dartR Workshop - GSA

Bernd Gruber & Arthur Georges

2018-08-28

Foreword

What it the workshop about?

In this workshop we provide an overview on the use of a recently developed R-package (dartR) that aims to integrate as many as possible ways to analyse SNP data sets. We will cover the following topics:

Overview (indicated times in brackets)

0. Learning R on (the quickest intro possible) (9:00-10:00) - Bernd

   - Rstudio
   - packages
   - Objects
   - vectors
   - matrices
   - plotting
   - indexing

1. Preparing data sets (10:00 - 10:30) - Bernd

   - loading data sets into R (DArT format)
   - how to load other formats

2. The idea of the a genlight object (10:45 - 11:00) - Bernd/Arthur

   - explore a genlight object
   - quality filter your data
   - subset/recode your data set

3. Population structure (11:00 - 11:45) - Arthur

   - PCoA
   - phylogenetic trees
   - fixed differences

4. Population assignment (11:45 - 12:15) - Arthur

5. Landscape genetics (12:45 - 13:15) - Bernd

   - isolation by distance
   - landscape resistance

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

- how to send a genlight objectto a friend
- FASTA
- STRUCTURE, fastSTRUCTURE, NewHybrid

The format of the workshop will be a mixture between short lectures on the different topics to explain the idea of an approach and the type of analysis followed by computer exercises how to implement and interpret such an analysis. To avoid setting up computers we will use our Rstudio server, with the advantage that everyone will be on the same page. Data sets can be downloaded from here [`http:github/dartRworkshop/data`]

Login details:

There are two logins, the first to log into the University computer, the second to log into Rstudo on the server.

1. Univeristy login: user: ucstaff\ucvisitor14 pw: ucvisitor
2. Rstudio login. [Open a web browser (chrome, firefox)]

- Goto: dungog.win.canberra.edu.au:8787
- user: guestxyz [your guest number]
- pw: guestxyz [your guest number]

You should now see and Rstudio window such as the one below:

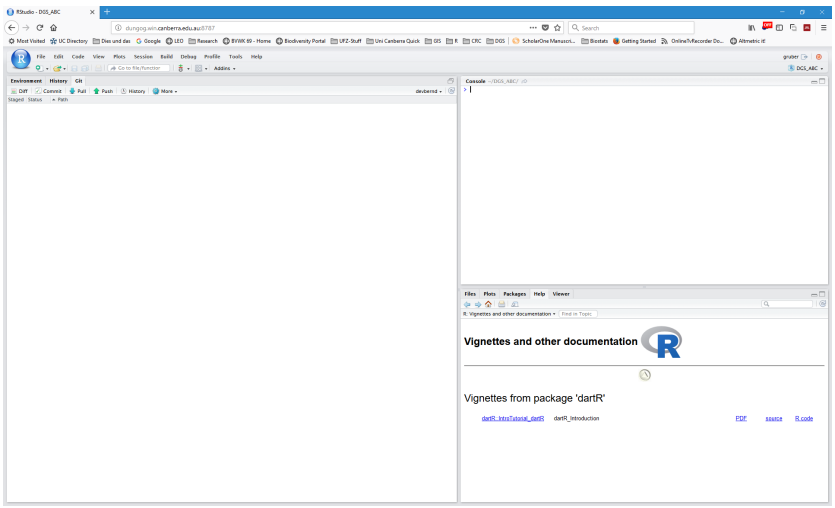


Figure 1: Rstudio

0. Learning R on 1-2 pages (9:00-10:00)

Obviously we cannot teach you R on a page. Rather this is a very quick refresher of a very small selection of R concepts, which we will

need during the workshop. A good starting points are the cheatshets linked from Rstudio (Help->Cheatsheats), but feel free to google for some excellent and free introductions to R.

Rstudio

There are four windows (panes in Rstudio).

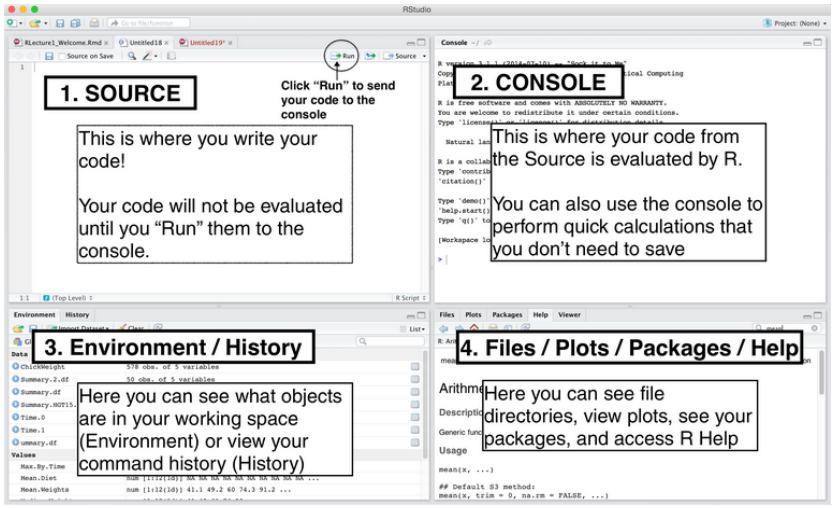


Figure 2: Windows in R studio

Packages

Before you start an analysis in R you need to load additional functions. The package we will use today is called dartR. The first code you should type in your source window (after you created a new file Shift-Control-N) is:

```
library(dartR)
```

```
## Loading required package: adegenet
```

```
## Loading required package: ade4
```

```
##
##    /// adegenet 2.1.1 is loaded ////////////
##
##    > overview: '?adegenet'
##    > tutorials/doc/questions: 'adegenetWeb()'
##    > bug reports/feature requests: adegenetIssues()
```

This loads the dartR package into your session and all functions included in the package can now be used. Be aware once back at home

you need to download and install dartR on your computer once before you can use it.[1]

[1] For information how to install dartR please refer to:`https://github.com/green-striped-gecko/dartR`

Objects and vectors

Data are stored in objects in R. Those objects have names and we can create them:

```
mydata <- c(6, 7, 3, 8)
```

And look into the content using the name and send it to the console:

```
mydata
```

```
## [1] 6 7 3 8
```

There are simple standard accessor functions you can use on every object.

```
str(mydata)
```

```
##  num [1:4] 6 7 3 8
```

```
class(mydata)
```

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

Often there are some additional function that work with certain types of objects. For example mydata is a vector (a one dimensional array/container) that holds numeric data. For this kind of data there are a lot of functions, such as[2]:

[2] When you learn a new package you often do not know the name of functions. Here a good start is to look at so-called tutorials for packages. For example dartR has a vignette that lists (almost) all available functions: Type: `browseVignettes("dartR")` into the console

```
mean(mydata)
```

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

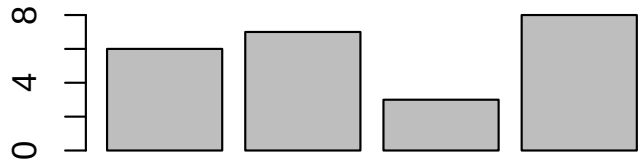
```
summary(mydata)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##    3.00    5.25    6.50    6.00    7.25
##    Max.
##    8.00
```

```
length(mydata)
```

```
## [1] 4
```

```
barplot(mydata)
```



matrices/data.frame

A matrix is basically the R equivalent of a table in Excel and it can store two dimnensional data sets. En example for this kind of data is the built-in data set on iris flowers (Fisher/Anderson 1936).

```
iris
```

Typing the name shows you the content (a long table, not shown here). There are several ways to summarise the data set using built-in functions for data.frames.

```
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"
## [3] "Petal.Length" "Petal.Width"
## [5] "Species"
```

```
nrow(iris)
```

```
## [1] 150
```

```
ncol(iris)
```

```
## [1] 5
```

```
dim(iris)
```

```
## [1] 150   5
```

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width
##  Min.   :4.300   Min.   :2.000
##  1st Qu.:5.100   1st Qu.:2.800
##  Median :5.800   Median :3.000
##  Mean   :5.843   Mean   :3.057
##  3rd Qu.:6.400   3rd Qu.:3.300
##  Max.   :7.900   Max.   :4.400
##   Petal.Length    Petal.Width
##  Min.   :1.000   Min.   :0.100
##  1st Qu.:1.600   1st Qu.:0.300
##  Median :4.350   Median :1.300
##  Mean   :3.758   Mean   :1.199
##  3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

indexing (finding subsets)

Being able to create subsets in R opens the avenue to a very powerful way to analyse your data. E.g. being able to run the same analysis for males and females seperately is often very simple in R, once you scripted your analysis. R achieves that via the indexing function "[]". A matrix/data.frame consists of rows and columns and we can use the indexing function to identify certain columns and rows.

```
iris[3, 4]  #returns the value in row 3, column 4
```

```
## [1] 0.2
```

```
iris[1:3, 4]  #returns the value from row 1 to 3 or column 4 (=a vector)
```

```
## [1] 0.2 0.2 0.2
```

```
iris[1:3, 4:5]  #returns values from row 1 to 3 and column 4 and 5 (=a matrix)
```

```
##   Petal.Width Species
## 1         0.2  setosa
## 2         0.2  setosa
## 3         0.2  setosa
```

We can now use certain colums to find subsets of interest. E.g. if you want to calculate the mean Petal.Length for 'setosa' we have to do two steps. First find all the rows that have 'setosa' in the Species column and then use that as an index for the column Petal.Length to calculate the mean.

```
### Step 1a: Finding the Species column
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"
## [3] "Petal.Length" "Petal.Width"
## [5] "Species"
```

```
### Step 1b: create an index
index <- iris$Species == "setosa"

### Step 2: calculate the mean
mean(iris$Petal.Length[index])
```

```
## [1] 1.462
```

The same result in a "shorter way". We can use the $ function to find a column of a matrix or we can use numbers/names to identify columns.

```
mean(iris[iris$Species == "setosa", "Petal.Length"])
```

```
## [1] 1.462
```

```
# only numbers
mean(iris[1:50, 3])
```

```
## [1] 1.462
```

Admittedly it takes some time to get used to it, but once the concept is understood it is very powerful. The good news is that dartR provides some help functions to subset data set, hence at the beginning subsetting using the '[]' function is not necessary.]

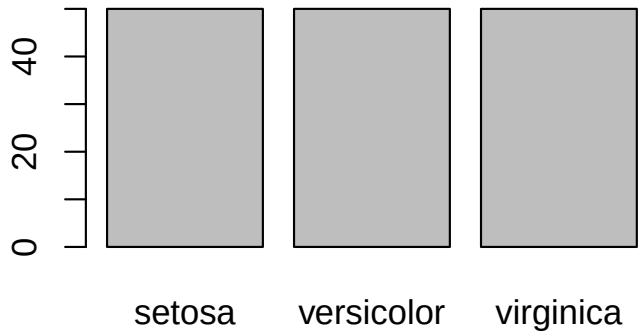
Plotting and summarising

There hundreds of different ways

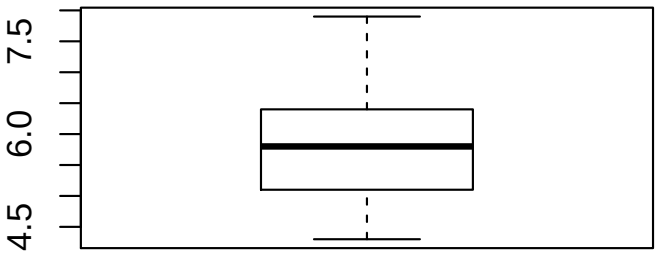
```
table(iris$Species)
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

```
barplot(table(iris$Species))
```




```
boxplot(iris$Sepal.Length)
```



Today we will use a quite specialised type of object which are called genlight objects. And here comes the first task. Access the built-in data set `testset.gl` and find out the following:

Question

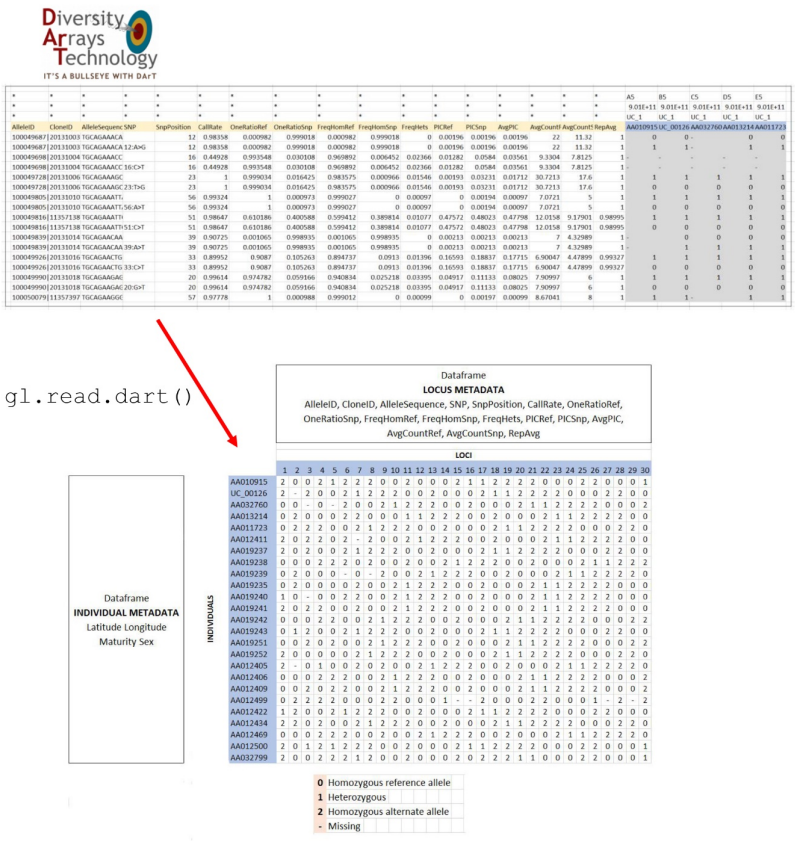
1. What type of data set is `testset.gl`?
2. Can you find the help page for the data set.
3. Type the name and try to understand the "structure" of this data type
4. Try the plot function on this data set.
5. There are additional accessor functions such as `indNames(), nLoc(), nInd(), ploidy()`
6. Try to use table function on pop(testset.gl).

- How many populations are there?
- Can you produce a barplot on the number of individuals per population?

7. **Extra: Can you subset the first four individuals and the first 10 loci and plot them?**

1. Preparing data sets (10:00 - 10:30)

Visualising the process described below:



Loading data sets into R (DArT format)

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.

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 `ind.metafile` 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")
dartfile
```

```
## [1] "C:/Program Files/R/library/dartR/extdata/testset_SNPs_2Row.csv"
```

```
ind.metafile <- system.file("extdata", "testset_metadata.csv",
    package = "dartR")
ind.metafile
```

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

```
gl <- gl.read.dart(filename = dartfile, ind.metafile = ind.metafile,
    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 CloneID AlleleSequence SNP SnpPosition CallRate OneRatioRef OneRatioSnp FreqHomRef FreqHomSn
## Number of rows per Clone. Should be only  2 s: 2
##  Recognised: 250 individuals and 255  SNPs in a 2 row format using C:/Program Files/R/library/dartR/e
## Start conversion....
## Format is 2 rows.
## Please note conversion of bigger data sets will take some time!
## Once finished, we recommend to save the object using save(object, file="object.rdata")
## Try to add covariate file: C:/Program Files/R/library/dartR/extdata/testset_metadata.csv .
## Ids of covariate file (at least a subset of) are matching!
```

```
## Found  250 matching ids out of 250 ids provided in the covariate file. Subsetting snps now!.
##  Added pop factor.
## Added latlon data.
## Added  id  to the other$ind.metrics slot.
##  Added  pop  to the other$ind.metrics slot.
##  Added  lat  to the other$ind.metrics slot.
##  Added  lon  to the other$ind.metrics slot.
##  Added  sex  to the other$ind.metrics slot.
##  Added  maturity  to the other$ind.metrics slot.
```

Hint

It is a good idea to check the example report and the ind.meta file and compare your files with their formats. For example the gl.read.dart function assumes the RepAvg is the header of the last column, before the SNPs column and also that missing values are coded as '-'. If for whatever reason your report from DArT looks different, you need to adjust the arguments in the function accordingly.

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

To display (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: 705.5 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
```

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
## AA010915                2               NA
## UC_00126                2               NA
## AA032760               NA               NA
##          100049728-23-T/G
## AA010915                0
## UC_00126                0
## AA032760                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"
## [3] "EmmacBurnBara" "EmmacClarJack"
## [5] "EmmacClarYate"
```

How to load other formats

Please refer to the vignette if you want to use SNP data from a different source than DArT (which is completely feasible). The general approach is that you need to load your data in such a way that you result in a genlight object. To make use of all functions you then need to add the meta data (for loci and individuals) "manually" to the genlight object. For example if you have coordinates the correct "slot" to add would be:

```
gl@other$latlong <- coordinatesforeachindividual
```

2. The format/structure of a genlight object (10:45 - 11:00)

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
AlleleID:	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:	(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

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

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.

AlleleID 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.

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 ind.metafile. Examples of user-defined metadata for individuals include:

metric	explanation
sex	Sex of the individual (Male, Female), necessary for gl.sexlinkage
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
##  [6] Male    Female  Female  Male    Female
## Levels: Female Male Unknown
```

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.

function	explanation
gl.filter.hwe	Filters departure of Hardy-Weinberg-Equilibrium for every loci per population or overall
gl.report.hwe	Reports departure of Hardy-Weinberg-Equilibrium for every loci per population or overall

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

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

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

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

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

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

```
## Starting gl.filter.callrate: Filtering on Call Rate
##   Removing loci based on Call Rate, threshold = 0.95
## gl.filter.callrate completed
```

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

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

```
## Starting gl.filter.callrate: Filtering on Call Rate
##   Removing individuals based on Call Rate, threshold t = 0.9
##   List of individuals deleted because of low call rate: AA010915 AA032760 AA011723 AA012411 AA019237
## Starting gl.filter.monomorphs: Deleting monomorphic loci
##   Deleting monomorphic loci and loci with all NA scores
## Completed gl.filter.monomorphs
##
## Starting gl.recalc.metrics: Recalculating locus metrics
## Starting utils.recalc.avgpic: Recalculating OneRatioRef, OneRatioSnp, PICRef, PICSnp and AvgPIC
## Completed utils.recalc.avgpic
##
## Starting utils.recalc.callrate: Recalculating CallRate
## Completed utils.recalc.callrate
##
## Starting utils.recalc.freqhets: Recalculating frequency of heterozygotes
## Completed utils.recalc.freqhets
##
## Starting utils.recalc.freqhomref: Recalculating frequency of homozygotes, reference allele
## Completed utils.recalc.freqhomref
##
## Starting utils.recalc.freqhomref: Recalculating frequency of homozygotes, alternate allele
## Completed utils.recalc.freqhomsnp
##
## Note: Locus metrics recalculated
## Completed gl.recalc.metrics
##
## gl.filter.callrate completed
```

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

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

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

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

5. Filter out monomorphic loci

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

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

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

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

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

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

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

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

Subsetting and Recoding Data

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

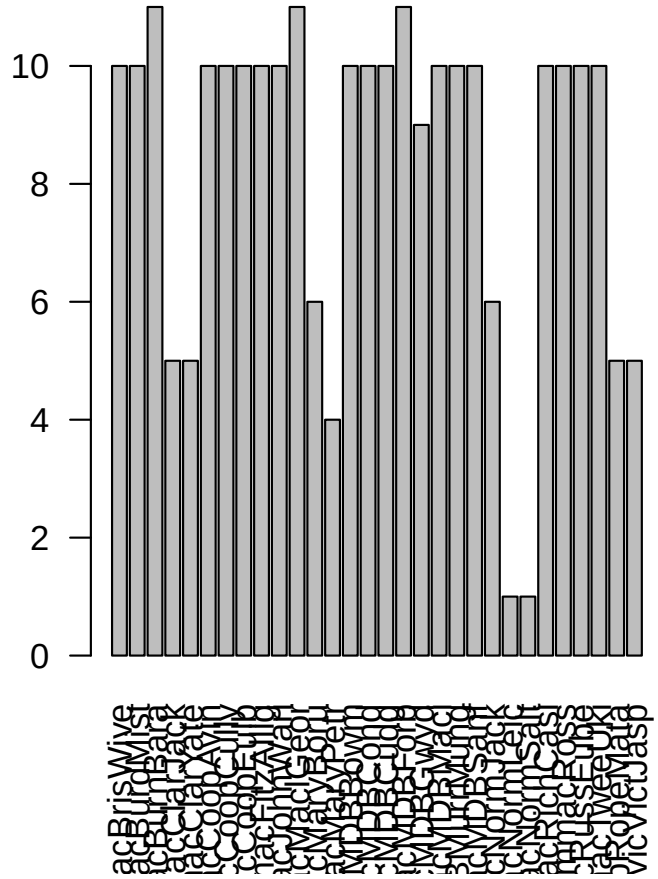
```
# table on individuals per population
table(pop(gl))
```

```
##
##    EmmacBrisWive    EmmacBurdMist
##               10               10
##    EmmacBurnBara    EmmacClarJack
##               11                5
##    EmmacClarYate    EmmacCoopAvin
##                5               10
##   EmmacCoopCully    EmmacCoopEulb
##               10               10
##   EmmacFitzAllig    EmmacJohnWari
```

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

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

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



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

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

Try these out for yourself.

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

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

will delete all individuals in all populations except those listed.

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

will delete all individuals in the listed populations.

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

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

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

will rename population EmvicVictJasp to Emydura_victoriae.

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

Individual reassignment

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

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

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

```
##   [1] "AA010915"  "UC_00126"  "AA032760"
##   [4] "AA013214"  "AA011723"  "AA012411"
##   [7] "AA019237"  "AA019238"  "AA019239"
##  [10] "AA019235"  "AA019240"  "AA019241"
##  [13] "AA019242"  "AA019243"  "AA019251"
##  [16] "AA019252"  "AA012405"  "AA012406"
##  [19] "AA012409"  "AA012499"  "AA012422"
##  [22] "AA012434"  "AA012469"  "AA012500"
##  [25] "AA032799"  "AA032826"  "AA010795"
##  [28] "AA010796"  "AA032800"  "AA032801"
##  [31] "AA032808"  "AA032809"  "AA032811"
##  [34] "AA032812"  "AA032822"  "AA032825"
##  [37] "AA010797"  "AA010752"  "AA010754"
##  [40] "AA010756"  "AA010798"  "AA010799"
##  [43] "AA010800"  "AA010802"  "AA010803"
##  [46] "AA010804"  "AA010809"  "AA010749"
##  [49] "AA010758"  "AA010763"  "AA010765"
```

```
##  [52] "AA010771"  "AA010772"  "AA010781"
##  [55] "AA032762"  "AA032763"  "AA032756"
##  [58] "AA032757"  "AA032758"  "AA032761"
##  [61] "AA032765"  "AA010931"  "AA010937"
##  [64] "AA010940"  "AA032764"  "AA032768"
##  [67] "AA010936"  "AA010909"  "AA010916"
##  [70] "AA010917"  "AA010920"  "AA010921"
##  [73] "AA020651"  "AA020652"  "AA020667"
##  [76] "AA020669"  "AA020655"  "AA020656"
##  [79] "AA020644"  "AA020645"  "AA020646"
##  [82] "AA020649"  "AA013203"  "AA013217"
##  [85] "AA013220"  "AA013202"  "AA013225"
##  [88] "AA018496"  "AA018497"  "AA018513"
##  [91] "AA013231"  "AA013261"  "AA013265"
##  [94] "AA013270"  "AA018492"  "AA018493"
##  [97] "AA018494"  "AA018495"  "AA018514"
## [100] "AA018515"  "AA018516"  "UC_00125"
## [103] "UC_00126a" "UC_00146"  "UC_00149"
## [106] "AA018640"  "AA018658"  "AA011729"
## [109] "UC_00132"  "UC_00137"  "UC_00143"
## [112] "UC_00157"  "UC_00161"  "AA018637"
## [115] "AA018638"  "AA018639"  "AA011731"
## [118] "AA033576"  "AA033577"  "AA011732"
## [121] "AA011737"  "AA011741"  "AA011744"
## [124] "AA011745"  "AA011746"  "AA011749"
## [127] "AA033575"  "AA033578"  "AA012411a"
## [130] "AA033579"  "AA033582"  "AA033593"
## [133] "AA033602"  "AA033609"  "AA033617"
## [136] "AA010915a" "AA011723a" "AA019158"
## [139] "AA020379"  "UC_01044"  "AA018380"
## [142] "AA018371"  "AA004553"  "AA000328"
## [145] "AA000311"  "AA019159"  "AA020378"
## [148] "UC_01060"  "AA018379"  "AA018365"
## [151] "AA004554"  "AA000303"  "AA000320"
## [154] "AA019160"  "AA020377"  "UC_01053"
## [157] "AA018375"  "AA004555"  "AA000304"
## [160] "AA019165"  "AA019161"  "AA020376"
## [163] "UC_01062"  "AA018374"  "AA04523"
## [166] "AA000305"  "AA019164"  "AA020375"
## [169] "AA018373"  "AA032875"  "AA000309"
## [172] "AA019163"  "AA020374"  "AA018368"
## [175] "AA032878"  "AA000302"  "AA019162"
## [178] "AA020365"  "UC_00150"  "AA018369"
## [181] "AA004551"  "AA032880"  "AA000307"
```

```
## [184] "AA019156"  "AA020371"  "UC_01051"
## [187] "AA018370"  "AA004552"  "AA032882"
## [190] "AA000310"  "AA019157"  "AA019075"
## [193] "AA004864"  "AA019071"  "AA004868"
## [196] "AA019083"  "AA019072"  "AA004858"
## [199] "AA004869"  "AA019082"  "AA019073"
## [202] "AA004859"  "AA004866"  "AA019077"
## [205] "AA004860"  "AA019080"  "AA004861"
## [208] "AA019079"  "AA004862"  "AA019078"
## [211] "AA004863"  "UC_00267"  "UC_00205"
## [214] "UC_00206"  "UC_00208"  "UC_00243"
## [217] "UC_00209"  "UC_00254"  "UC_00210"
## [220] "UC_00259"  "UC_00126c" "AA063718"
## [223] "AA063720"  "AA063722"  "AA063726"
## [226] "AA063732"  "AA063708"  "AA063710"
## [229] "AA063712"  "AA063714"  "AA063716"
## [232] "AA020735"  "AA032442"  "AA032441"
## [235] "AA020749"  "AA020746"  "AA020744"
## [238] "AA020743"  "AA020739"  "AA020738"
## [241] "AA001451"  "AA01452"   "AA001454"
## [244] "AA001455"  "AA001446"  "AA001456"
## [247] "AA001447"  "AA001448"  "AA001449"
## [250] "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.

Recode tables

Alternatively, reassignment and deletion of populations can be effected using a recode table, that is, a table stored as a csv file containing the old population assignments and the new population assignments

as two columns. The quickest way to construct a recode table for an active genlight object is using

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

Hint

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

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

The population reassignments are then applied using:

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

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

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

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


Task

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

Another way of population reassignment is to use:

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

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

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

Deleting populations with a recode table

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

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


Task

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

Relabeling and deleting individuals with a recode table

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

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

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

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

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

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

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

The population reassignments are then applied using

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

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

Another way of individual reassignment is to use

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

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

Deleting individuals

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

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

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

Recalculating locus metadata

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

gl <- gl.recalc.metrics(gl)

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

Using R commands to manipulate the genlight object

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

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

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

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

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

For example:

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

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

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

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

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

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

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

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

you see that the subsetting of the meta data for individuals worked fine (we have seven 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
##   EmmacMaclGeor       11    0
##   EmmacMaryBoru        6    0
##   EmmacMaryPetr        4    0
##   EmmacMDBBowm        10    0
##   EmmacMDBCond        10    0
##   EmmacMDBCudg        10    0
##   EmmacMDBForb        11    0
##   EmmacMDBGwyd         9    0
##   EmmacMDBMaci        10    0
##   EmmacMDBMurrMung    10    0
##   EmmacMDBSanf        10    0
##   EmmacNormJack        6    0
##   EmmacNormLeic        1    0
##   EmmacNormSalt        1    0
##   EmmacRichCasi       10    0
##   EmmacRoss           10    0
##   EmmacRussEube        0   10
##   EmmacTweeUki        10    0
##   EmsubRopeMata        5    0
##   EmvicVictJasp        0    5
```

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

```
##  [1] 242 138  67 243 254  28 218 115  16   9
## [11]  34 219  27  21 222  80 109  66 230 157
## [21]  65 231  46 234 200 158 169 118  48  59
```

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: 186.1 Kb
##  30 (6.67 %) missing data
##
##  // Basic content
##    @gen: list of 15 SNPbin
##    @ploidy: ploidy of each individual  (range: 2-2)
##
##  // Optional content
##    @ind.names:  15 individual labels
##    @loc.names:  30 locus labels
##    @loc.all:  30 alleles
##    @position: integer storing positions of the SNPs
##    @pop: population of each individual (group size range: 5-10)
##    @other: a list containing: loc.metrics  latlong  ind.metrics
```

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

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

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

```
## [1] 30 19
```

For those not fully versed in R, there are the above {dartR} filters to achieve the same end and the advantage is that the filters do handle

subsets of data correctly without any additional need to subset the meta data. The advantage of the R approach is that it is much more useful in case you want to script your analysis without intervention of a user when recoding your data set.

3. Population structure (11:00 - 11:45) [Arthur]

Visualisation

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

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

PCoA in `dartR`

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

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

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

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

```
names(pc)
```

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

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

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

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

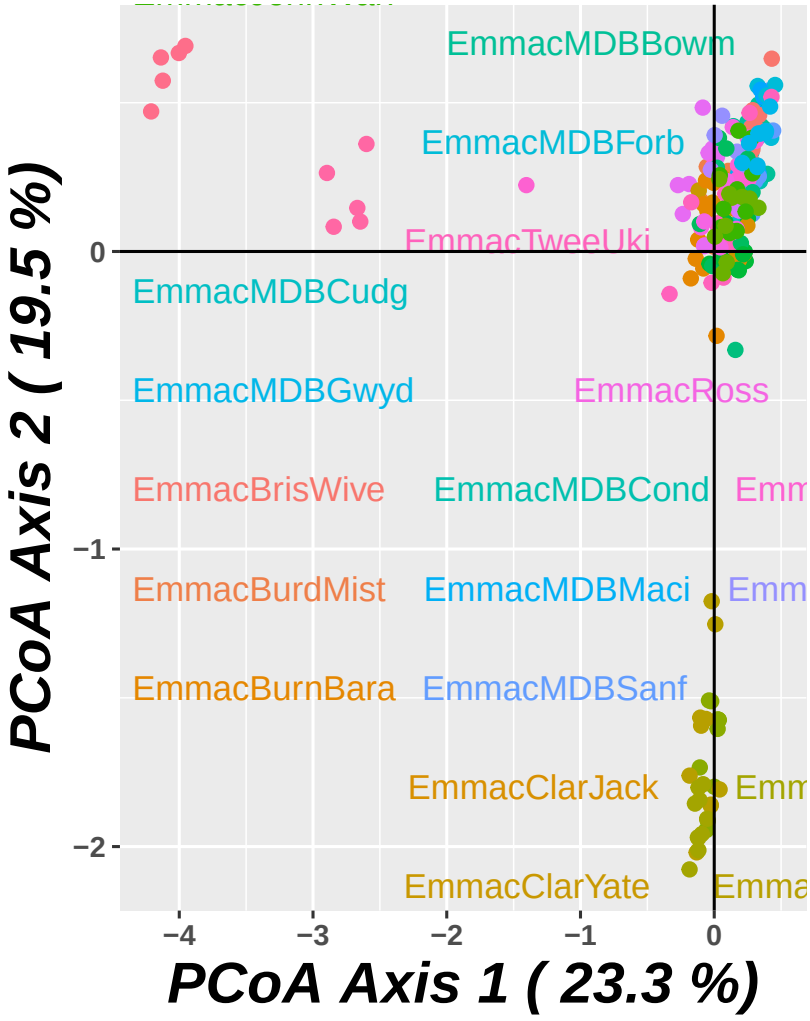
Plotting the results of PCoA

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

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

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

```
## Plotting populations
```



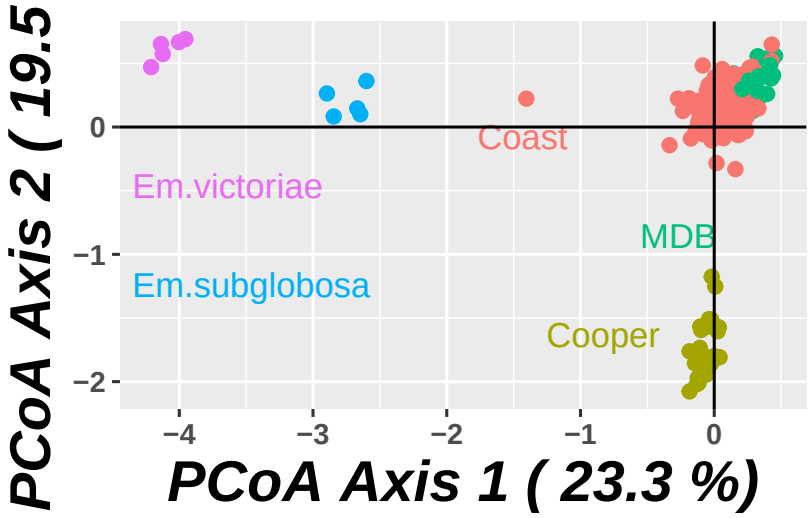
You can see that this plot is very busy, and that the many labels are displaced quite some distance from their associated points. This is because there is a tradeoff between avoiding overlap of the labels and proximity of the labels - you can use colour to identify which labels go with which points.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

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

or using R

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

```
## Plotting populations
```



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

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

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

In case ggplotly is not installed, please type:

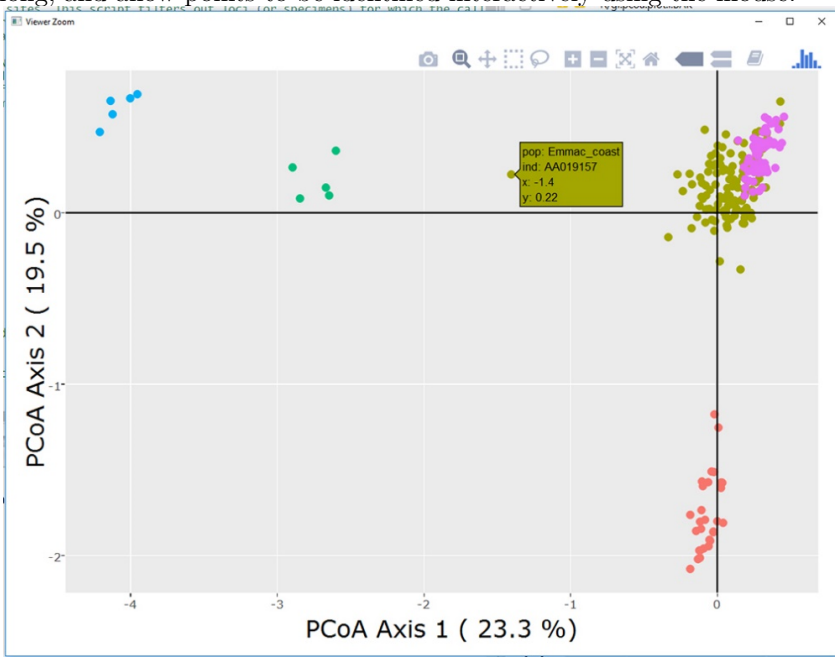
```
install.packages("devtools")
library(devtools)
install_github("hadley/ggplot2")
library(ggplot2)
```

The commands

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

will plot the individuals in the top two dimensions of the ordinated

space, colour the points in accordance to the population to which they belong, and allow points to be identified interactively using the mouse.



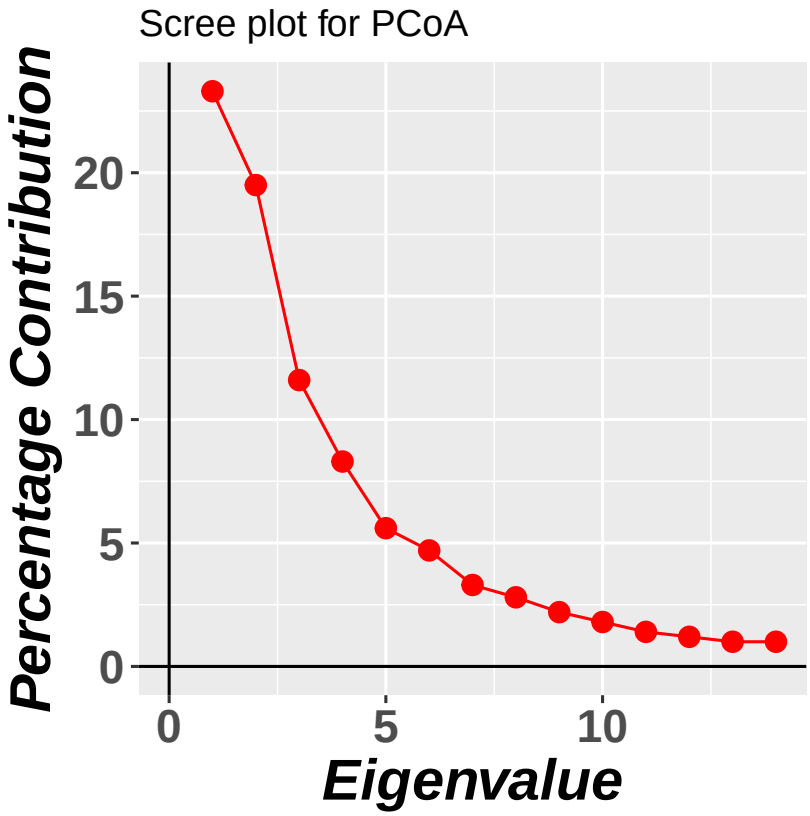
Now moving the mouse over the point reveals its identity. The animal is AA19157, from the coastal populations, and further scrutiny reveals it is from the Barron River in northern Queensland. Seems there has been some allelic exchange there.

The Scree Plot

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

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

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

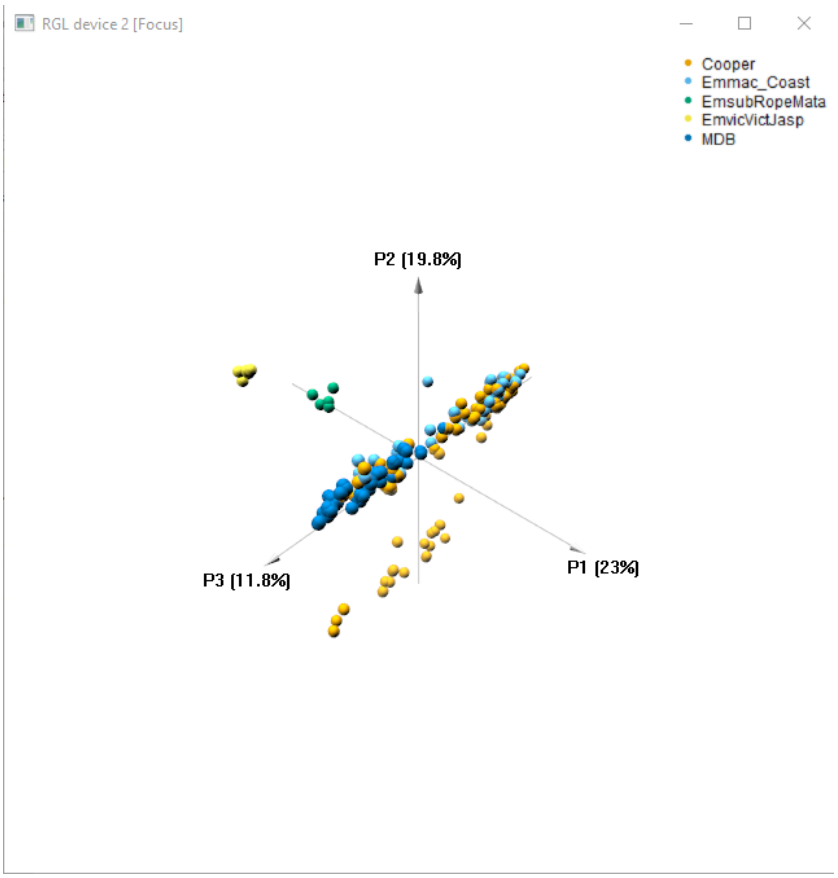


This plot, by default, will show the percentage variation in the data explained by each axis successively where the amount of variation is substantive. By substantive, I mean explaining more than the original variables did on average. As a rule of thumb, one should examine all dimensions that explain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

3D Plot

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

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



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

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

```
- phylogenetic analysis

    - concatonation sequence tags
    - distance meassures
    - svdquartetts
    - treemix
    - fixed differences
```

4. Population assignment (11:45 - 12:15) [Arthur]

5. Landscape genetics (12:45 - 13:15) [Bernd]

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

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

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

Task

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

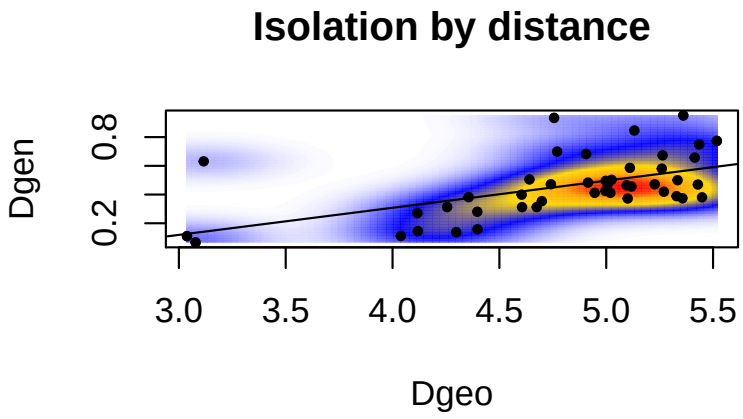
Isolation by distance

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

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

```
## Standard analysis performed on the genlight object. Mantel test and plot will be Fst/1-Fst versus log
## Coordinates not transformed. Distances calculated on the provided coordinates.
## Mantel statistic based on Pearson's product-moment correlation
##
## Call:
## mantel(xdis = Dgen, ydis = Dgeo, permutations = 999, na.rm = TRUE)
##
## Mantel statistic r: 0.5513
##       Significance: 0.001
##
```

```
## Upper quantiles of permutations (null model):
##   90%   95% 97.5%   99%
## 0.234 0.295 0.355 0.414
## Permutation: free
## Number of permutations: 999
```



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

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

Landscape genetics using a landsacpe restistance approach

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

Calculation of cost distances

In this example we use populations as the entity of interest. Hence we need to calculate three distance matrices, namely a Euclidean distance matrix, a cost distance matrix and finally a genetic distance matrix. The two distance matrices can then be used (very similar to the partial mantel test above) to compete against each other how well

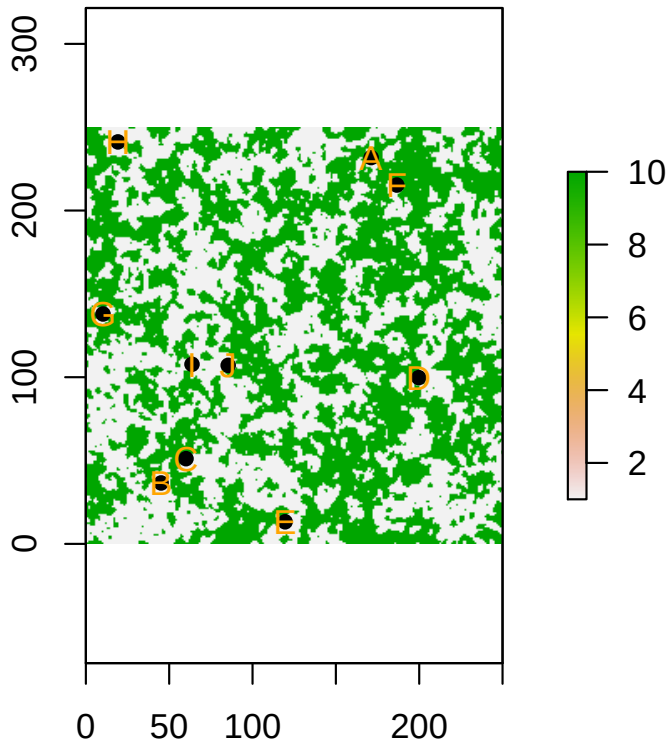
they explain genetic distances. As mentioned we base our analysis on indviduals, therefore we first need to calculate the coordinates of our population centers. But first we load our (resistance) landscape.

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

We calculate the population centers via:

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

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



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

Euclidean distance

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

Costdistances

```
library(PopGenReport)
```

```
## Loading required package: raster
```

```
## Loading required package: sp
```

```
##
## Attaching package: 'raster'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
## The following object is masked from 'package:ggplot2':
##
##     calc
```

```
##
## Attaching package: 'PopGenReport'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     possums
```

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

genetic distance

For simplicity we will use pairwise Fsts between population here

```
library(StAMPP)
```

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

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

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

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

##
## Attaching package: 'pegas'

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

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

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

And finally run a partial mantel test

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

```
## $mantel.tab
##                   model       r     p
## 1 Gen ~cost | Euclidean  0.4945  0.03
## 2 Gen ~Euclidean | cost -0.2502 0.819
```

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

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

```
## Start conversion....
## Please note conversion of bigger data sets will take some time!
## Once finished, we recommend to save the object using >save(object, file="object.rdata")
##
  |
  |                                     |   0%
  |
  |                                     |   1%
  |
```

```
|=                                                                    |   2%
|
|=                                                                    |   3%
|
|=                                                                    |   4%
|
|==                                                                   |   4%
|
|==                                                                   |   5%
|
|==                                                                   |   6%
|
|==                                                                   |   7%
|
|===                                                                  |   7%
|
|===                                                                  |   8%
|
|===                                                                  |   9%
|
|===                                                                  |  10%
|
|====                                                                 |  10%
|
|====                                                                 |  11%
|
|====                                                                 |  12%
|
|====                                                                 |  13%
|
|=====                                                                |  13%
|
|=====                                                                |  14%
|
|=====                                                                |  15%
|
|=====                                                                |  16%
|
|======                                                               |  16%
|
|======                                                               |  17%
|
|======                                                               |  18%
|
```

```
|=======                               |  19%
|
|=======                               |  20%
|
|=======                               |  21%
|
|========                              |  22%
|
|========                              |  23%
|
|========                              |  24%
|
|=========                             |  24%
|
|=========                             |  25%
|
|=========                             |  26%
|
|=========                             |  27%
|
|==========                            |  27%
|
|==========                            |  28%
|
|==========                            |  29%
|
|==========                            |  30%
|
|===========                           |  30%
|
|===========                           |  31%
|
|===========                           |  32%
|
|===========                           |  33%
|
|============                          |  33%
|
|============                          |  34%
|
|============                          |  35%
|
|============                          |  36%
|
```

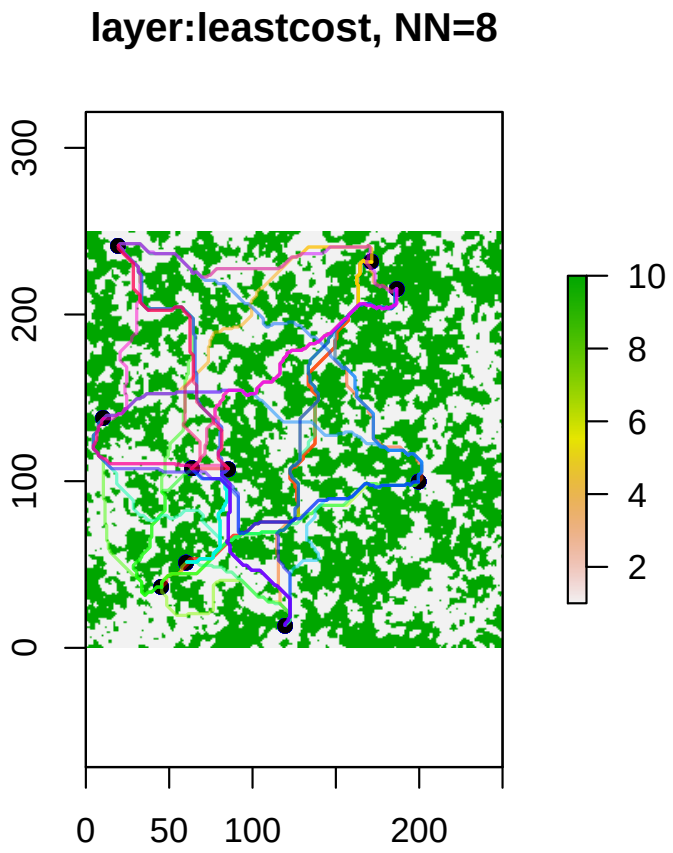
```
|============                          |  36%
|
|============                          |  37%
|
|============                          |  38%
|
|=============                         |  39%
|
|=============                         |  40%
|
|=============                         |  41%
|
|==============                        |  42%
|
|==============                        |  43%
|
|==============                        |  44%
|
|===============                       |  44%
|
|===============                       |  45%
|
|================                      |  46%
|
|================                      |  47%
|
|================                      |  47%
|
|=================                     |  48%
|
|=================                     |  49%
|
|=================                     |  50%
|
|==================                    |  50%
|
|==================                    |  51%
|
|==================                    |  52%
|
|==================                    |  53%
|
|===================                   |  53%
|
```

```
|==================                    |  54%
|
|==================                    |  55%
|
|==================                    |  56%
|
|==================                    |  56%
|
|===================                   |  57%
|
|===================                   |  58%
|
|====================                  |  59%
|
|====================                  |  60%
|
|====================                  |  61%
|
|=====================                 |  62%
|
|=====================                 |  63%
|
|=====================                 |  64%
|
|======================                |  64%
|
|======================                |  65%
|
|======================                |  66%
|
|======================                |  67%
|
|=======================               |  67%
|
|=======================               |  68%
|
|=======================               |  69%
|
|=======================               |  70%
|
|========================              |  70%
|
|========================              |  71%
|
```

```
|========================              |  72%
|
|========================              |  73%
|
|=========================             |  73%
|
|=========================             |  74%
|
|=========================             |  75%
|
|=========================             |  76%
|
|==========================            |  76%
|
|==========================            |  77%
|
|==========================            |  78%
|
|===========================           |  79%
|
|===========================           |  80%
|
|===========================           |  81%
|
|============================          |  82%
|
|============================          |  83%
|
|============================          |  84%
|
|=============================         |  84%
|
|=============================         |  85%
|
|=============================         |  86%
|
|=============================         |  87%
|
|==============================        |  87%
|
|==============================        |  88%
|
|==============================        |  89%
|
```

```
  |================================    |  90%
  |
  |================================    |  90%
  |
  |================================    |  91%
  |
  |================================    |  92%
  |
  |================================    |  93%
  |
  |================================    |  93%
  |
  |================================    |  94%
  |
  |================================    |  95%
  |
  |================================    |  96%
  |
  |================================    |  96%
  |
  |================================    |  97%
  |
  |================================    |  98%
  |
  |================================    |  99%
  |
  |================================    | 100%
## Matrix converted.. Prepare genind object...
##
## Finished! Took 0 seconds.
```

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



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

Send to a friend

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

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

All export functions start with gl2….

function	explanation
gl2fasta	Concatenates DArT trimmed sequences and outputs a fastA file
gl2shp	creates a shp file to be used with ArcGIS and the like
gl2structure	creates a file to be use with structure

function	explanation
gl2faststrcture	creates an input file to be used with faststructure
gl2svdquartets	Convert a genlight object to nexus format PAUP SVDquartets
gl.nhybrids	runs a newhybrids analysis (needs to be installed)
gl2gi	converts a genlight to a genind object

Fasta file

check methods

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

FastSTRUCTURE

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

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

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