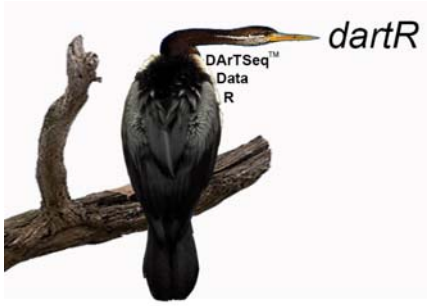
Wildlife Genomics@UC
wildlifegenomics.biomatix.org

UNIVERSITY OF CANBERRA

SNP Analysis using dartR



Guide to Preparatory Analysis

Copies of these workshop notes are available from:

The Institute for Applied Ecology
University of Canberra ACT 2601
Australia

Email: georges@aerg.canberra.edu.au

Contents

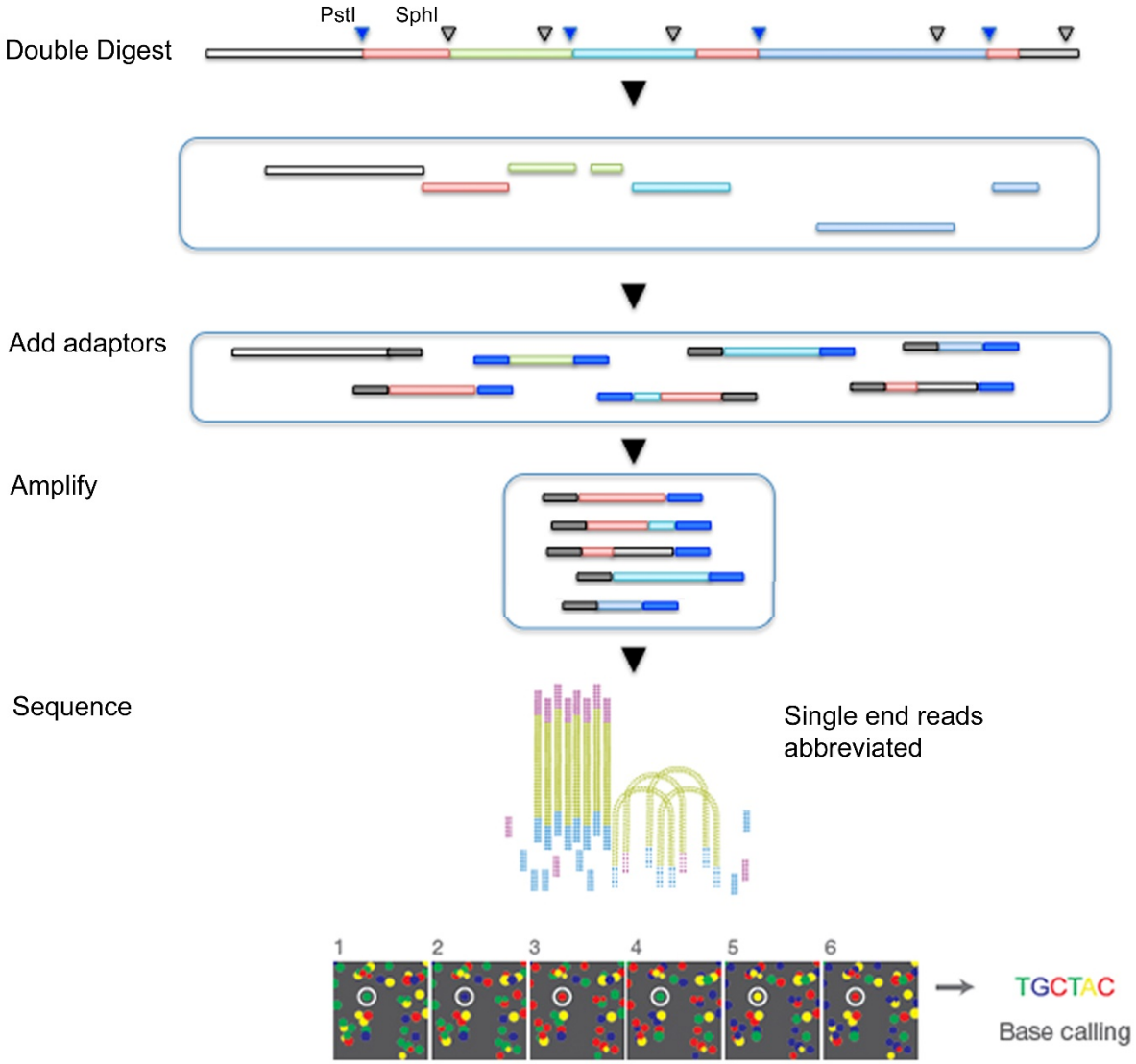
Session 1: Introduction to DArTSeq

Sequencing

Diversity Arrays Technology Pty Ltd (DArT PL) is a private company that specializes in genotyping by sequencing. Their approach is one of genome complexity reduction. But what does this mean?

Basically, DArTSeq is a method that digests genomic DNA using pairs of restriction enzymes (cutters). When the DNA is cut at two locations within a reasonable distance of each other, the fragment is available for sequencing using the Illumina short-read platforms.

The first step in the process involves the selection of restriction enzymes that provide the best balance between getting adequate fraction of the genome represented, an adequate read depth for each fragment, and adequate levels of polymorphism. This is species specific and so requires some initial optimization.



Once the best restriction enzymes are selected, say PstI (recognition sequence 5'-CTGCA|G-3') and SphI (5'-GCATG|C-3'), then the DNA is digested, and various adaptors added to the sequence fragments to allow Illumina short-read sequencing to proceed. These additional terminal sequences include a barcode to allow disaggregation of the sequences for each sample during later analysis.

The fragments of DNA selected by this process are sequenced in an abbreviated process to yield a set of raw "sequence tags" each of around 75 bp. They are filtered on sequence quality, particularly in the barcode region, truncated to 69 bp and stacked based on sequence similarity. A series of proprietary filters are then applied to select those sequence tags that include a reliable SNP marker.

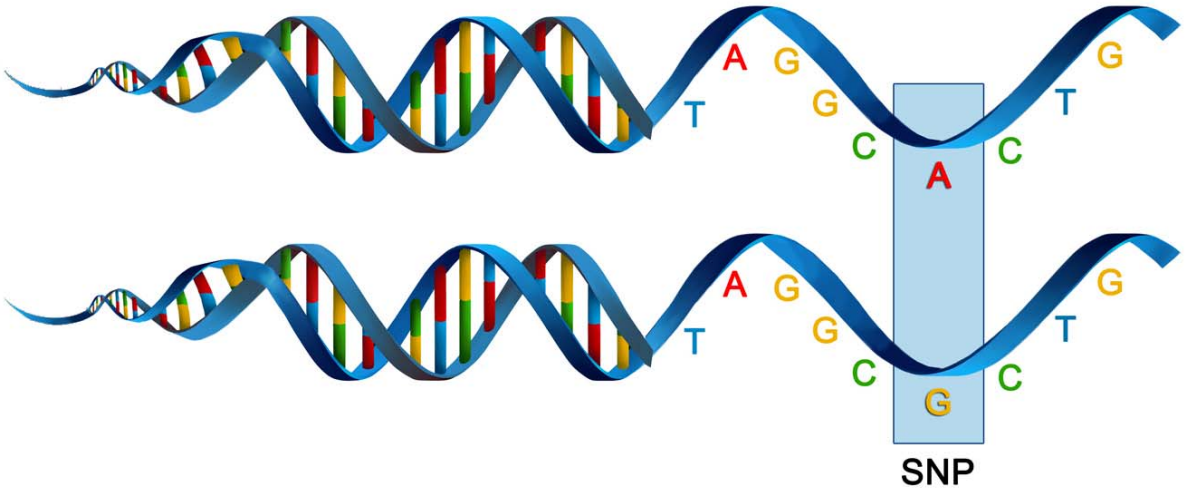
In particular, one third of samples are processed twice as technical replicates, from DNA and using independent adaptors, through to allelic calls. Scoring consistency (repeatability) is used as the main selection criterion for high quality/low error rate markers.

These DArT analysis pipelines have been tested against hundreds of controlled crosses to verify mendelian behaviour of the resultant SNPs as part of their commercial operations.

When you come to publish, you may receive requests to be more elaborative than you are able to, because of the proprietary nature of the pipelines. DArT Pty Ltd is a private company and needs to hold some of its proprietary analyses inhouse. Note that other companies with whom you interact, including Illumina, do the same. The work is reproducible in that using the same service/equipment on the same samples will yield the same result. Most journals accept this.

The SNP dataset

SNPs, or single nucleotide polymorphisms, are single base pair mutations at a nuclear locus. That nuclear locus is represented in the dataset by two sequence tags which, at a heterozygous locus, take on two allelic states, one referred to as the reference state, the other as the alternate or SNP state.



Because it is extremely rare for a mutation to occur twice at the same site in the genome (perhaps with the exception of Eucalypts), the SNP data are effectively biallelic.

The data can be represented in a table of SNP bases (A, T, C or G), with two states for each individual at each locus in a diploid organisms.

	Ind 01	Ind 02	Ind 03	Ind 04	Ind 05	Ind 06	Ind 07	Ind 08	Ind 09	Ind 10
Locus 1	A/A	A/A	A/A	A/A	A/G	A/A	A/A	A/A	A/A	-/-
Locus 2	C/C	C/C	C/C	C/C	C/C	C/C	C/T	C/C	C/C	C/C
Locus 3	C/G	G/G	G/G	G/G	G/G	C/C	C/C	C/C	C/C	C/C
Locus 4	A/A	A/T	A/A	A/T	T/T	A/A	A/A	A/A	A/A	A/A
Locus 5	A/A	A/A	A/A	A/A	-/-	A/G	A/A	A/A	A/A	A/A
Locus 6	C/C	C/C	C/C	C/C	C/C	C/C	C/T	C/C	C/C	C/C
Locus 7	C/G	G/G	G/G	G/G	G/G	C/C	C/C	C/C	C/C	C/C
Locus 8	A/A	A/T	A/A	A/T	T/T	A/A	A/A	A/A	A/A	A/A
Locus 9	A/A	A/A	A/A	A/A	A/A	A/A	A/A	A/A	A/A	A/A
Locus 10	C/C	C/C	C/C	C/C	C/C	C/C	C/T	C/C	C/C	C/C
Locus 11	C/G	G/G	G/G	G/G	G/G	C/C	C/C	C/C	C/C	C/C

Alternatively, because the data are biallelic, it is convenient to code the data as coded 0 for homozyogotes for one allele, 1 for heterozygotes, and 2 for homozygotes of the other allele.

The reference allele is arbitrarily taken to be the most common allele, so 0 is the score for homozygous reference, and 2 is the score for homozygous alternate or SNP state. NA indicates that the SNP could not be scored.

	Ind01	Ind02	Ind03	Ind04	Ind05	Ind06	Ind07	Ind08	Ind09	Ind10
Locus 1	0	0	0	0	1	0	0	0	0	NA
Locus 2	0	0	0	0	0	0	1	0	0	0
Locus 3	1	2	2	2	2	0	0	0	0	0
Locus 4	0	1	0	1	2	0	0	0	0	0
Locus 5	0	0	0	0	NA	1	0	0	0	0
Locus 6	0	0	0	0	0	0	1	0	0	0
Locus 7	1	2	2	2	2	0	0	0	0	0
Locus 8	0	1	0	1	2	0	0	0	0	0
Locus 9	0	0	0	0	0	0	0	0	0	0
Locus 10	0	0	0	0	0	0	1	0	0	0
Locus 11	1	2	2	2	2	0	0	0	0	0

This is the form the data are stored in in dartR, though note that it departs from the coding arrangement used by DArT PL.

Some sequence tags might contain more than one SNP, in which case they are likely to be closely linked when passed from parent to offspring. These may need consideration when preparing your data for analysis.

The SNP data are provided in two forms by DArT, which will be described later.

SilicoDArT

As well as individuals varying in allelic composition at SNP sites, they can vary at the restriction sites used to pull the representation from the genome. A mutation at one or both of the restriction sites will result in allelic drop-out or null alleles. The presence or absence of particular sequence tags across individuals provides a source of information additional to the SNP data.

Broadly, SilicoDArT markers can be considered analogous to AFLPs (Amplified Fragment Length Polymorphisms).

DArT PL provide this second dataset, the presence or absence of scored sequence tags across individuals in what it calls the SilicoDArT dataset. The filtering pipeline

applied to generate these data has been highly optimized for reliability, so do not be tempted to use the null alleles (missing data) present in the SNP dataset.

Where have we come?

The above Session was designed to give you a very brief overview to the Diversity Arrays Technology pipelines for producing SNP and associated data. Having completed this Session, you should now be familiar the following concepts.

- The concept of a SNP marker and how they are generated.
- The distinction between DArTSeq and SilicoDArT datasets.
- The coding used for genotypes – 0 for homozygous reference, 2 for homozygous alternate, 1 for heterozygous, and NA for missing.

Further reading

Sansaloni, C., Petroli, C., Jaccoud, D., Carling, J., Detering, F., Grattapaglia, D., & Kilian, A. (2011). Diversity Arrays Technology (DArT) and next-generation sequencing combined: genome-wide, high throughput, highly informative genotyping for molecular breeding of *Eucalyptus*. BMC Proceedings 5(Suppl 7), P54. doi:10.1186/1753-6561-5-S7-P54.

Kilian, A., Wenzl, P., Huttner, E., Carling, J., Xia, L., Blois, H., … Uszynski, G. (2012). Diversity arrays technology: a generic genome profiling technology on open platforms. *Methods in Molecular Biology 888*:67–89.

Session 2: Getting Started

Install dartR

To be able to install dartR properly we recommend installation of a recent version of R (at least version 3.5) and Rstudio (at least version 1.1.463).

Installing from CRAN

As dartR is now a fullyfledged package, simply type:

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

then load it with

```
library(dartR)
```

Occasionally, this fails. If it does, you should read the warnings and errors, identify the packages that have not loaded, install them manually using `install.packages`, and try again.

Install dartR and then load the library

Installing from GitHub

The latest stable version can be installed directly from GitHub.

```
install.packages("devtools")
library(devtools)
install_github("green-striped-gecko/dartR")
library(dartR)
```

Developmental version

The latest developmental version can also be installed from GitHub. Note that this is a Beta version, and may contain bugs.

```
install.packages("devtools")
library(devtools)
install_github("green-striped-gecko/dartR", ref="dev")
library(dartR)
```

If something goes wrong

If some packages have not been loaded, an error message will be given and you will need to install those packages manually. Sometimes packages from the Bioconductor repository are not installed (e.g. SNPRelate or qvalue). If this is the case copy paste the following code:

```
install.packages("devtools")
library(devtools)
install.packages("BiocManager")
BiocManager::install(c("SNPRelate", "qvalue"))
```

```
install_github("green-striped-gecko/dartR")
library(dartR)
```

Additional help

Additional information can be found at the github pages:

https://github.com/green-striped-gecko/dartR

You can post questions on the GitHub site or on Google group dartR

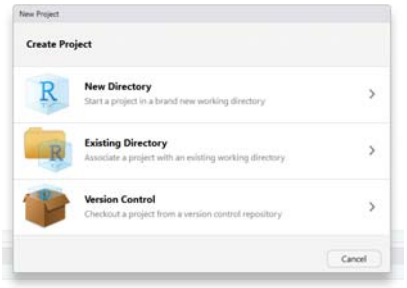
https://groups.google.com/forum/#!forum/dartr

Create a Project

The next step is to set up a project, that is, a self-contained set of analyses. You do this by pulling down the `File | New Project` menu, and following the prompts. You will be asked to nominate a new working directory or to link to an existing working directory. Remember to keep your projects distinct, by assigning a working directory to each discrete project.

Once you have established a project, you can now leave RStudio and pick up where you left off next time, by simply loading the project again.

It is usual to have multiple projects on the go.



Set a Working Directory

Once installed and invoked into your current R session, we need to explicitly confirm the working directory. The working directory is the location on your hard drive, that holds your data and scripts.

```
setwd("c:/your.working.directory/")
```

Note that the file specification uses forward slashes, which works for the three main Operating Systems (Linux, Windows and MacOS).

The above statement would appear early in all of your scripts.

CRAN forces us to save files by default to a temporary directory, which is often difficult to locate. You need to pay attention to this in functions that write to disk. Essentially, you need to set the outpath at the time of running scripts that write to disk.

```
new <- function (....., outpath <- getwd(), .....)
```

This will ensure that output files generated by the invocation of a particular function will be stored in your working directory, which is where you would normally expect them to be.

We are highly disciplined about the files we save to disk, so this should not present you with any problems.

Downloading the sample files for this tutorial

dartR has some datsets that you will access from time to time, the most commonly used one being testset.gl. This small genlight object is used to illustrate functions quickly, as part of the help associated with each function.

There are also some sample files in the format provided by Diversity Arrays Technology Pty Ltd.

- sample_data_1Row.csv
- sample_data_2Row.csv
- sample_data_silicodart.csv
- sample_ind_metrics.csv
- sample_metadata.xlsx

These files will serve to illustrate how to get DArT PL data into dartR.

Where have we come?

Hopefully you have been able to install dartR. Most times the installation works seamlessly; sometimes the installation is a little difficult. If you have been unable to get it installed, post to

https://groups.google.com/forum/#!forum/dartr

On completion of this session, you will have

- Reinforced your knowledge of installing packages;
- Learned how to initiate a project to assist with workflow management;
- Learned how to set a working directory and the repository for output files;
- Been provided with some real datasets to play with.

Session 3: Getting data into dartR

If you are coming back to us, create or load a project, set a working directory, and do not forget to set the default directory for files, the outpath, to getwd(). Refer to the previous Session.

A sensible pipeline

Let us begin by jumping the gun and defining a sensible pipeline for entering your data, as a context for the material in this and subsequent Sessions.

1. Examine the data provided by DArT PL in Excel to confirm that it conforms to expectations of the dartR package. There needs to be a `AlleleID` column, and the locus metadata needs to finish with the column `RepAvg`. The row with the locus metadata labels needs to be the same row that holds the individual (= specimen or sample labels). This is usually the case, but some older datasets may need a little modification.
2. Read the data into dartR
3. Correct any errors that might have crept in on individual labels, population labels, and remove individuals that perhaps should not be there, etc.
4. Examine the data to ensure it has the correct number of individuals, the correct number of loci, and that populations are correctly defined.
5. Remove any monomorphic loci and recalculate the locus metrics.
6. Save the genlight object in binary form for future use. This clean, unfiltered dataset will be your point of access to the data in future analyses.

Once you have completed these steps, your preparatory analysis is complete, and you are ready to proceed to some more substantive analyses, like phylogenetics,analysis of relatedness, spatial analysis or indeed any analysis using third party software.

How dartR stores data

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

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

A genlight object can be considered to be a matrix containing the SNP data encoded in a particular way. The matrix entities (rows) are the individuals, and the attributes (columns) are the SNP loci. In the body of this individual x locus matrix

are the SNP data, coded as 0 for homozygous reference state, 1 for heterozygous, and 2 for homozygous alternate (or SNP) state.

The genelight objects used in dartR not only have the SNP data, but also allow for attachment of locus metadata to the loci, and attachment of individual metadata to the individuals/samples. This is represented diagrammatically below.

Locus metadata

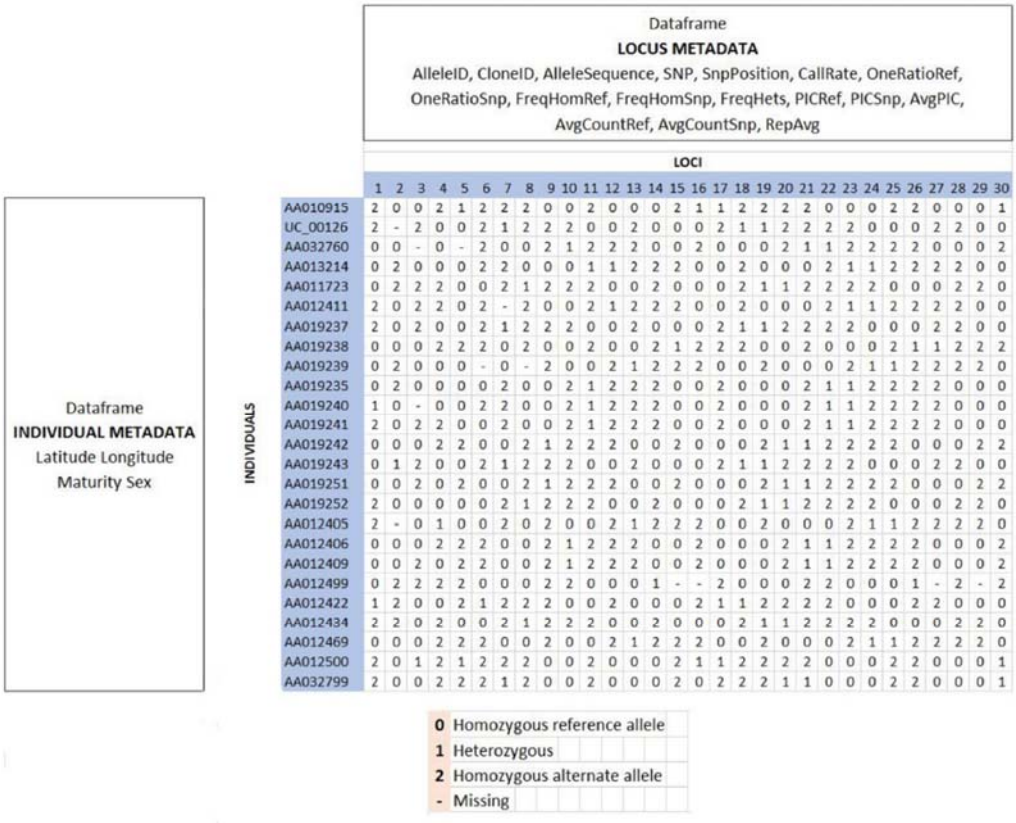
The locus metadata included in the genlight object are those provided as part of your DArT PL report. These metadata are obtained from the DArT PL csv file when it is read in to the genlight object. The locus metadata are held in an R data.frame that is associated with the SNP data as part of the genlight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

The locus metadata would typically include:

SNP — the mutational change and its position in the sequence tag, referenced from zero

SnpPosition — position (zero is position 1) in the sequence tag of the defined SNP variant base

TrimmedSequence (optional) — The sequence containing the SNP or SNPs (the sequence tag), trimmed of adaptors.



CallRate — proportion of samples for which the genotype call is non-missing (that is, not "-")

OneRatioRef — proportion of samples for which the genotype score is 0

OneRatioSnp	proportion of samples for which the genotype score is 2
FreqHomRef	proportion of samples homozygous for the Reference allele
FreqHomSnp	proportion of samples homozygous for the Alternate (SNP) allele
FreqHets	proportion of samples which score as heterozygous, that is, scored as 1
PICRef	polymorphism information content (PIC) for the Reference allele
PICSnp	polymorphism information content (PIC) for the SNP
AvgPIC	average of the polymorphism information content (PIC) of the Reference and SNP alleles
AvgCountRef	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	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	proportion of technical replicate assay pairs for which the marker score is consistent

In addition, dartR calculates the minor allele frequency and stores it in the locus metadata.

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

```
# Make a genlight object to work with
  gl <- testset.gl

# Only entries for the first 10 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:

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

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

Examine the first 10 values of RepAvg, CallRate and some other listed locus metadata in `testset.gl` and your own dataset.

Depending on the report from DarT you may have additional (fewer) loc.metrics (e.g. `TrimmedSequence` 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/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:

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

Individual 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 using:

```
pop(gl)
```

```
popNames(gl)
```

```
indNames(gl)
```

Try these for yourself to see the output they produce.

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

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 at the time of reading in the data.

The individual metadata are held in the genlight object as a dataframe named `ind.metrics`. You can check the names of all available `ind.metrics` via:

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

```
[1] "id"       "pop"      "lat"      "lon"      "sex"      "maturity" "collector" "location"  "basin"    "drainage"
```

and can be accessed using the following form:

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

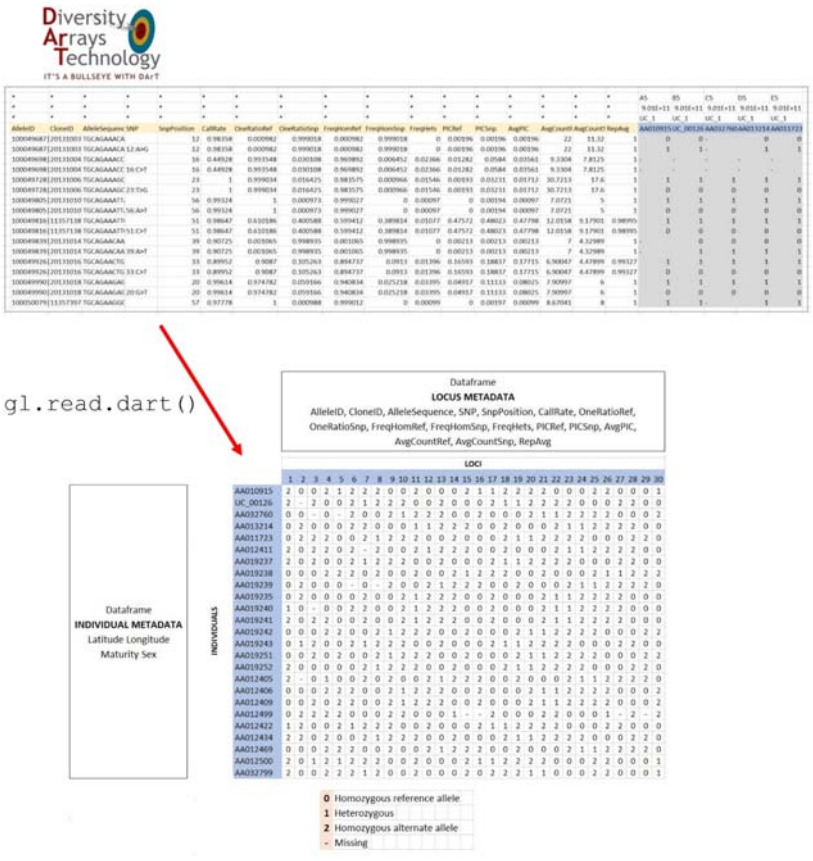
```
[1] Male   Male   Male   Male   Unknown Male   Female Female Male   Female
Levels: Female Male Unknown
```

Try these for yourself to see the output they produce.

Reading DArT files into a genlight object

SNP data can be read into a genlight object using `gl.read.dart()`. This function intelligently interrogates the input csv file to determine

- if the file is a 1-row or 2-row format, as supplied by Diversity Arrays Technology Pty Ltd.
- the number of locus metadata columns to be input (the first typically being `AlleleID` and the last `RepAvg`).
- the number of lines to skip at the top of the csv file before reading the specimen IDs and then the SNP data themselves.
- if there are any errors in the data.



An example of the function used to input data is as follows:

```
gl <- gl.read.dart(
  filename="sample_data_2Row.csv",
  ind.metafile="sample_metadata.csv")
```

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

The resultant genlight object contains the SNP genotypes, the individual metadata and the locus metadata.

If you have saved the sample files provided into your working directory, read `sample_data_2Row.csv` into a genlight object using `DFwt16-sample_metadata.csv` to assign individual metics.

Verify that the genlight object contains the right number of loci, individuals and populations.

Reading non-DArT files into a genlight object

If you are working with data that have not been prepared by Diversity Arrays Technology, you can still input the data to dartR provided you can get it into the appropriate format.

The SNP data need to be in one of two forms. SNPs can be coded 0 for homozygous reference, 2 for homozygous alternate, and 1 for heterozygous with NA for missing values; or it can be coded A/A, A/C, C/T, G/A etc, with -/- as missing. Other formats will thow an error.

The first column of the matrix should be the individual labels, and the first row should be the locus labels.

The individual metadata needs to be in a csv file, with headings, with a mandatory id column corresponding exactly to the identities provided with the SNP data.

The locus metadata needs to be in a csv file with headings. Note that the locus metadata will be complemented by calculable statistics corresponding to those that would be provided by Diversity Arrays Technology (e.g. CallRate).

The script will then calculate the locus metrics that can be calculated, such as Call Rate. Others locus metrics, such as the trimmed sequences need to be provided in the locus metrics file if you wish to use them later.

Once you have the data in this form, you can read it into a genlight object using

```
gl <- gl.read.csv(filename="snp_matrix.csv",
      ind.metafile = "ind_metrics.csv",
      loc.metafile = "loc.metrics.csv")
```

Saving a genlight object

Reading the data in from an Excel spreadsheet and converting to a genlight object takes a lot of computation, and so time. You will also have done some tidying up of the data. It is sensible to save your genlight object in binary form using

```
saveRDS(gl,file="tmp.Rdata")
```

and then read it in again with

```
gl.new <- readRDS("tmp.Rdata")
```

Try saving `gl` or your own genlight object to your workspace, and verify that it has been saved to the appropriate directory. Then restore it to a new genlight object.

Exercises

Exercise 1

2-Row Format

Open the file sample_data_2Row.csv in Excel. This is a set of SNP data for *Emydura*, a freshwater turtle, in 2-row format as would be supplied by DArT Pty Ltd.

Refer to the documentation on the Diversity Arrays Technology web page to understand the scoring of SNPs in the 2-row format.

Also refer to the MetaDataDefinition file provided by DArT Pl as part of their report. In this case, a definition file is provided as sample_metadata.xlsx.

Now examine the individual metadata in the file sample_metadata.csv. Note the two mandatory columns id and pop.

Read the SNP data in to dartR as a genlight object called gl.2row and check the contents.

Exercise 2

1-Row Format

Open the file sample_data_1Row.csv in Excel. This is a set of SNP data provided in 1-row format by DArT PL.

Refer to the documentation on the Diversity Arrays Technology web page to understand the scoring of SNPs in the 1-row format.

Read the SNP data in to dartR as a genlight object called gl.1row and

	check the contents.

Exercise 3

SilicoDArT

Open the file sample_data_silicodart.csv in Excel. This is a set of marker presence/absence data for Cherax destructor provided in SilicoDArT format by DArT Pty Ltd.

Refer to the documentation on the Diversity Arrays Technology web page to understand the scoring of the data in the SilicoDArT format.

At this stage, dartR does not support analysis of SilicoDArt data.

Tidy up the workspace

We have created files that we will not use again, so they should be removed from the workspace.

```
rm(gl.new, gl.1row, gl.2row)
```

Where have we come?

In this Session, we have covered a range of topics on data entry, the storage of data and some preliminary approaches to examining those data. Having completed the Session, you should understand

- What is a sensible pipeline for preliminary handling of your SNP data.
- How a genlight object is organised in terms of the SNP scores (which are different from the scores used by DArT PL) and how locus and sample metadata are associated with the genotypes.
- The different types of locus metadata generated by DArT PL, and how to look up what each metric means.
- How to read data from DArT Pty Ltd into a genlight object.
- How to interrogate the locus and individual (specimen/sample) metadata.

References

Jombart T. and Caitlin Collins, C. (2015). Analysing genome-wide SNP data using adegenet 2.0.0. http://adegenet.r-forge.r-project.org/files/tutorial-genomics.pdf

Jombart T. and Ahmed, I. (2011). *adegenet 1.3-1*: new tools for the analysis of genome-wide SNP data. *Bioinformatics*, 27: 3070–3071.

Jombart, T., Kamvar, Z.N., Collins, C., Lustrik, R., Beugin, M.P., Knaus, B.J., Solymos, P., Mikryukov, V., Schliep, K., Maié, T., Morkovsky, L., Ahmed, I., Cori, A., Calboli, F. and Ewing, R.J. (2018). Package 'adegenet'. Version 2.1.1. Exploratory Analysis of Genetic and Genomic Data. https://github.com/thibautjombart/adegenet

Session 4: Interrogating a genlight object

If you are coming back to us, load your project, confirm the working directory, and do not forget to set the default directory for files, the outpath, to `getwd()`. Refer to Session 3.

Examine attributes

The first step in examining your genlight object is to simply type its name. This will give you the attributes associated with the object, which is of Class "genlight".

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

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

 // 250 genotypes,  255 binary SNPs, size: 751.3 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
```

Try this for yourself using `gl` or the genlight object containing your own data.

Displayed are the number of genotypes (individuals/specimens/samples), the size of the genlight object, and the number of missing values. The ploidy value should be 2-2 for a diploid organism (dartR does not have support for polyploid organisms), so if it is something else, you have a problem with your data.

Slots containing important information are listed, such as `@position`, which lists the position of the SNPs in the sequence tags (referenced from 0 as position 1). The `@other` slot is particularly important, because it holds the `loc.metrics` from DArT PL and your `ind.metrics`.

Examine metrics

To access metadata directly you can use commands of the form

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

Try this for yourself to examine `CallRate`, `RepAvg`, `rdepth` and other locus metrics in `testset.gl`. See if you can access individual metadata like `sex`.

Remind yourself of the variables in the metadata using

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

and

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

Use adegenet accessors

You can also interrogate the genlight object using commands built into the {adegenet} package.

`nLoc(gl)`	number of loci
`locNames(gl)`	list of loci
`nInd(gl)`	number of individuals (specimens or samples)
`indNames(gl)`	list of individuals
`nPop(gl)`	number of populations
`popNames(gl)`	list of populations
`pop(gl)`	list of population assignments for each individual
`as.matrix(gl)`	generate a matrix of the SNP scores, with 0 as homozygous reference, 2 as homozygous alternate, and 1 as heterozygous.
`gl.plot(gl)`	a smear plot of individual against locus, useful for gross pattern identification and assessment of allelic dropout. This version adds individual labes and is only useful if there are not too many individuals.

These are all useful for interrogating your genlight object, and of course can be used in your r scripts to subset and manipulate your data.

Exercises

1. Copy the genlight object `testset.gl` to a working genlight object `gl`.
2. How many loci are represented in this dataset?
3. How many individuals have been scored?
4. Are the individuals assigned to populations and if so, how many populations? What are the names of the populations?
5. Examine the genotypes for the first 5 individuals for the first 10 loci.
6. How are missing values represented?
7. Examine the structure of the dataset in a smear plot. What can you say about allelic dropout?
8. Redo activities 2-7 with your own data.

Use dartR report functions

Report scripts provide a summary of locus metrics to assist you with decisions on appropriate thresholds for filtering. Note: Do not assign the output of the report to your genlight object or you will overwrite it.

`gl.report.callrate`	summarises CallRate values
`gl.report.repavg`	summarises repAvg values
`gl.report.monomorphs`	provides a count of polymorphic and monomorphic loci
`gl.report.secondaries`	provides a count of loci that are secondaries, that is, loci that have more than one sequence tag represented in the dataset.
`gl.report.maf`	reports on minor allele frequencies overall, and by population.
`gl.report.hamming`	reports on Hamming distances between sequence tags.
`gl.report.hwe`	reports on loci that depart from Hardy Weinberg equilibrium.
`gl.report.heterozygosity`	reports on heterozygosity by population.

Example: Reporting on Call Rate

As an example, let us consider reporting on Call Rate.

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

```
Starting gl.report.callrate: Reporting distribution of Call Rate
Starting utils.recalc.callrate: Recalculating CallRate
Completed utils.recalc.callrate

Reporting Call Rate by Locus
No. of loci = 255
No. of individuals = 250
  Miniumum Call Rate:  0.06
  Maximum Call Rate:  1
  Average Call Rate:  0.877
  Missing Rate Overall:  0.12

   Threshold Retained Percent Filtered Percent
1       1.00       78    30.6      177    69.4
2       0.95      173    67.8       82    32.2
3       0.90      197    77.3       58    22.7
4       0.85      203    79.6       52    20.4
5       0.80      209    82.0       46    18.0
6       0.75      213    83.5       42    16.5
7       0.70      216    84.7       39    15.3
8       0.65      219    85.9       36    14.1
9       0.60      223    87.5       32    12.5
10      0.55      226    88.6       29    11.4
11      0.50      228    89.4       27    10.6
12      0.45      230    90.2       25     9.8
13      0.40      234    91.8       21     8.2
14      0.35      236    92.5       19     7.5
15      0.30      239    93.7       16     6.3
16      0.25      242    94.9       13     5.1
17      0.20      245    96.1       10     3.9
18      0.15      252    98.8        3     1.2
```

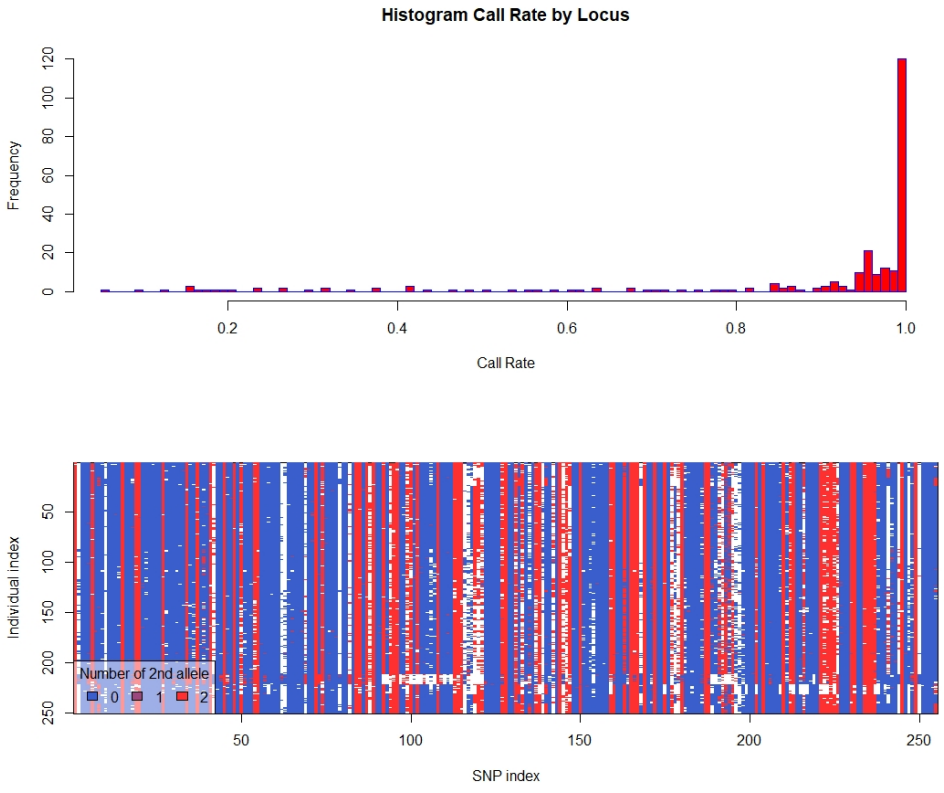
```
19      0.10      253    99.2        2     0.8
20      0.05      255   100.0        0     0.0
21      0.00      255   100.0        0     0.0

gl.report.callrate Completed
```

Try this for yourself

The report is fairly self-evident. The table provides you with a basis for deciding an appropriate Call Rate to use as a threshold for filtering loci. For example, if you reject all loci that fail to call in any individual, you will lose all but 30% of your loci. That is too stringent. However, a threshold of 0.95 might be appropriate, with 67.8% of loci retained.

By setting plot=TRUE, the table is presented as a histogram and smearplot=TRUE generates a smearplot (computationally intensive for large datasets).

```
gl.report.callrate(gl, plot=TRUE, smearplot=TRUE)
```



Try this for yourself, this time setting plot=TRUE and smearplot=TRUE

The smear plot is informative, as the loci that failed to be called for particular individuals are shown as a white pixel. Loci with poor call rates are seen as white-dominated columns. Individuals with poor call rates are seen as white-dominated rows.

By default, the function gl.report.callrate will calculate call rates by locus. To calculate call rates by individual, use

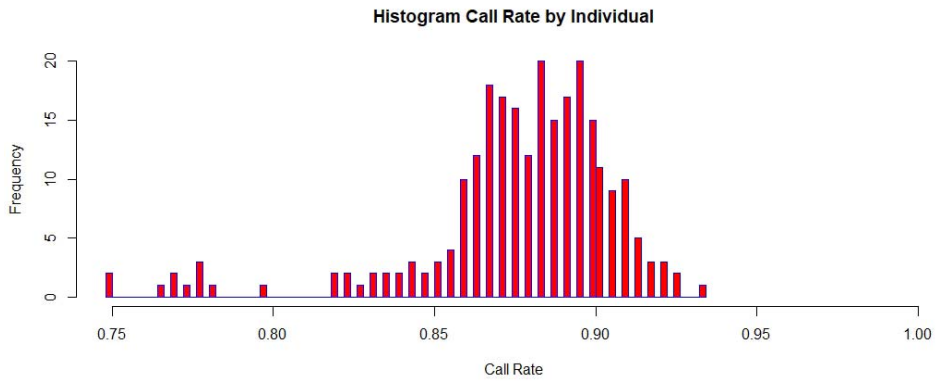
```
gl.report.callrate(gl, method="ind")
```

```
Starting gl.report.callrate: Reporting distribution of Call Rate
Starting utils.recalc.callrate: Recalculating CallRate
Completed utils.recalc.callrate

Reporting Call Rate by Individual
No. of loci = 255
No. of individuals = 250
  Miniumum Call Rate:  0.75
  Maximum Call Rate:  0.93
  Average Call Rate:  0.877
  Missing Rate Overall:  0.12

   Threshold Retained Percent Filtered Percent
1       1.00        0     0.0      250   100.0
2       0.95        0     0.0      250   100.0
3       0.90       44    17.6      206    82.4
4       0.85      223    89.2       27    10.8
5       0.80      239    95.6       11     4.4
6       0.75      248    99.2        2     0.8
7       0.70      250   100.0        0     0.0
8       0.65      250   100.0        0     0.0
9       0.60      250   100.0        0     0.0
10      0.55      250   100.0        0     0.0
11      0.50      250   100.0        0     0.0
12      0.45      250   100.0        0     0.0
13      0.40      250   100.0        0     0.0
14      0.35      250   100.0        0     0.0
15      0.30      250   100.0        0     0.0
16      0.25      250   100.0        0     0.0
17      0.20      250   100.0        0     0.0
18      0.15      250   100.0        0     0.0
19      0.10      250   100.0        0     0.0
20      0.05      250   100.0        0     0.0
21      0.00      250   100.0        0     0.0

gl.report.callrate Completed
```

The table provides you with a basis for deciding an appropriate Call Rate to use as a threshold for filtering individuals. No individuals have 100% call rate. If you set the threshold at 0.8, 4.4% of individuals will be discarded on filtering.

By setting plot=TRUE, the table is presented as a histogram.

```
gl.report.callrate(gl, method="ind", plot=TRUE)
```



The smear plot is the same as was output for the report against loci.

Try this for yourself, filtering call rate by individual instead of by locus

All of the report functions operate in a similar, though not identical, manner.

Generate reports for `RepAvg, maf, rdepth, secondaries` and other locus metadata in `gl`. Try it on your own data.

Where have we come?

In this Session, you have learnt the various ways to interrogate a genlight object, namely by

- simply typing its name
- examining the locus and individual metrics
- using adegenet accessors
- using the dartR report functions

These techniques are important for familiarizing yourself with your dataset at the most fundamental level, and for detecting any errors or outliers likely to cause problems in later analyses.

Session 5. Data Manipulation

If you are coming back to us, load your project, set a working directory, and do not forget to set the default directory for files, the outpath, to `getwd()`. Refer to Session 3.

Overview

Almost certainly you will need to undertake various manipulations of your data prior to analysis. This might include altering or correcting some individual or population labels, removing some individuals included in error or perhaps duplicated, and recalculating the locus metrics to include some new metrics not provided by DArT PL.

During the course of analysis as well, you will need to amalgamate individuals from sampling sites, delete populations (say the outgroup taxa for an analysis directed at the ingroup taxa alone), or examine statistics from one population on its own.

An early cautionary note

While it is possible to manipulate the content of genlight objects directly, this can result in serious miscalculation. For example, deleting loci directly with

```
index <- glNA(gl, alleleAsUnit=FALSE)<= 0.05*nInd(gl))
gl <- gl[, index]
```

is fine, provided you remember to apply the restriction to the locus metrics as well. Otherwise, the locus metrics will become out of sync with the loci, which is potentially disasterous.

```
gl@other$loc.metrics <- gl@other$loc.metrics[index,]
```

Even subsetting is potentially problematic. For example

```
gl <- gl[,1:100]
```

will generate a genlight object containing only the first 100 loci, but the locus metrics will retain data for all the original loci. This is a bug in adegenet.

So even if you are well versed in R programming, and tempted to take shortcuts, they are best avoided.

Instead, there are functions in dartR to assist you in manipulating a genlight object. It is safest to use them.

Removing populations and individuals

Dropping populations and individuals

The following set of functions apply to populations and individuals.

`gl <- gl.drop.pop()` remove listed populations

`gl <- gl.keep.pop()` keep only the listed populations

`gl <- gl.drop.ind()` remove listed individuals

`gl <- gl.keep.ind()` keep only the listed individuals

Selection on an individual metric

You can select individuals based on one of your locus metrics using

```
gl <- gl.keep.pop(gl,"Male", as.pop="sex")
```

where sex is one of the variables among the individual metrics in `gl@other$ind.metrics`, and where you want to retain only the males. This parameter works with `gl.drop.pop` also.

Replacing the pop variable

It may also be convenient to assign one of your individual metrics as the population variable. Some analysts have a series of different population breakdowns in the individual metrics, and move from one to the other during analysis. For example, the `popNames(gl)` might be based on sample sites, but another metric might be 'region'. The region can be assigned as the `pop` variable using

```
gl <- gl.reassign.pop(gl, as.pop="sex")
```

This will overwrite the existing population assignments with values of the individual metric 'sex'.

Copy `testset.gl` to a new genlight object, and list the populations to which individuals are assigned. Delete one or more populations. Delete one or more individuals. Confirm that the population(s) or individual(s) has been dropped.

Try this with your own data.

Recalculating locus metrics

These scripts delete individuals, and so many of the metadata variables provided by DArT Pty Ltd no longer apply. For example, deleting individuals with a low call rate across loci will alter the call rate and so the `CallRate` provided by DArT Pty Ltd will be incorrect in the new genlight object.

There are parameters to set in each of these scripts to recalculate the locus metadata, but it is best to do your manipulations and then run

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

This script will also create a new locus metadata variable containing minor allele frequencies, `maf`, if it does not already exist.

Recalculate the locus metrics for your new genlight object, using a verbose = 3 to track what the function does.

These scripts will also generate monomorphic loci, where the deleted individuals have all the variation at a locus, and these monomorphic loci can be deleted using

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

Examine your new genlight object with a smear plot. Remove monomorphic loci (and those that are all NA) from your new genlight object. Again, set verbosity to 3. Re-examine your new genlight object with a smear plot.

You can define a new population for a set of listed individuals, merge two populations, rename a population, using

`gl <- gl.define.pop()`	create a new population for listed individuals
`gl <- gl.merge.pop()`	merge two populations under a new name, or if applied to one population, to rename it.

Create a new genlight object from `gl`, at the same time merging two selected populations into one. Confirm that the population reassignment has been achieved.

List the individuals in the genlight object `gl`. Select two individuals and assign them to a new population called `newguys`. Confirm that the population reassignment has been achieved.

Recode tables

An alternative way to manipulate population assignments and individual labels is to use a recode table in the form of a comma-delimited csv file. This is efficient if you have a lot of changes or if you want to keep a good record of individual recoding and population assignments.

`gl <- gl.make.recode.pop()`	make a recode table based on existing population labels. You will need to edit the second column of the recode table to specify the new labels to apply.
`gl <- gl.make.recode.ind()`	make a recode table based on existing individual labels. You will need to edit the second column of the recode table to specify the new labels to apply. Individuals assigned the new label 'Delete' will be removed from the genlight object.
`gl <- gl.recode.pop()`	apply the specified pop.recode table to the populations
`gl <- gl.recode.ind()`	apply the specified ind.recode table to the individuals

`gl <- gl.edit.recode.pop()`	edit population assignments, and apply the changes on closure
`gl <- gl.edit.recode.ind()`	edit population assignments, and apply the changes on closure

Note that populations or individuals assigned the label 'Delete' will be removed from the genlight object when the recode table is applied. If this occurs, you will need to run

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

to recompute the locus metrics.

1. Just in case you have accidentally modified the genlight object `gl`, recreate it by copying it from `testset.gl`. List the populations and the number of populations.
2. Use `gl.make.recode.pop` to create a draft recode table with a specified name (make sure it is a `.csv` file). Edit this in excel to make changes to the population assignments. Make one population assignment `Delete`.
3. Apply the recode table to genlight object `gl`.
4. List the populations and the number of populations. Have the anticipated changes been made?
5. Now try editing the population assignments with `gl.edit.recode.pop`. List the populations and the number of populations. Have the anticipated changes been made?
6. If you deleted a population, be sure to filter out monomorphic loci and to recalculate the locus metadata.

Tidy up the workspace

We have created files that we will not use again, so they should be removed from the workspace. Check the list under the tab Environment, and use `rm()`.

```
rm(newguys)
```

Where have we come?

The above Session was designed to give you an overview of the scripts in dartR for undertaking basic manipulations of your data. Having completed this Session, you should now able to:

- Visualize your data in a smear plot to assess structure and frequency of null alleles.
- Delete individuals and populations, rename individuals and populations, merge populations, assign individuals to new populations.
- Recalculate locus metrics after deleting individuals or populations.

Session 6: Basic Filtering

Overview

DArT Pty Ltd has already done much of the filtering of the sequences used to generate your SNPs that would normally be undertaken by researchers who generate their own ddRAD data. Here we present some other filters that you might wish to apply.

It is a good idea to run the gl.report functions (Session 4) in advance of filtering to provide a foundation for selecting thresholds.

Several filters are available to improve the quality of the data represented in your genlight object.

`gl <- gl.filter.repavg()`	filter out loci for which the repeatability is less than a specified threshold, say threshold = 0.99
`gl <- gl.filter.callrate()`	filter out loci for which the call rate (rate of non-missing values) is less than a specified threshold, say threshold = 0.95
`gl <- gl.filter.maf()`	filter on minor allele frequency
`gl <-gl.filter.secondaries()`	filter out SNPs that share a sequence tag, except one retained at random
`gl <- gl.filter.hamming()`	filter out loci that differ from each other by less than a specified number of base pairs
`gl <- gl.filter.monomorphs()`	filter out monomorphic loci and loci that are scored all NA
`gl <- gl.filter.rdepth()`	filter out loci with exceptionally low or high read depth (coverage)

The filter `gl.filter.callrate` function can also be applied to individuals, typically with a lower threshold (say 0.80), in which case the locus metrics will need to be recalculated.

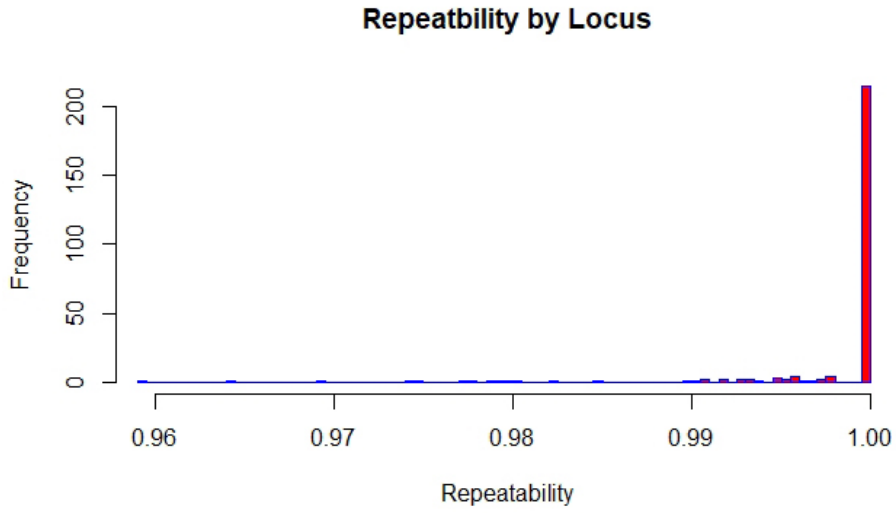
The order of filtering can be important and requires some thought. Filtering on call rate by individual before filtering on call rate by locus or choosing the alternative order will depend on the weight placed on losing individuals versus losing loci, for example.

Example: Filtering on repeatability

First, we should examine the distribution of repeatability measures (RepAvg) in our dataset.

```
gl <- testset.gl
gl.report.repavg(gl,plot=TRUE)
```

```
Starting gl.report.repavg: Reporting distribution of Repeatability (repAvg)
No. of loci = 255
No. of individuals = 250
  Miniumum repeatability:  0.96
  Maximum repeatability:  1
  Mean repeatability:  0.998

   Threshold Retained Percent Filtered Percent
1  1.0000000      214    83.9       41    16.1
2  0.9979729      214    83.9       41    16.1
3  0.9959459      222    87.1       33    12.9
4  0.9939189      231    90.6       24     9.4
5  0.9918918      237    92.9       18     7.1
6  0.9898647      242    94.9       13     5.1
7  0.9878377      242    94.9       13     5.1
8  0.9858107      242    94.9       13     5.1
9  0.9837836      243    95.3       12     4.7
10 0.9817565      244    95.7       11     4.3
11 0.9797295      246    96.5        9     3.5
12 0.9777025      248    97.3        7     2.7
13 0.9756754      250    98.0        5     2.0
14 0.9736483      252    98.8        3     1.2
15 0.9716213      252    98.8        3     1.2
16 0.9695942      252    98.8        3     1.2
17 0.9675672      253    99.2        2     0.8
18 0.9655401      253    99.2        2     0.8
19 0.9635131      254    99.6        1     0.4
20 0.9614860      254    99.6        1     0.4
21 0.9594590      255   100.0        0     0.0

gl.report.repavg Completed
```



Clearly, with a minimum repeatability of 0.96 and a maximum of 1 across loci, we can be fairly stringent in our choice of a threshold. A value of 0.99 will not result in the loss of much data.

We now filter on that basis.

```
gl <- gl.filter.repavg(gl, threshold=0.99, v=3)
```

```
Starting gl.filter.repavg: Filtering on repeatability
Note: RepAvg is a DArT statistic reporting repeatability averaged across
alleles for each locus.

Initial no. of loci = 255
  Removing loci with RepAvg < 0.99
No. of loci deleted = 14
Summary of filtered dataset
  Repeatability >= 0.99
  No. of loci: 241
```

```
  No. of individuals: 250
  No. of populations:  30

gl.filter.repavg completed
```

Only 14 loci out of 255 were deleted.

Exercises

1. Just in case you have accidentally modified the genlight object `gl`, recreate it by copying it from `testset.gl`.
2. Filter `gl` using the filters listed above. Request a report first to inform your choice of threshold. Be sure to set plot=TRUE to examine the distribution of each parameter and optionally smearplot=TRUE to examine the smear plot.

Recalculating locus metadata after filtering

Remember, the locus metrics are no longer valid if individuals or populations are deleted from the dataset. For example, if you filter out a population for which the individuals have particularly bad call rates, then the call rate parameter held in the locus metrics will no longer be accurate. It will need to be recalculated. This is true of many of the locus metrics.

So, after filtering your data, it is wise to recalculate the locus metrics with

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

Try this for yourself on genlight object `gl` after filtering or on your own data

Similarly, when filtering has resulted in removal of some individuals or populations, variation at several loci may be lost. Some loci may even be scored as missing across all individuals. You may wish to remove these monomorphic loci from your dataset with

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

Try this for yourself on genlight object `gl` after filtering or on your own data

Where have we come?

The above Session was designed to give you an overview of the scripts in dartR for filtering your data. Having completed this Session, you should now able to:

- Filter on call rate, repeatability, secondaries, hamming distance, and minor allele frequency.
- Recalculate locus metrics after deleting individuals or populations as part of the filtering process.
- Filter out resultant monomorphic loci.

Further reading

Jombart T. and Caitlin Collins, C. (2015). Analysing genome-wide SNP data using adegenet 2.0.0. http://adegenet.r-forge.r-project.org/files/tutorial-genomics.pdf

Gruber, B., Unmack, P.J., Berry, O. and Georges, A. 2018. dartR: an R package to facilitate analysis of SNP data generated from reduced representation genome sequencing. Molecular Ecology Resources, 18:691–699

Session 7: Visualization

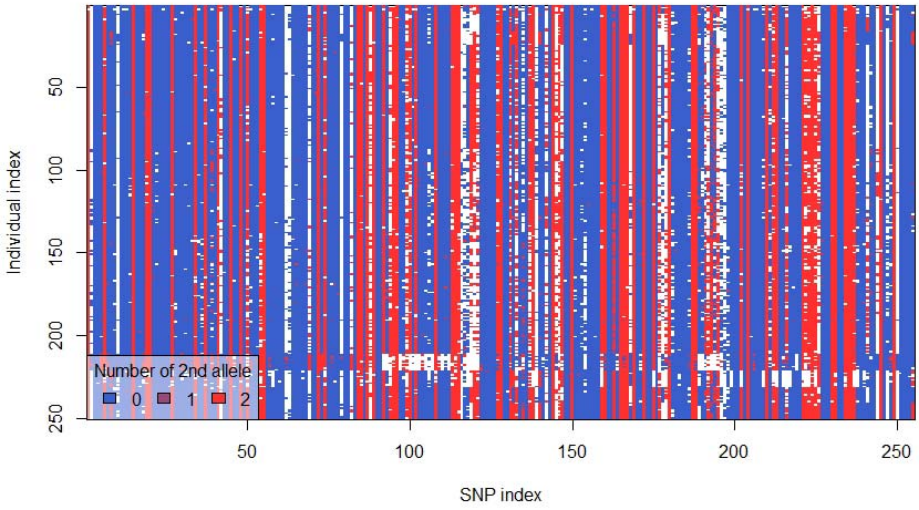
Overview

Exploring your data before getting into more serious analyses is considered by many to be an essential step in any analysis, because it allows you to recognise novel opportunities for analysis presented by your data. Exploration also builds familiarity with your data that allows you to notice when errors creep in. Visualization tools are valuable for exploratory analysis.

We have already demonstrated the value of a smear plot, generated with

`gl.plot(gl)` display a smear plot, optionally with labels for individuals (useful if not too many)

for showing the spread of scores across loci for each individual.

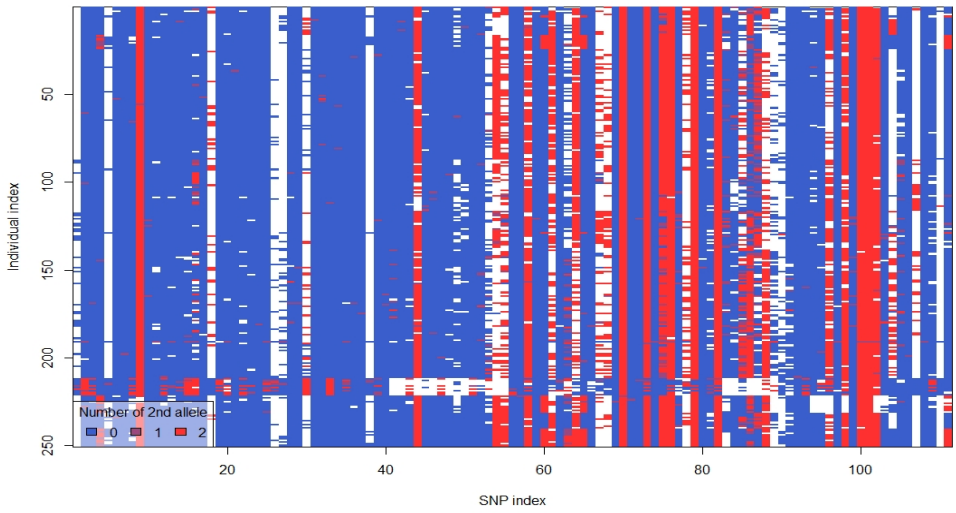


This plot is quite informative, as it shows a large number of failed calls by locus (vertical white streaking) and by individual (horizontal white streaking). Uniform blue or red vertical streaking is indicative of monomorphic loci.

Try it for yourself on genlight object `gl<-testset.gl` or on your own data

Let's see what happens when we filter for monomorphic loci.

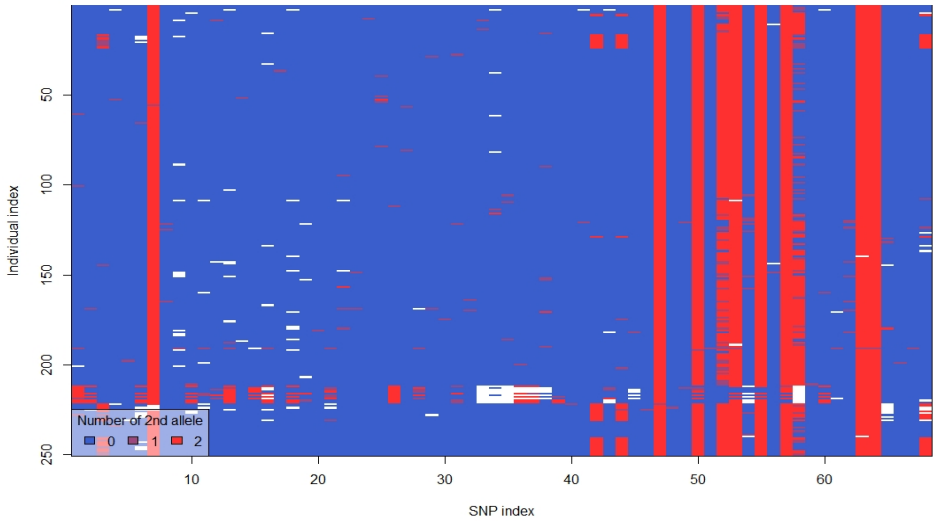
```
gl <- gl.filter.monomorphs(gl)
gl.plot(gl)
```



Try it for yourself on genlight object `gl` or on your own data

You can see that the smear plot is a bit tidier, but there is still the call rate issue to deal with.

```
gl <- gl.filter.callrate(gl,threshold=0.95)
gl.plot(gl)
```

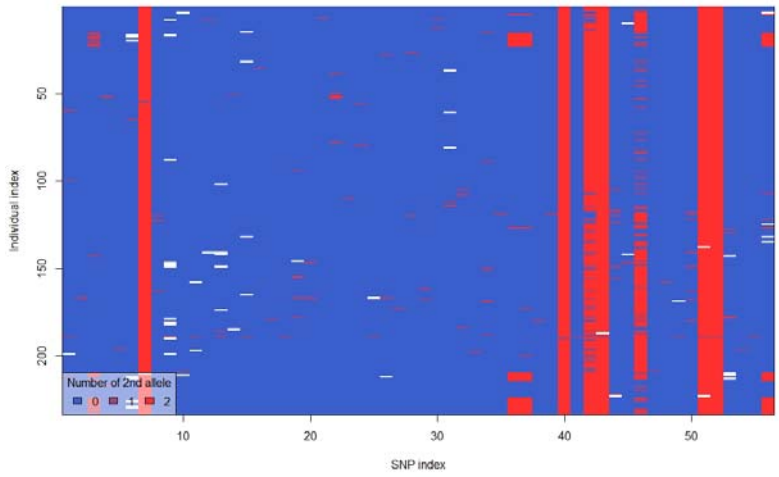


Much better, but there are still some issues with call rate for individuals.

```
gl <- gl.filter.callrate(gl, method="ind", threshold=0.95)
gl.plot(gl)
```

```
Starting gl.filter.callrate: Filtering on Call Rate
  Removing individuals based on Call Rate, threshold = 0.95
  Individuals deleted (CallRate <=  0.95 :
AA032760[EmmacMDBMaci], UC_00132[EmmacClarYate], UC_00267[EmvicVictJasp],
UC_00205[EmsubRopeMata], UC_00206[EmsubRopeMata], UC_00208[EmsubRopeMata],
UC_00243[EmvicVictJasp], UC_00209[EmsubRopeMata], UC_00254[EmvicVictJasp],
```

```
UC_00210[EmsubRopeMata], UC_00259[EmvicVictJasp], UC_00126c[EmvicVictJasp],
AA063718[EmmacCoopAvin], AA063722[EmmacCoopAvin], AA063726[EmmacCoopAvin],
AA063708[EmmacCoopAvin], AA063716[EmmacCoopAvin]

Note: Locus metrics not recalculated
```



That has worked well, but we have probably filtered too heavily as the "outgroup taxa" `Emvic` and `Emsub` have been removed. This is because individuals belonging to more distantly related taxa are likely to have higher allelic drop-out rates than closely related taxa. So it does well not to over-filter and certainly to take into account the level of divergence among populations.

You can see from the above examples how the smear plot can be used to guide your filtering, in addition to the gl.report functions.

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.

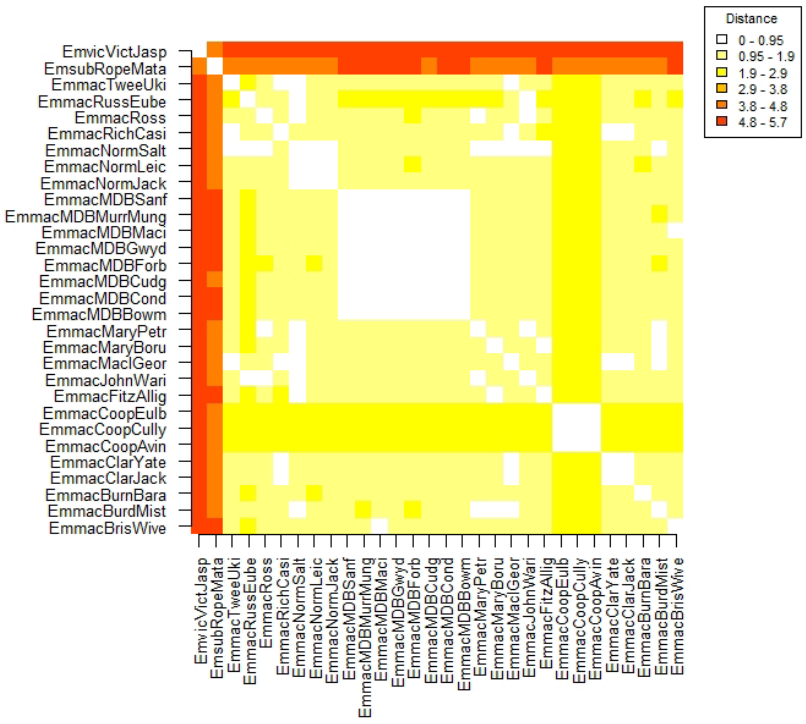
There are a very many measures of genetic distance, but they collapse in number when applied to bi-allelic SNP data. For example, Rogers D and Euclidean D differ only by a constant multiplier. Distance matrices can be generated by a number of R functionss, the most popular of which are the functions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

The function `gl.dist.pop` will also calculate a distance matrix based on fixed allelic differences and a count of private alleles.

```
D <- gl.dist.pop(gl,method="euclidean")
```

which can be visualized as a heatmap.

```
gl.heatmap(D, labels=TRUE, label.cex=0.8, values=FALSE)
```



Immediately evident is the low level of difference among populations of the Murray-Darling Basin (MDB) and the high level of divergence between the outgroup populations (*E. victoriae* and *E. subglobosa*) compared to the remaining populations.

1. Just in case you have accidentally modified the genlight object `gl`, recreate it by copying it from `testset.gl`.
2. Use ?gl.dist.pop to see the range of distance measures available.
3. Calculate a range of genetic distance measures between populations.
4. Represent them as a heatmap.

Principal Coordinates Analysis

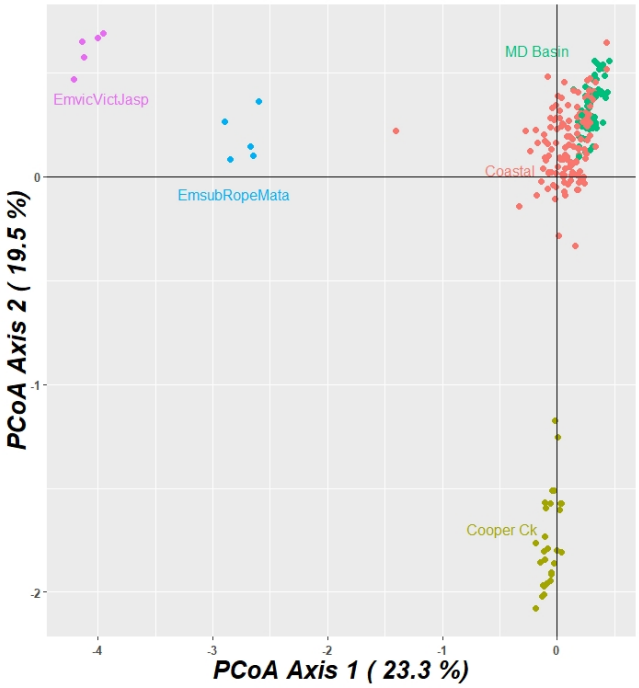
Genetic similarity of individuals and populations can be visualized by way of Principal Coordinates Analysis (PCoA) ordination (Gower, 1966). Individuals (entities) are represented in a space defined by loci (attributes) with the position along each locus axis determined by genotype (0 for homozygous 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 (one per locus) 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, PCoA-3 is

orthogonal to PCoA-1 and 2 and explains most of the residual variation after fitting PCoA-1 and 2 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.The relevant scripts are:

`pcoa <- gl.pcoa(gl)`	conduct the principal coordinates analysis
`gl.pcoa.scree(pcoa)`	plot eigenvalues for leading PCoA axes to enable assessment of the number of ordinated dimensions to examine
`gl.pcoa.plot(pcoa, gl)`	plot the individuals in the space defined by two specified PCoA axes
`gl.pcoa.plot.3d(pcoa,gl)`	plot the individuals in the space defined by three specified axes, and allow mouseover rotation

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}.



This PCoA plot shows the first two dimensions in the ordination. It is important to note that together, they explain 42.8% of the variation present in the data. So there is 57.2% of variation not visible in this representation. Emmac_Coast, MDB and Cooper populations form a cluster, but what if we examined deeper dimensions? Might not they separate out? Clearly there is room here for misinterpretation.

Exercise: Reproduce the Principal Coordinates Analysis above.

1. Before we start, check the number of populations.

```
popNames(gl)
```

2. It is sensible to reduce the number of populations for plotting so that the plot is not cluttered with labels.
3. Merge populations using gl.merge.pop (best to cut and paste):

```
gl <- gl.merge.pop(gl,old=(c("EmmacBrisWive",
"EmmacBurdMist", "EmmacBurnBara",
"EmmacClarJack", "EmmacClarYate",
"EmmacFitzAllig", "EmmacJohnWari",
"EmmacMaclGeor", "EmmacMaryBoru",
"EmmacMaryPetr", "EmmacNormJack",
"EmmacNormLeic", "EmmacNormSalt",
"EmmacRichCasi", "EmmacRoss", "EmmacRussEube",
"EmmacTweeUki")), new="Coastal")

gl <- gl.merge.pop(gl,old=(c("EmmacCoopAvin",
"EmmacCoopCully", "EmmacCoopEulb")),
new="Cooper Ck")

gl <- gl.merge.pop(gl,old=(c("EmmacMDBBowm",
"EmmacMDBCond", "EmmacMDBCudg", "EmmacMDBForb",
"EmmacMDBGwyd", "EmmacMDBMaci",
"EmmacMDBMurrMung", "EmmacMDBSanf")), new="MD
Basin")
```

4. Check that the job is done.

```
popNames(gl)
```

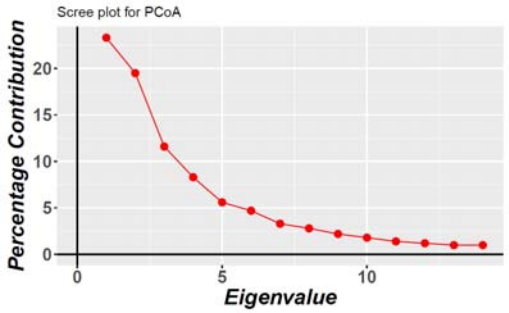
5. Run the Principal Coordinates Analysis

```
pcoa <- gl.pcoa(gl)
```

and plot the results

```
gl.pcoa.plot(pcoa, gl)
```

How many dimensions?



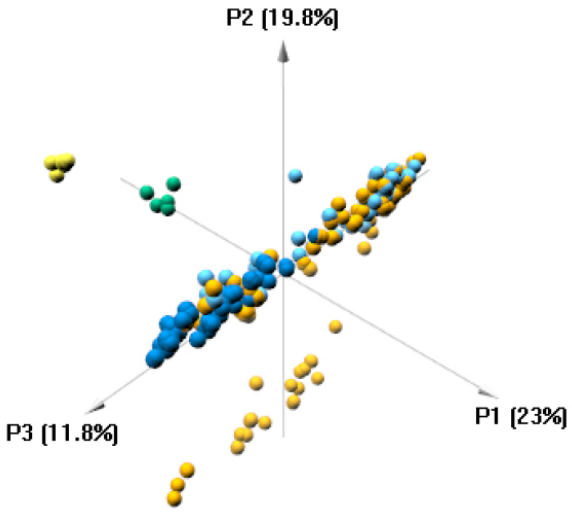
One way to decide how many dimensions to examine is a scree plot. A scree plot shows the percentage variation explained by each axis in the ordination successively. The script `gl.pcoa.scree()`, by default, will show the percentage variation in the data explained by each axis where the amount of variation 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.

In this case, at least three dimensions should be examined to capture most of the variation present in the dataset.

Try it for yourself on genlight object `testset.gl` or on your own PCoA analysis.

The 3D plot

Two dimensions is fine, three dimensions is also easy to visualize. For this we use the script `gl.pcoa.plot.3d(pcoa,gl)`. This function is essentially a wrapper for the corresponding function in {pca3d}, adding percentage variation explained to each axis and fixing some parameters.



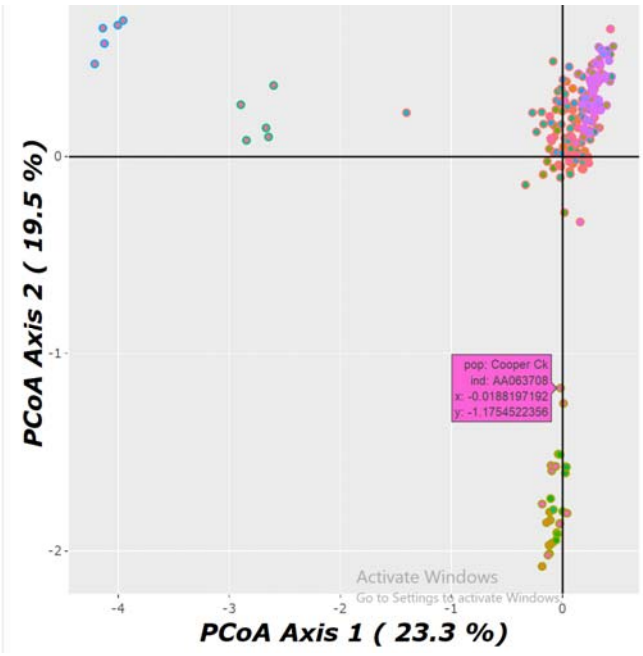
The plot is interactive in the sense that you can rotate the plot using the mouse to obtain the most discriminatory view.

For more than three dimensions, when the data cluster tightly, additional dimensions can be visualized by removing all individuals from the analysis except those belonging to a single cluster and re-running the PCoA (Georges and Adams, 1992).

Try a 3D plot for yourself on the PCoA dataset generated from `testset.gl` or on your own PCoA analysis.

Identifying individuals in the plot

One of the annoyances of a PCoA plot is seeing that one recalcitrant outlier, and not knowing which one individual it is. You can identify individuals within a PCoA by re-plotting with the option `labels="interactive"` then running `ggplotly()`. The new plot allows you to mouse-ever the points whereby the identity of the individual will be shown.



In the above plot, moving the mouse over the point reveals animal AA083708, from the Cooper Creek populations.

Try an interactive plot for yourself on the PCoA dataset generated from `testset.gl`. What is the identity of that specimen curiously intermediate between the MDB/Coastal populations and the outgroup taxa?

Exercises

1. Examine the recode table `recode_to_regions.csv`.
2. Create a genlight data set `gl` from `testset.gl` by applying the recode table.
3. Conduct a principal coordinates analysis.
4. Examine the scree plot – how many dimensions in the PCoA would you consider informative?
5. Plot the data using pairs of informative axes, starting with the two most informative.
6. Plot again the two most informative axes, but this time with the option `labels="interactive"`. Run `ggplotly()` to enable mouse-over to identify points.
7. Generate a three-dimensional plot using the most informative axes, and then undertake a rotation of the 3D solution to display any groupings most clearly.

Where have we come?

The above Session was designed to give you an overview of the scripts in dartR for viusalizing your data. Having completed this Session, you should now able to:

- Generate smearplots, heatmaps and PCoA plots to look for structure in your data graphically.
- Obtain a good understanding of what to expect when you run more sophisticated analyses to answer specific questions.

.

Session 8: Moving from dartR to other packages

Overview

The preparatory stage in your analysis is now complete, and you are ready to move on to more sophisticated analyses using a range of software other than dartR.

Package dartR does not pretend to provide a comprehensive range of analyses, but rather to provide avenues to convert from SNP data stored as a genlight object to other available software packages. Indeed, you will find that most of your work in dartR will be data curation and checking, filtering and then output for analysis by other packages.

Conversion scripts

To assist with connecting to other packages, the following conversion functions are available:

`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).
`gl2nhyb()`	outputs 200 loci selected according to user specified criteria for input to the package NewHybrids (Anderson and Thompson, 2002).
`gl2snapp()`	converts a genlight object to nexus format suitable for phylogenetic analysis by SNAPP (via BEAUti)
`gl2svdquartets()`	converts a genlight object to nexus format for used by PAUP SVDquartets (Chifman and Kubatko, 2014)
`gl2phylip()`	creates a distance matrix (or set of distance matricies for bootstrapping) from a genlight object for input to Phylip (Felsenstein, 1989)
`gl2treemix()`	Convert a genlight object to a treemix input file (Pickrell and Pritchard, 2012)
`gl2faststructure()`	converts a genlight object to faststructure format (Raj et al., 2014)
`gl2gds()`	converts a genlight object to format suitable for SNPRelate (gds format) (Zheng et al. 2012)

`gl2shp()`	exports cordinates in a genlight object to a point shape or kml file.
`gl2gi()`	converts a genlight object to a genind object as defined by the {adegenet} package
`gi2gl()`	converts a genind object as defined by the {adegenet} package to a genlight object
`gl2plink()`	exports a genlight object as defined by the {adegenet} package to an plink file

Exercises

1. Convert the genlight object testset.gl to a fastA file. Examine it.
2. Generate an input file for SVDQuartets from testset.gl. Examine it.

References

Anderson, E. C., & Thompson, E. A. (2002). A model-based method for identifying species hybrids using multilocus genetic data. Genetics, 160, 1217–1229.

Chifman, J. and L. Kubatko. 2014. Quartet inference from SNP data under the coalescent, Bioinformatics, 30: 3317-3324

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

Pickrell and Pritchard (2012). Inference of population splits and mixtures from genome-wide allele frequency data. PLoS Genetics https://doi.org/10.1371/journal.pgen.1002967

Pritchard, J.K., Stephens, M. and Donnelly, P. (2000). Inference of population structure using multilocus genotype data. Genetics 155:945-959.

Raj, A., Stephens, M. and Pritchard, J.K. (2014). fastSTRUCTURE: Variational Inference of Population Structure in Large SNP Data Sets. Genetics 197:573-589.

Zheng X, Levine D, Shen J, Gogarten SM, Laurie C, Weir BS. (2012). A High-performance Computing Toolset for Relatedness and Principal Component Analysis of SNP Data. Bioinformatics. 28:3326-3328.

Ende