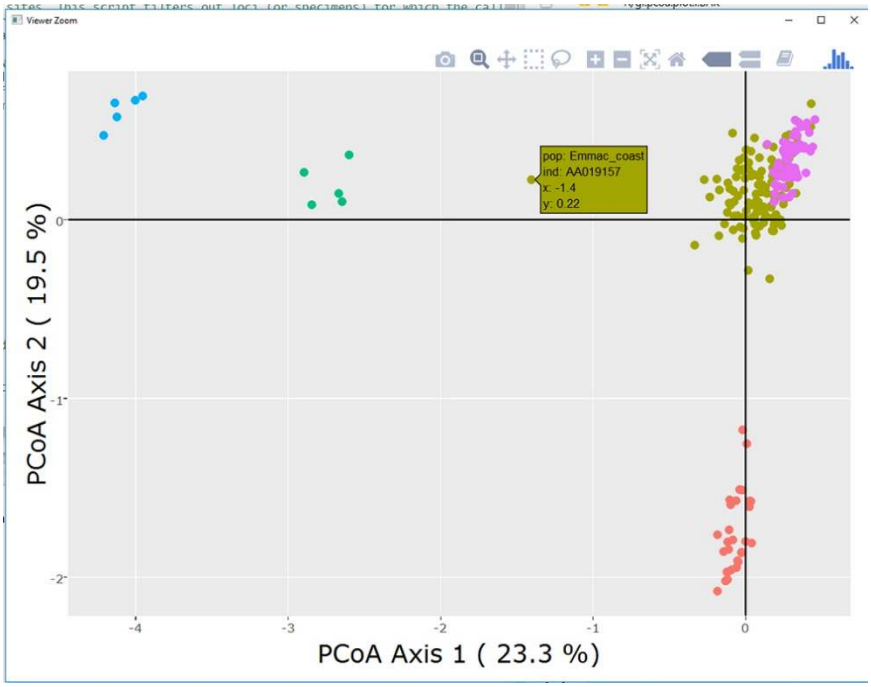
There are other options for gl.pcoa.plot() that allow the axes to be scaled on the basis of proportion of variation explained, to select other combinations of axes to plot, and for adding confidence ellipses. Use the R help facility to explore these additional options.

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

The commands

```
gl.pcoa.plot(pc, gl.new, 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.



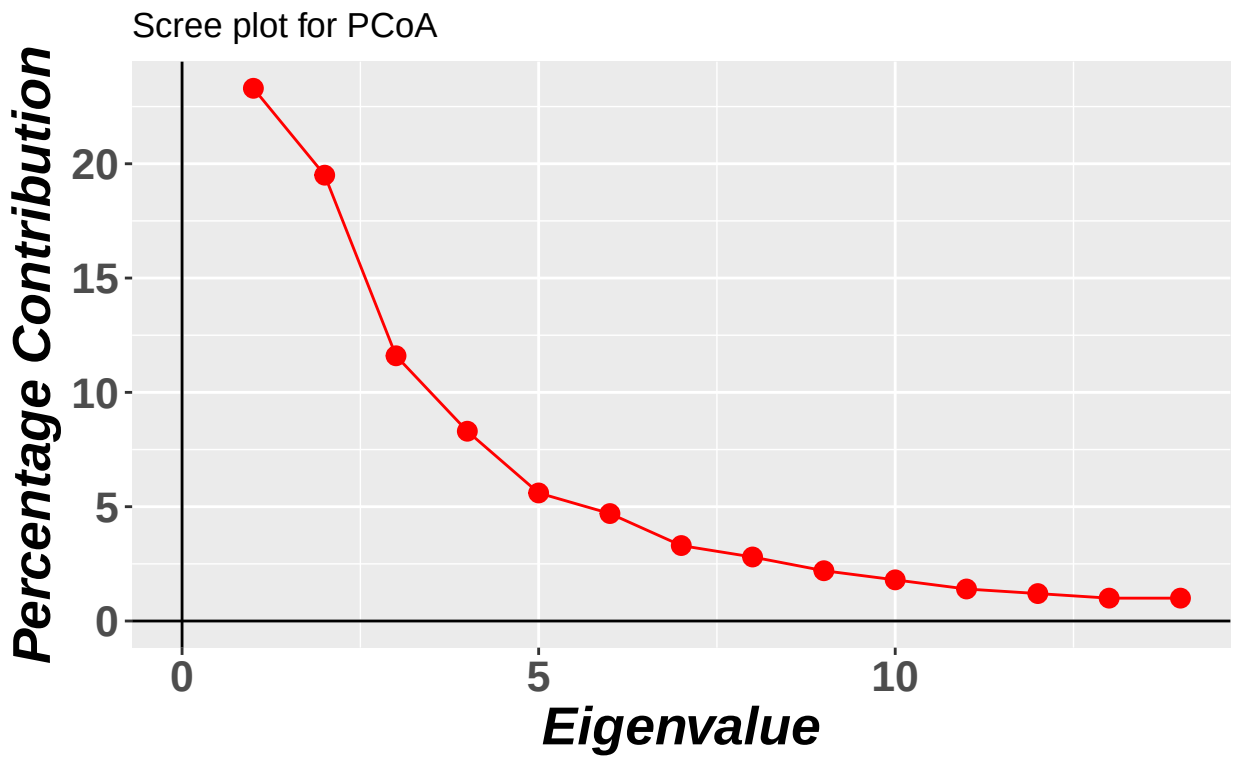
Now moving the mouse over the point reveals its identity. The animal is AA19157, from the coastal populations, and further scrutiny reveals it is from the Barron River in northern Queensland. Seems there has been some allelic exchange there.

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 a
## No. of axes each explaining 10% or more of total variation: 3
```

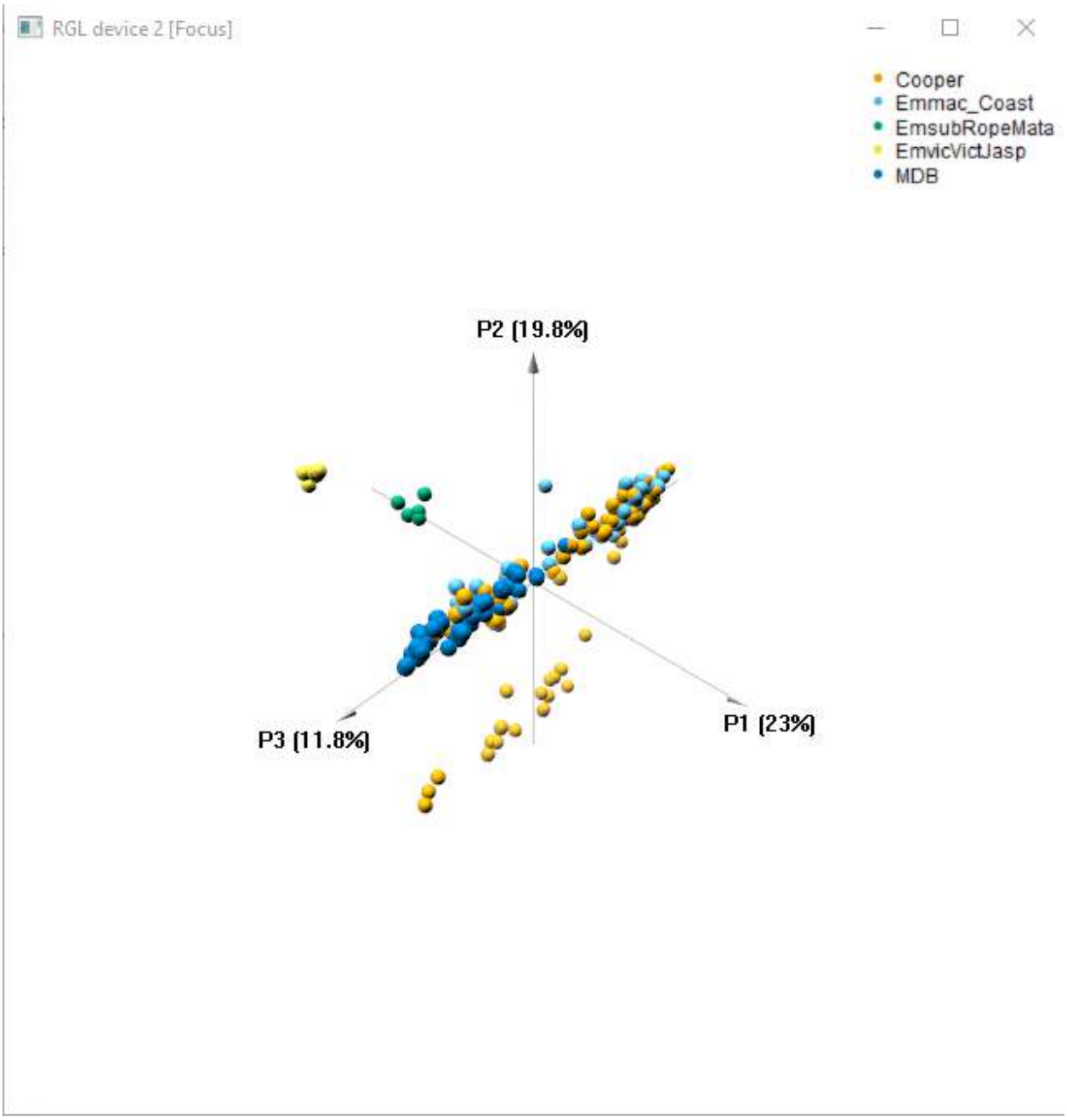


This plot, by default, will show the percentage variation in the data explained by each axis successively where the amount of variation is substantive. By substantive, I mean explaining more than the original variables did on average. As a rule of thumb, one should examine all dimensions that explain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

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.

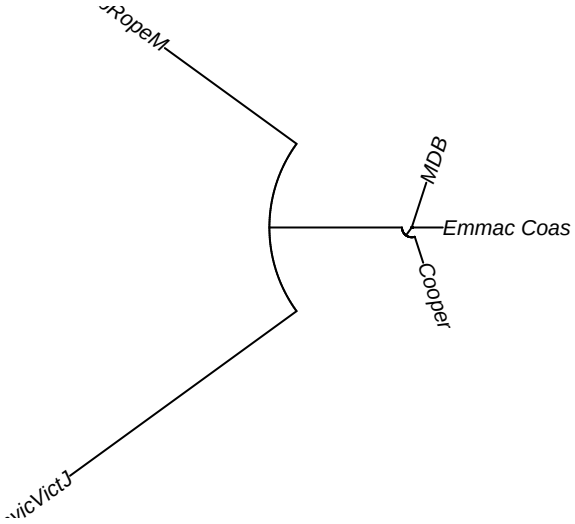
This function is essentially a wrapper for the corresponding function in {pca3d}, adding percentage variation explained to each axis and fixing some parameters.

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")
```

```
## Converting to a matrix of frequencies, locus by populations
## Computing Euclidean distances
```




```
##
## Phylogenetic tree with 5 tips and 3 internal nodes.
##
## Tip labels:
## [1] "Cooper    " "MDB       " "Emmac_Coas" "EmsubRopeM" "EmvicVictJ"
##
## 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, that is, 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. Gene frequencies may ebb and wane, but once a locus becomes fixed for an allele or suite of alleles, there is no returning. 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. The script

```
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.

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

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.

One challenge to this approach stems from low sample sizes. The probability of all alleles in one such population having one allelic state, say the homozygous reference allele 0/0 (2-row format), and all the individuals in the other population having the alternate allelic state, homozygous 1/1, by chance alone depends on the allelic frequency in the parent populations and on the sample size (2n for diploids).


Hint

The consequence of low sample sizes is to generate spurious fixed differences, that is, fixed differences arising via sampling error. The probability of such errors compounds as more and more SNP loci are examined

When the number of loci examined gets up into the 10s of thousands and the sample sizes are small (say < 5 individuals per population), the probability of a spurious fixed difference between two populations rises to unacceptable levels. To manage this, it is wise to ensure that all populations are sampled with 10 or more individuals (2n=20), whereby the probability of generating spurious fixed differences is vanishingly small. The compounded probability of one spurious fixed difference in 100,000 loci is complicated to calculate without detailed knowledge of parental allele frequencies, but remains manageably small for n=10.

11 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.

11.1 Distance Methods

Under ideal conditions, the true phylogeny can be uniquely recovered from the true measures of divergence between OTUs. 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.

11.2 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.

11.3 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 use the command:

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

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)
```

```
## Key variable TrimmedSequence found.
##
## Average trimmed sequence length: 51 ( 20 to 69 )
## Total number of trimmed sequences: 255
## Base frequencies (%)
##   A: 27.83
##   G: 24.71
##   T: 26.18
##   C: 21.29
##
## Transitions  : 52.55
## Transversions: 47.45
## tv/ts ratio: 1.1074
```

```
##      col1 col2
## [1,] "A"  "27.83"
## [2,] "G"  "24.71"
## [3,] "T"  "26.18"
## [4,] "C"  "21.29"
## [5,] "tv" "47.45"
## [6,] "ts" "52.55"
```

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")
```

11.4 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 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)
```

```
## Key variable TrimmedSequence found.
##
## Average trimmed sequence length: 51 ( 20 to 69 )
## Total number of trimmed sequences: 255
## Base frequencies (%)
##   A: 27.83
##   G: 24.71
##   T: 26.18
##   C: 21.29
##
## Transitions  : 52.55
## Transversions: 47.45
## tv/ts ratio: 1.1074
```

```
##      col1 col2
## [1,] "A"  "27.83"
## [2,] "G"  "24.71"
## [3,] "T"  "26.18"
## [4,] "C"  "21.29"
## [5,] "tv" "47.45"
## [6,] "ts" "52.55"
```

12 Genlight Conversion

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

```
gi <- gl2gi(gl, probar=FALSE)
```

```
## Start conversion....
## Please note conversion of bigger data sets will take some time!
## Once finished, we recommend to save the object using >save(object, file="object.rdata")
##
## Matrix converted.. Prepare genind object...
## Finished! Took 1 seconds.
```

Conversion in the opposite direction can be achieved with

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

12.1 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}.

12.2 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 (F1's, F2's, 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 generate an input file for NewHybrids, use the function

```
glnew <- gl2nhyb(gl, outfile = file.path(tempdir(),"nhyb.txt"))
```

```
## Extracting the SNP data
##   No parental population specified
##     Selecting ca 200 random loci
## Subsampling at random, approximately 200 loci from genlight object
##    No. of loci retained = 128
##    Note: SNP metadata discarded
## Converting data to NewHybrids format
##   Adding sequential number
## Writing the NewHybrids input file C:\Users\s425824\AppData\Local\Temp\RtmpQRwb3B/nhyb.txt
##   NumIndivs  250
##   NumLoci  128
##   Digits 1
##   Format Lumped
```

with appropriate options set, for example:

```
gl.new <- gl2nhyb(gl, outfile = "nhyb.txt", p0 = NULL,p1 = NULL, t = 0,   m = "random")
```

Refer to help 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.

12.3 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.

13 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 (IBD) 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

13.1 Faststructure

STRUCTURE is one of the most widely used population analysis tools that allows researchers to assess patterns of genetic structure in a set of samples (Porras-Hurtado et al., 2013). STRUCTURE is freely available

software for population analysis developed (Pritchard et al., 2000). STRUCTURE analyses differences in the distribution of genetic variants amongst populations and places individuals into groups where they share similar patterns of variation. STRUCTURE both identifies populations from the data and assigns individuals to those populations. fastSTRUCTURE is an improved implementation to analyse large quantities of data (Raj et al., 2014).

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

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

```
## Saved faststructure file: C:/Users/s425824/AppData/Local/Temp/RtmpMbk9lR/Rbuild2e348d32837/dartR/vig
## Consists of 250 individuals and  255 loci.
```

```
## NULL
```

14 References

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

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.

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

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.

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.

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.