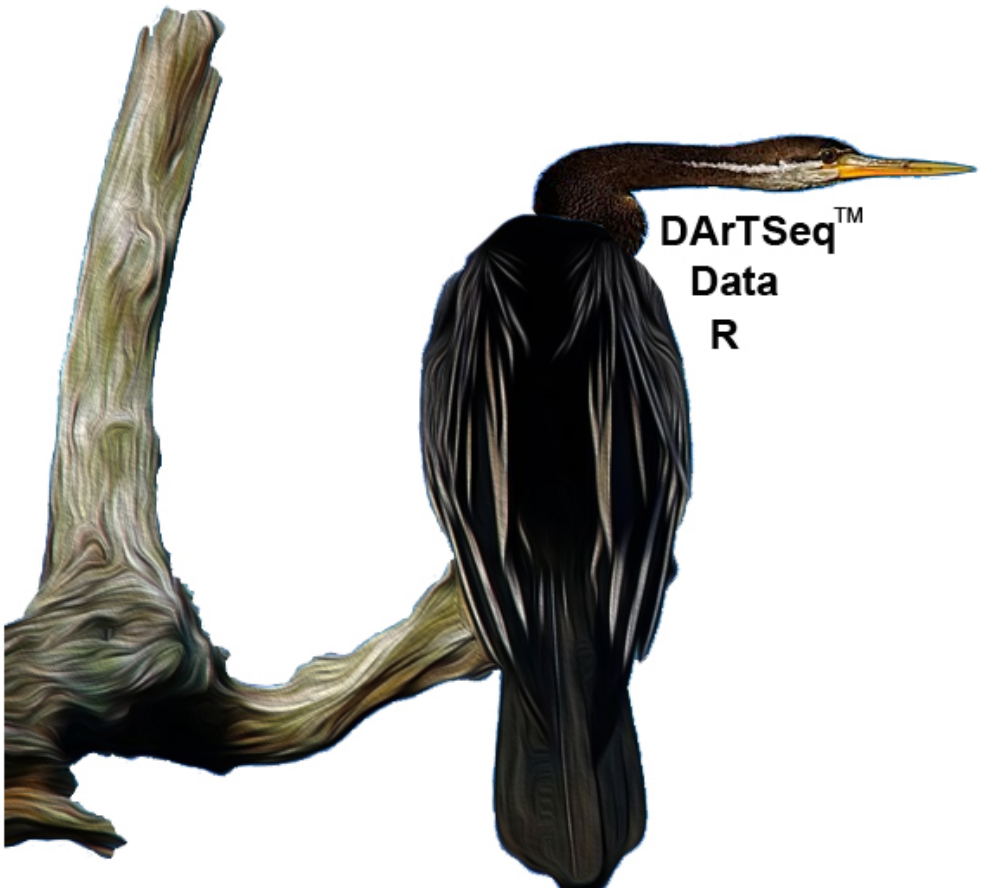


A Workshop on Analysing DArT PL SNP data using R Package dartR

DRAFT

(not for circulation)

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

What is R?

R is a statistical computing language based on an earlier implementation of a programming language called S. S is still available in the commercial form of S-plus, whereas R is in the public domain.

R was created by Ross Ihaka and Robert Gentleman (hence the name R) at the University of Auckland, New Zealand, and is now developed by the R Development Core Team.

Many statistical packages on the market, such as SAS, SPSS and Statistica are regarded as fourth generation statistical programming languages. The R programming language is a hybrid between a third generation language such as C or FORTRAN and a fourth generation language such as SAS. This provides for much greater flexibility for the analyist, but demands much more in terms of programming skills.

R supports a wide variety of statistical and numerical techniques, with comparable benchmark results to Octave and its proprietary counterpart MATLAB. R is also provides the analyst with a very wide range of packages, which are user-submitted program **libraries**, for specific functions or specific areas of study. As a result, R is one of the most comprehensive statistical analysis systems on the market. R has exceptionally good graphical capacity, and can be used to produce publication-quality graphs.

The library we will be primarily using in this workshop is **dartR**, a collection of scripts to facilitate analysis of data provided by Diversity Arrays Technology Pty Ltd. Although dartR has some unique analyses, it is primarily for data manipulation, exploratory analysis, and a conduit to other packages used for SNP analysis.

The versatility of R has led to many different styles in the way the program is used. A programmer will use R in a very different way from someone using R to undertake statistical analyses. In this workshop, you will require familiarity with the R GUI, RSTudio, but will not necessarily need to be well versed in R programming.

Lesson 1: The RStudio Operating Environment

The Command Line Interface

Rather than using the standard distribution of R, we will be using an implementation with a graphical user interface called R-studio.

When you first start R-studio, a graphical user interface opens with many features to assist you (Figure 1-1). After some introductory text appears in the Console, a **Command Prompt** is presented (>), and the system awaits instructions.

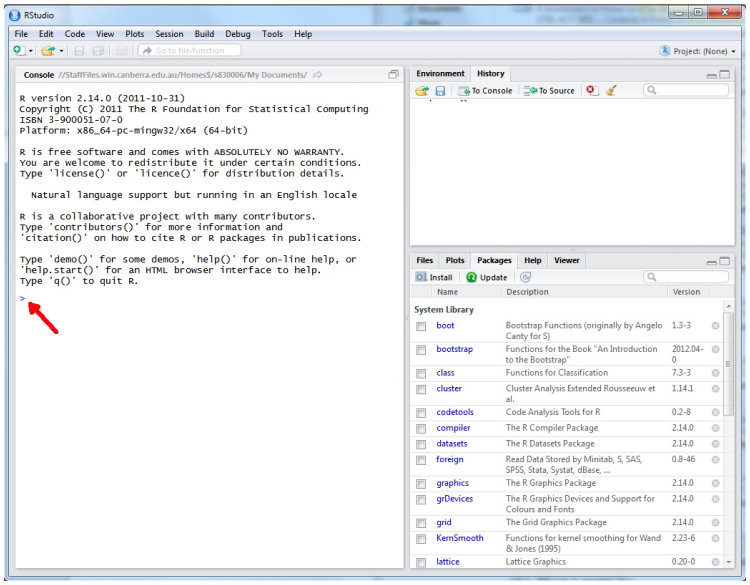


Figure 1-1. R as it appears when it first starts. The R Console window and two other windows are visible. The Program Editor and R Graphics windows do not appear until required.

Before we move on, there are a couple of little tricks here that are worth mentioning. The first is that the Console can be cleared of text using control-L (^l). The second tip is that the up-arrow will recall previously submitted commands, which will save you a lot of typing. Try these as you go along.

The simplest way of using R is to supply instructions to this command prompt.

```
> beetles <- c(15.2,12.1,17.8,13.9,16.4,15.1)
```

There is a lot to this simple command. What we are doing here is creating a list of values, referred to as a **vector** in R terminology. In this case, the data are lengths of beetle elytra. The concatenate function c() is used to create the vector which is then assigned to the **object** `beetles` using the **assignment operator** `<-`. The object `beetles` is called an object because it is a self-contained entity that can be used in subsequent calculations.

Instructions to the command line are terminated with a return (↵) or a semi-colon (;). R instructions are case sensitive, so the objects `beetles`, `Beetles` and `BEETLES` are all considered as separate objects. It is wise to adopt a consistent practice, such as always using lower case unless upper case is demanded by the R syntax.

Spaces matter, sometimes. So you will need to watch that.

If you instruct R to undertake some action, and do not assign it to an object, then R will direct the results of the instructions to the screen. For example, requesting

R to create the vector of beetle elytra without assigning it to `beetles` will result in the vector being listed on the screen.

```
> c(15.2,12.1,17.8,13.9,16.4,15.1)
[1] 15.2 12.1 17.8 13.9 16.4 15.1
```

You can view the contents of an object simply by giving its name in response to the command prompt.

```
> beetles
[1] 15.2 12.1 17.8 13.9 16.4 15.1
```

As an object, `beetles` can be used in subsequent calculations. For example,

```
> mean(beetles)
[1] 15.08333
```

R programs often comprise a series of nested instructions, and the same result could have been obtained by using

```
> mean(c(15.2,12.1,17.8,13.9,16.4,15.1))
[1] 15.08333
```

This is the advantage of an object-oriented approach to programming.

The Graphics Interface

When a command requires more sophisticated output, R will open a purpose-built window. The most useful of these is the graphics window.

A scatter plot of 1000 pairs of coordinates drawn at random from a bivariate standard normal distribution (mean=0, stdev=1) is made by combining the `plot()` function with the `rnorm()` function as follows:

```
> plot(rnorm(1000,0,1),rnorm(1000,0,1))
```

The result is shown in in Figure 1-2. This scatter plot can be saved to a file or copied to the clipboard by right-clicking on the graphics window and choosing the desired outcome.

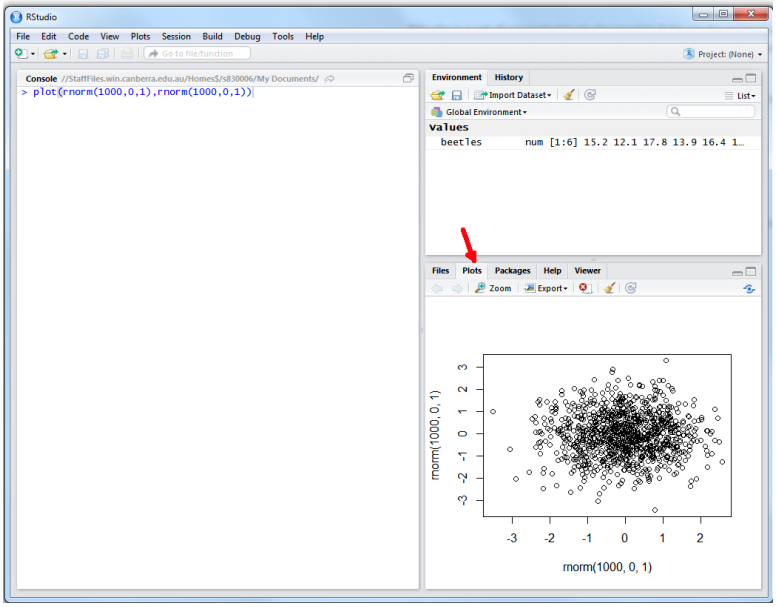


Figure 1-2. R as it appears after activating the graphics window.

You can pull the graphics out into its own window with the [Zoom] tab, or export the image in one of the standard formats using the [Export] tab.

The R Editor Interface

Using the Command Line Interface is great for a quick analysis, but it is essentially a calculator mode. Once you have done the calculations, you walk away only with the results. In more substantial analyses, we need to better manage the set of programming instructions needed to do the job. We do this using the **R editor**, which can be accessed from the file menu [File>New File>R Script] or by typing control-N (^N).

The idea is to type all instructions in the R editor for progressive submission or for submission as a block. At the end of the process, you have a complete program listing that can be saved to disk for later use.

The R editor is not all that sophisticated. Each instruction is typed in on its own line. A line can be submitted for execution by placing the cursor on it and typing control-enter (^↵). Alternatively, blocks of instructions can be highlighted and submitted in the same way. This allows progressive debugging of the program as it is constructed.

It is wise to include abundant comments as part of your programs, so that you can understand them later or pass them to others in a comprehensible form. Comments are preceeded by the # character and terminated by a return (↵).

Our simple R program, with comments added is shown in Figure 1-3.

```
# Creating and displaying a list of beetle measures
beetles <- c(15.2,12.1,17.8,13.9,16.4,15.1)
beetles
mean(beetles)
```

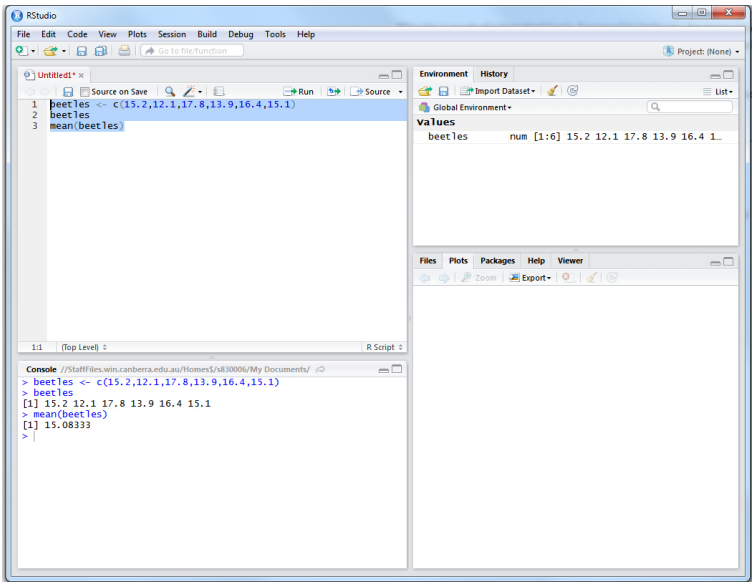


Figure 1-3. R as it appears after submitting a simple program.

Accessing Packages

Not many users of R program all the scripts that they require to undertake a task. This would be like reinventing the wheel. Instead, it is possible to access scripts written by those who have come before you, and who have made those scripts available as a package. A complete list of available packages can be obtained from the *Comprehensive R Archive Network* known as CRAN (**https://cran.r-project.org/**).

You will generally identify the packages you require after some investigative work on the web, by talking to colleagues or taking pointers from the literature. For example, the package reshape2 is useful for rearranging data, and can be accessed using the R-studio menus (Packages>Install Packages) or using the statement

```
> install.packages("reshape2", lib="C:/Program Files/R/R-3.2.1/library")
```

You may need administrator rights to install packages. Packages are only installed once, not every time you require them.

The package we will be using a lot in this workshop is dartR.

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

The directory where packages are stored is called the library. R comes with a standard set of packages. A list of installed packages can be obtained using

```
> library()
```

or by examining the list using the Packages menu.

Apart from those included in the standard implementation of R, installed packages are loaded for use in a session with the library function.

```
> library(dartR)
```

A list of loaded packages is obtained with

```
> search()
```

Error Handling and Help

When you make an error in the syntax of commands given to R, the program will respond with some form of diagnostic message. Sometimes these are self-explanatory, sometimes they are not.

Help files

Fortunately, R has very extensive help documentation. If you know the exact name of the function you want help on (e.g. sort), help can be obtained using

```
> ?gl.filter.callrate
```

More extensive help can be obtained from the help files. These are accessed via the [Help] tab of R-studio. Here you can use the Search Engine and Keywords link to access a wide range of information on the operations of R.

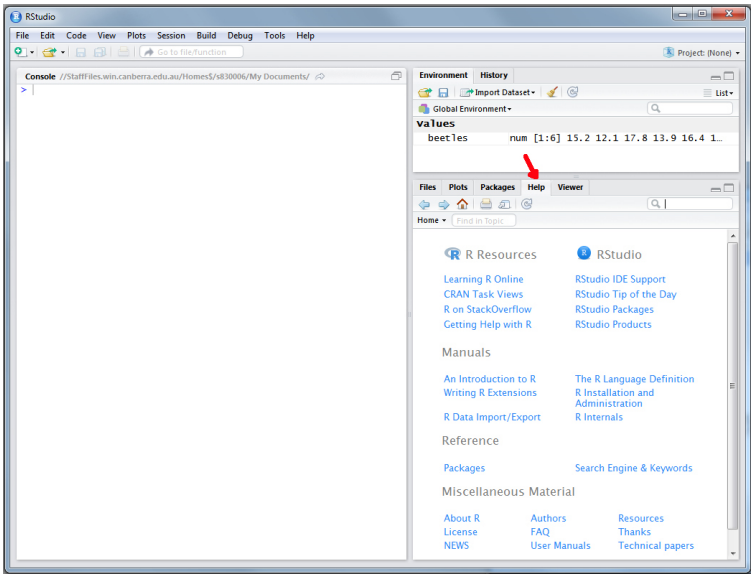


Figure 1-4. Useful information available using the [Help] tab.

Vignettes

Some packages in R have what are called **vignettes**. These are how-to guides for topics, and usually offer gentle introductions and examples. Alternatively, you can view vignettes from any loaded package by going to the 'Vignettes' menu and selecting the required package name. This will give a list of all available vignettes for you to open. Sometimes this menu doesn't appear until you load a package which has a vignette.

The Web

The web and Google are good places to turn for assistance. An excellent quick reference to R can be found on http://www.statmethods.net/

Managing Your Workspace

Starting a New Session

R facilitates the management of workflow by defining a **workspace** to hold all of your objects – vectors, dataframes, user-defined functions and the like. A workspace can be saved at the end of a session, and reloaded at a later time when you want to continue the analysis.

Managing workflow can be difficult in R, and we need some basic rules to minimize confusion.

- Identify discrete projects or analyses and create a separate Windows directory for each one. This way you will avoid having a jumble of objects from many analyses in your workspace.
- Tidy up after each session, by removing all unwanted and temporary objects, before saving your workspace.
- Use standard file naming conventions, such as filename.R for R programs and filename.dat for raw data files.

Once you have started R, you need to start a new project using File>New Project. R will prompt you for a directory in which to save all temporary and working files, and the project image if you choose to save it later.

R may ask you to save existing work before opening a new project. You should do this if you have important work that has been executed in a previously open project.

Resuming a Session

To open an existing project use File>Open Project. This will prompt you for the working directory, scan it for a previously saved workspace, load that workspace if it exists and pass control back to you via the command prompt.

Either way, with an existing or a new project, you can now begin a new analysis, or continue an existing analysis confident that the objects, datafiles, and other associated elements of your analysis are a self-contained unit.

Terminating a Session

Exit a session by exiting from R, at which time you will be asked whether or not you wish to save your workspace.

Managing your Objects

The active objects associated with your workspace are listed when you select the [Global Environment] tab in R-studio.

In addition, there are a number of useful functions for managing your workspace.

`> setwd("c://R_analysis")`	Sets the default directory for files, and action that can also be done from the R-studio menus.
`> ls()`	provides a list of objects in your current workspace.
`> rm(`*`object`*`)`	deletes an object from your workspace.
`> rm(list=ls())`	deletes all objects from your workspace.
`> sessionInfo()`	provides information about your session.

Setting a default directory

The best way to manage your work is to ensure that the files associated with each project is in a separate directory on disk. To set the default directory use

```
> setwd("C:/Users/username/Documents/R_demo")
```

R will then look in the directory R_demo when locating a file to read, and to write a file. Note the direction of the backslashes in the file specification.

Where have we come?

The above lesson was designed to give you an overview of the operation of R through the RStudio graphical user interface. Having completed this lesson, you should now be familiar the following concepts.

- R has available a Graphical User Interface (GUI) called R-studio, and within it, the R Console, R Editor Window, Graphics Output Window, and various Help Windows.
- R establishes a workspace. Managing the objects in that workspace is challenging for the new user of R, but proficiency will come with practice.

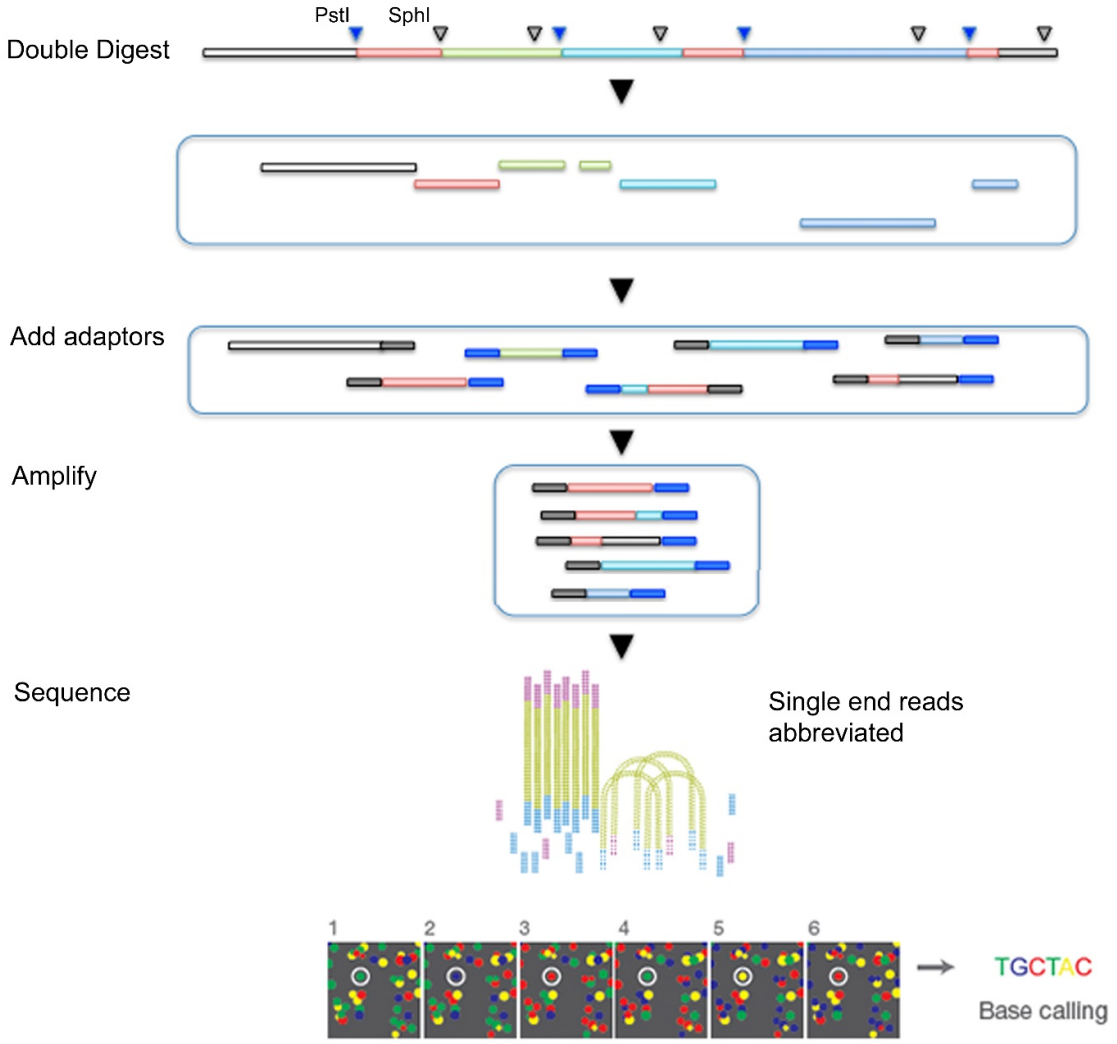
Lesson 2: Introduction to DArTSeq

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.

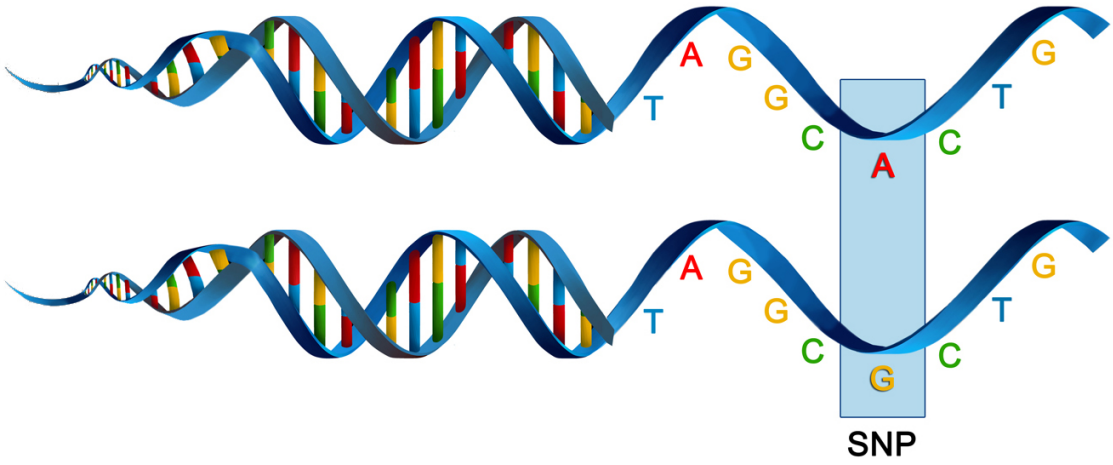
In particular, one third of samples are processed twice as technical replicates, from DNA and using independent adaptors, through to allelic calls. Scoring consistency (repeatability) is used as the main selection criterion for high quality/low error rate markers.

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 PL 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. A genotype can this be coded 0 for aa homozygotes, 1 for ab heterozygotes, and 2 for bb homozygotes.

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.

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 lesson was designed to give you a very brief overview to the Diversity Arrays Technology pipelines for producing SNP and associated data. Having completed this lesson, 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.

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.

Lesson 3: Getting data into dartR

Installing dartR

Installing from CRAN

As dartR is now a fully fledged package, simply type:

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

then load it with

```
library(dartR)
```

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

Installing from GitHub

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

Again, 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)
source("http://bioconductor.org/biocLite.R")
biocLite("qvalue", suppressUpdates=T)
biocLite("SNPRelate", suppressUpdates=T)
install_github("green-striped-gecko/dartR")
library(dartR)
```

Additional information can be found at the github pages:
https://github.com/green-striped-gecko/dartR

Set a Working Directory

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

```
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).

Now you are ready to go.

A sensible pipeline

Let us begin by defining a sensible pipeline for entering your data, to provide a context for the material in the remainder of this lesson and the next.

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 `CloneID` column, and the locus metadata needs to finish with the column `RepAvg`. The row with the 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 for future use. This will be your point of access to the data in future analyses.

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:

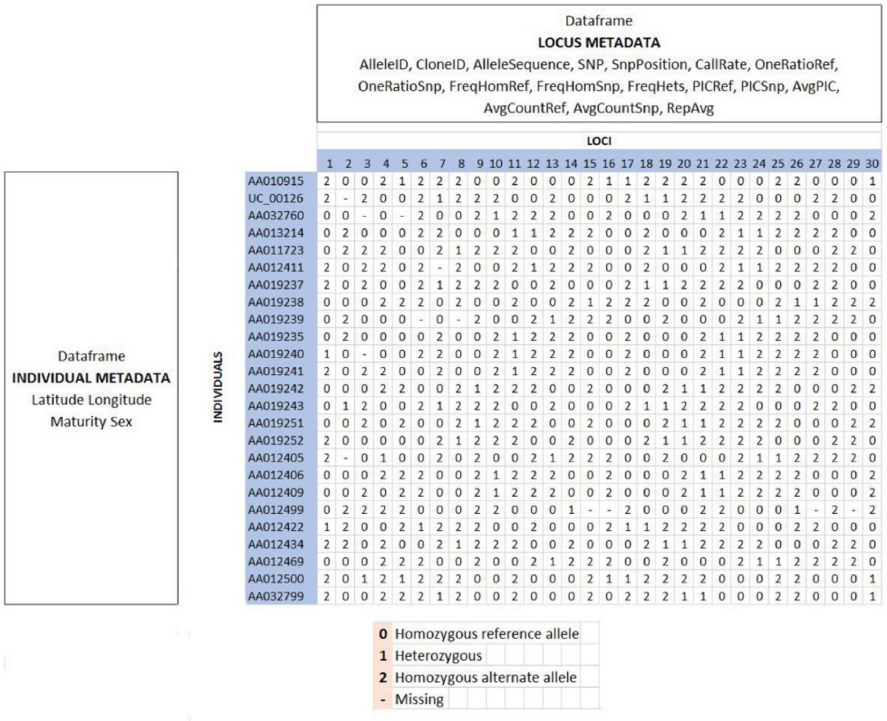
SNP — the mutational change and its position in the sequence tag, referenced from zero

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



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
AA010915	2	0	0	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2	2	0	0	0	2	2	0	0	0	1
UC_00126	2	-	2	0	0	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2	2	0	0	0	2	2	0	0
AA032760	0	0	-	0	-	2	0	0	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2	2	0	0	0	2
AA013214	0	2	0	0	0	2	2	0	0	0	1	1	2	2	2	0	0	2	0	0	0	0	2	1	1	2	2	2	2	0	0
AA011723	0	2	2	2	0	0	2	1	2	2	2	0	0	2	0	0	0	0	2	1	1	2	2	2	2	0	0	0	2	2	0
AA012411	2	0	2	2	0	2	-	2	0	0	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2	2	0	0
AA019237	2	0	2	0	0	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2	2	0	0	0	0	2	2	0	0
AA019238	0	0	0	2	2	2	0	2	0	0	2	0	0	0	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2
AA019239	0	2	0	0	0	-	0	-	2	0	0	2	1	2	2	2	0	0	2	0	0	0	0	2	1	1	2	2	2	2	0
AA019235	0	2	0	0	0	0	2	0	0	2	1	2	2	2	0	0	2	0	0	0	0	2	1	1	2	2	2	2	0	0	0
AA019240	1	0	-	0	0	2	2	0	0	2	1	2	2	2	0	0	2	0	0	0	0	2	1	1	2	2	2	2	0	0	0
AA019241	2	0	2	2	0	0	2	0	0	2	1	2	2	2	0	0	2	0	0	0	0	2	1	1	2	2	2	2	0	0	0
AA019242	0	0	0	2	2	0	0	2	1	2	2	2	0	0	2	0	0	0	0	2	1	1	2	2	2	2	0	0	0	2	2
AA019243	0	1	2	0	0	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2	2	0	0	0	2	2	0	0	0
AA019251	0	0	2	0	2	0	0	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2	2	2	0	0	0	2	2
AA019252	2	0	0	0	0	0	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2	2	2	0	0	0	2	2	0
AA012405	2	-	0	1	0	0	2	0	2	0	0	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2	2	0	0
AA012406	0	0	0	2	2	2	0	0	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2	2	0	0	0	2	2
AA012409	0	0	2	0	2	2	0	0	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2	2	0	0	0	2	2
AA012499	0	2	2	2	2	0	0	0	2	2	0	0	0	1	-	-	2	0	0	0	2	2	0	0	0	1	-	2	-	2	2
AA012422	1	2	0	0	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2	2	0	0	0	2	2	0	0	0	0
AA012434	2	2	0	2	0	0	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2	2	0	0	0	2	2	0	0
AA012469	0	0	0	2	2	2	0	0	2	0	0	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2	2	0	0
AA012500	2	0	1	2	1	2	2	2	0	0	2	0	0	0	2	1	1	2	2	2	2	0	0	0	2	2	0	0	0	1	1
AA032799	2	0	0	2	2	2	1	2	0	0	2	0	0	0	2	0	2	2	2	1	1	0	0	0	2	2	0	0	0	1	1

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"
```

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

These metadata are used by the {dartR} package for various purposes, so if any are missing from your dataset, then there will be some analyses that will not be possible. For example, TrimmedSequence is used to generate output for subsequent phylogenetic analyses that require estimates of base frequencies and transition and transversion ratios.

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

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:

`gl@pop` or `pop(gl)` and `popNames(gl)`

and

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

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

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.

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 shown
  gl@other$ind.metrics$sex[1:10]
```

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

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

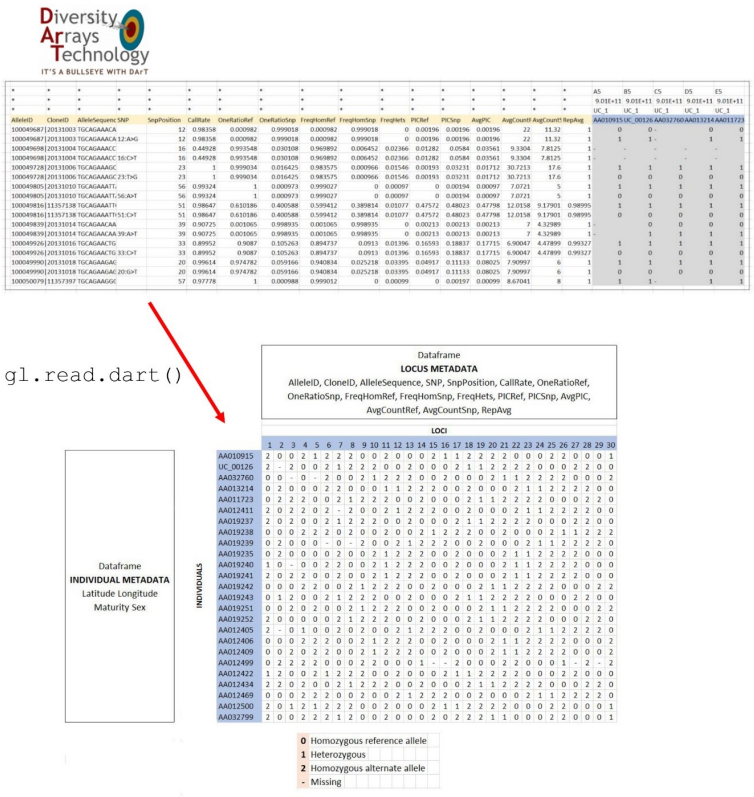
- if the file is a 1-row or 2-row format, as supplied by Diversity Arrays Technology Pty Ltd.

- the number of locus metadata columns to be input (the first typically being `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="
       DCra15_Chdes_SNP_2row.csv",
       ind.metafile="cherax_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, etc).



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

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="gl.Rdata")
```

and then read it in again with

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

Exercises

Open the file `DCra15_Chdes_SNP_2row.csv` in Excel. This is a set of SNP data for *Cherax destructor* provided in 2-row format by DArT PL.

1. Note that the data comprise the individuals/samples/specimens as columns and the loci as rows.

   Each locus is represented by two rows, one corresponding to the scores for the reference allele and one corresponding to the scores for the alternate (or SNP) allele.

   A score of 1 for the reference and 0 for the alternate indicates that the individual is homozygous for the reference allele, and this will be scored in the genlight object as a 0.

   A score of 0 for the reference and 1 for the alternate indicates that the individual is homozygous for the alternate allele, and this will be scored in the genlight object as a 2.

   A score of 1 for the reference and 1 for the alternate indicates that the individual is heterozygous, and this will be scored in the genlight object as a 1.

   Null alleles and missing data are scored as – for reference and alternate, and this will be scored as NA in the genlight object.

   Note that the data do not include – 0 or – 1, as these data will either be scored as 1 0 (0) or 0 1 (2), respectively, or scored as – – (NA) during the SNP calling process depending upon quality attributes.
2. Note the locus metadata provided in columns 1 to 23, starting with `AlleleID` and ending with `RepAvg`. The meanings of these metadata are provided in a metadata file also provided by DArT PL.
3. Note the first three lines of the file, which are ignored when dartR reads the data in. These lines are obvious because of the * * *.
4. Examine the individual metadata in the file `cherax.metadata.csv`. Note the two mandatory columns id and pop.
5. Read the SNP data in to dartR as a genlight object called `cherax.2row`.
6. Check the locus metadata that is contained in the genlight object `cherax.2row`.
7. Check the individual metadata that is contained in `cherax.2row`.
8. List the populations to which individuals have been assigned in `cherax.2row`.
9. Save the genlight data in `cherax.2row` as a binary file.

Open the file `DCra15_Chdes_SNP_1row.csv` in Excel. This is a set of SNP data for *Cherax destructor* provided in 1-row format by DArT PL.

1. Note that the data comprise the individuals/samples/specimens as columns and the loci as rows.

Each locus is represented by one row.

A score of 0 indicates that the individual is homozygous for the reference allele, and this will be scored in the genlight object as a 0.

A score of 1 indicates that the individual is homozygous for the alternate allele, and this will be scored in the genlight object as a 2.

A score of 2 indicates that the individual is heterozygous, and this will be scored in the genlight object as a 1.

Null alleles and missing data are scored as – and this will be scored as NA in the genlight object.

A bit confusing, admittedly, but the scoring scheme in the genlight object allows one to sum columns and rows to get the frequency of the alternate allele, which is very convenient.

2. Note the locus metadata provided in columns 1 to 19, starting with `CloneID` and ending with `RepAvg`. The meanings of these metadata are provided in a metadata file also provided by DArT PL.
3. Note the first five lines of the file, which are ignored when dartR reads the data in. These lines are obvious because of the * * *.

Read these data in to dartR as a genlight object called `cherax.1row`.

Open the file `DCra15_Chdes_SilicoDArT.csv` in Excel. This is a set of SilicoDArT data for *Cherax destructor* provided by DArT PL.

1. Note that the data comprise the individuals/samples/specimens as columns and the loci as rows. Each locus is represented by one row.

   A score of 0 indicates that the sequence tag represented in other individuals as having a SNP was not present in this individual, that is, the SNP was not called.

   A score of 1 indicates the SNP was called in this individual.

   A missing score is a true missing value, in that it could not be determined if the SNP was called or not.

dartR does not yet support SilicoDArT data.

Where we have come

In this lesson, 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 lesson, you should understand

- How to install published version of dartR from the CRAN repository, and how to get the latest copy from github.
- 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 DArT PL data 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**

Lesson 4: Basic Filtering and other Manipulations

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, as well as some scripts to manipulate the content of your genlight data.

Interogating a genlight object

As outlined earlier, the genlight object that holds your SNP data also contains metadata for your specimens/samples and for the scored loci. You can interrogate the genlight object (which we will for convenience refer to as `gl`) using a number of 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.
`glPlot(gl)`	a smear plot of individual against locus, useful for gross pattern identification and assessment of allelic dropout

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 exercises 2.-7. with the in-built data set `foxes.gl` (dartR Version 1.1.1)

Manipulating populations and individuals

There are a number of scripts to assist you in manipulating a genlight object.

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

`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

These scripts delete individuals, and so many of the metadata variables provided by DArT 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 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( )` recalculate locus metrics, where appropriate

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

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( )` remove monomorphic loci, including all NAs

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.

Exercises

1. Examine again the population assignments in the genlight object `gl`.
2. Create a new genlight object from `gl`, but dropping one of the populations. Confirm that the selected population has been dropped.
3. Create a new genlight object from `gl`, by merging two populations into one. Confirm that the population reassignment has been achieved.
4. 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.
5. Check again the number of loci in the genlight object `gl` and again generate a smear plot. Filter out monomorphic loci. How many loci were removed. Can you see this effect by repeating the smear plot after filtering? What change do you see?

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.

`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( )`	recalculate locus metrics

to recompute the locus metrics.

Exercises

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.

Filtering

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

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.

These filtering scripts typically have a report script to assist you with the decision on an appropriate threshold. Do not assign the output of the report to your genlight object or you will overwrite it.

`gl.report.callrate( )`	provides a summary of CallRate values
`gl.report.repavg( )`	provides a summary of 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, that have more than one sequence tag represented in the dataset.
`gl.report.maf( )`	provides a report on minor allele frequencies overall, and by population.
`gl.report.hamming( )`	provides a report on Hamming distances between sequence tags.

Exercises

1. Start by copying `testset.gl` to `gl`.
2. Check the statistics of `gl`, particularly the number of loci. Recreate the smear plot.
3. Filter `gl` on `repAvg` at a threshold of 0.99. How many loci have you lost. What effect has this had on the smear plot?
4. Now filter on call rate at a threshold of 0.95. How many more loci have you lost? What effect has this had on the smear plot?
5. Filter out the secondaries. You do not want closely linked loci for your analysis.
6. Finally, filter out the monomorphic loci.
7. Create a report of minor allele frequencies. It is a bit trivial in this case because of the low number of loci, but it will be quick and will give you an idea of what a report script can give you.

Where have we come?

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

- Interrogate your data to confirm the number of loci, individuals and population assignments.
- Extract the SNP genotypes into a simple data matrix.
- 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.
- Filter on call rate, repeatability, secondaries, minor allele frequency.
- Filter out 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

Lesson 5: Visualization

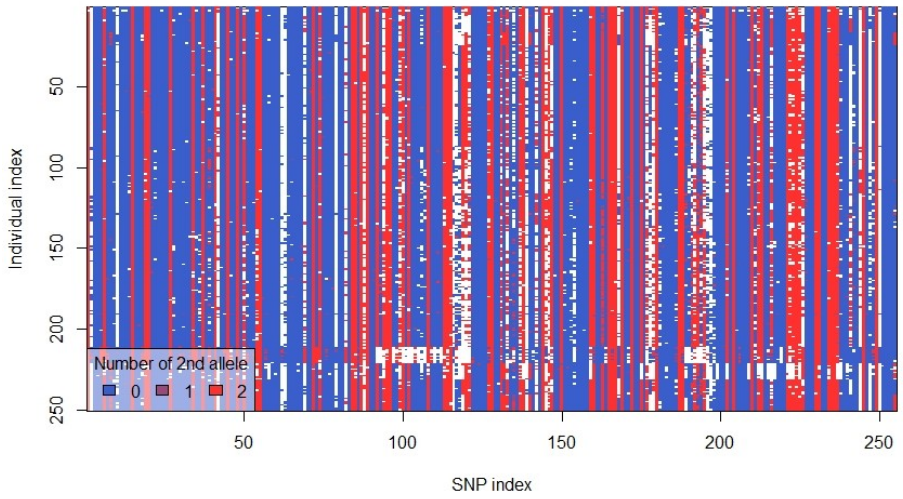
Overview

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

`glPlot(gl)`	display a smear plot of individuals by loci
`gl.plot(gl)`	display a smear plot with labels for individuals (useful if not too many)

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



Other common summary and visualization tools include

`gl.dist.pop(gl)`	generate a genetic distance matrix between populations (Euclidean distance ~ Rogers D by default)
`gl.pcoa.plot(pcoa, gl)`	display a plot of principal coordinates generated from a PCoA analysis applied to a distance matrix
`gl.tree.nj(gl)`	display genetic similarity in the form of a phenetic tree

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="pcfixed")
```

Exercises

1. Copy the test data set `testset.gl` into genlight object `gl`.
2. Calculate a range of genetic distance measures between the populations.

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.

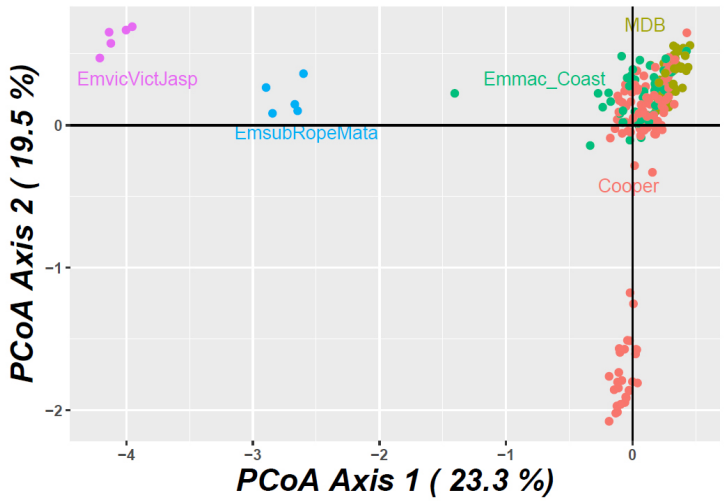
The relevant scripts are:

`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

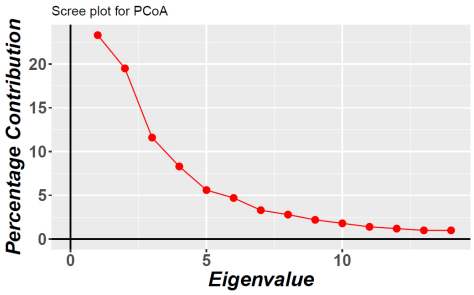
The results of the PCoA can be plotted using `gl.pcoa.plot()` with a limited range of options. The script is essentially a wrapper for `plot` {ggplot2} with the added functionality of {directlabels} and {plotly}.



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.

How many dimensions?

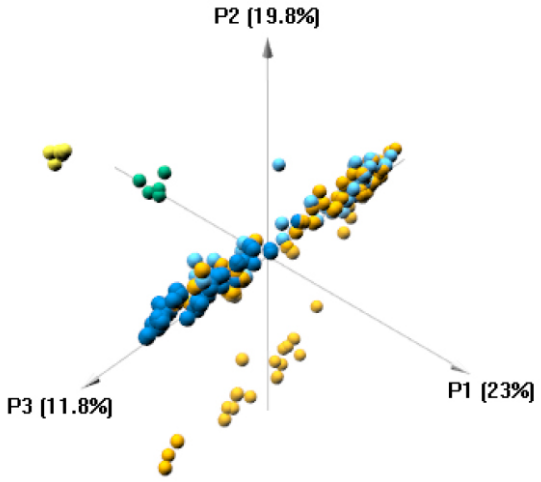
One way to decide how many dimensions to examine is a scree plot. A scree plot shows the percentage variation explained by each axis in the ordination successively. The script `gl.pcoa.scree()`, by default, will show the percentage variation in the data explained by each axis where the amount of variation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

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.

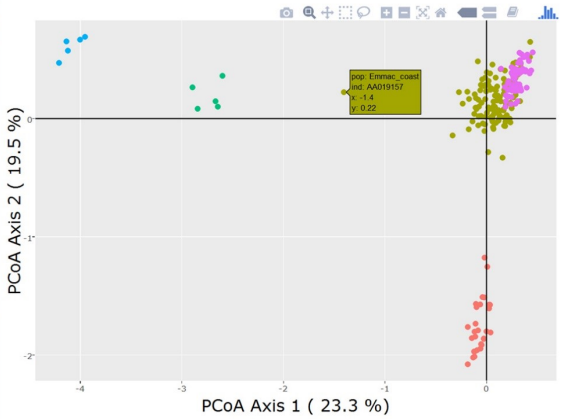


The plot is interactive in the sense that you can rotate the plot using the mouse to obtain the most discriminatory view.

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).

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 AA19157, from the coastal populations.

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.

Lesson 6: Fixed Difference Analysis

What is a fixed difference?

The PCoA approach outlined above considers allele frequency differences between individuals or populations as a basis for constructing a distance matrix which is then used by the PCoA algorithms to execute the ordination. There is some advantage in considering only fixed differences between populations, that is, allelic differences where the alleles have come to fixation to alternative states in populations taken pairwise. A fixed difference between two populations at a specific locus occurs when the population share no alleles at that locus. Allele frequencies may ebb and wane, but once a locus becomes fixed for an allele or suite of alleles, there is no returning until you get mutation or gene flow. Fixation is a significant biological event. The accumulation of fixed differences between two populations is considered a robust indication of lack of gene flow.

Fixed difference analysis in dartR

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

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

When the number of loci examined gets up into the 10s of thousands and the sample sizes are small (say < 5 individuals per population), the probability of a spurious fixed difference between two populations rises to unacceptable levels. To manage this, we use simulation to generate an expectation for the number of false positives, given the observed allele frequencies and the sample sizes. The observed count of fixed differences between two sample sites is compared with the null expectation to generate a p value for the significance of the observed count.

This approach differs from the more conventional approach using STRUCTURE (Pritchard, Stephens and Donnelly, 2000) in the emphasis placed on fixed allelic differences and diagnosability of resultant taxa. STRUCTURE and related analyses define aggregations such that each is in Hardy-Weinberg and linkage equilibrium across all loci. Under these criteria, STRUCTURE will define two populations as different even if they have only modest differences in allele frequencies that would bring them out of equilibrium if combined. Populations that are regarded as distinct by STRUCTURE would not necessarily be diagnosable, a criterion that is central to the fixed difference method. STRUCTURE also struggles with small sample sizes, whereas the fixed difference approach addresses this issue by the introduction of a test of significance.

The relevant scripts are

`fd <- gl.fixed.diff(gl)`	generate a matrix of fixed differences between populations
`fd <- gl.collapse(gl)`	aggregate populations for which there are no fixed differences (or a count of fixed differences less than a specified threshold, and collapse the difference matrix
`fd <- gl.collapse.recursive(gl)`	recursively apply gl.collapse until no further amalgamations occur
`fd <- gl.collapse.recursive(gl,test=TRUE)`	test the significance of the pairwise counts of fixed differences (computationally expensive)
`fd <- gl.collapse.pval(gl)`	aggregate populations for which there is not a significant count of fixed differences (computationally expensive)
`gl.pcoa.plot.3d(pcoa,gl)`	plot the individuals in the space defined by three specified axes, and allow mouseover rotation

Exercises

1. Generate a matrix of fixed differences for testset.gl and store the output in a new object called fd.

   Examine the elements of fd. Note that the p values have not been calculated by default, as this is very computationally expensive, especially with so many sampling sites.

2. Generate a collapsed fixed difference matrix by amalgamating populations that have no fixed allelic differences. Again, assign the output to a new object called fd.

3. Recode the new populations labels to something sensible.

4. Plot the resulting OTUs in a PCoA plot.

References

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

Georges, A., Gruber, B., Pauly, G.B., White, D., Young, M.J., Kilian, A., Zhang, X., Shaffer, H.B. and Unmack, P.J. 2018. Genome-wide SNP markers breathe new life into phylogeography and species delimitation for the problematic short-necked turtles (Chelidae: Emydura) of eastern Australia. Molecular Ecology, in press.

Gower, J. C. (1966). Some distance properties of latent root and vector methods used in multivariate analysis. Biometrika, 53, 325–338.

Lesson 7: Connection to other established software

Conversion scripts

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.

To assist with this, 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 file
`gl2genind()`	converts a genlight object to a genind object as defined by the {adegenet} package
`genind2gl()`	converts a genind object as defined by the {adegenet} package to a genlight object

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.

Lesson 8: Support for Phylogenetics

Overview

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

When applied to mitochondrial sequence data, essentially a single locus, the resultant tree is for the mitochondrion, from which one may choose to infer the species tree. If the analysis is applied to nuclear data for individuals or populations subject to tokogenic exchange (horizontal transfer), the resultant tree is a summary of genetic similarity, not a phylogeny, or a loose combination of the two.

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

Unfortunately, in nature, phylogenies do not follow a process that maintains tree distances or strict tree-like structure in the data. Homoplasy and differential histories among the genes (independent lineage sorting) means that the process is not one of mining the true tree, but one of estimating the “best tree” given that the data contains conflicting signals on the true trajectory of ancestry and descent. The best tree might be the most parsimonious tree, or the maximum likelihood tree, or the best bet optimised using all available information. There are various approaches to gaining consensus across multiple gene trees.

dartR and Phylogenetics

This is a controversial area, and so we have simply provided streamlined pathways to some more common analyses, and leave it to you to decide which you think is most appropriate to your case. The options are

`gl2phylip()` produces a distance matrix as input for Phylip (Felsenstein, 1989), including an option for bootstrapping. This is a distance approach to recovering phylogenies.

`gl2svdquartets()` produces an input nexus file for PAUP* (Swofford, 2003) to undertake a phylogeny using SVD quartets. This

approach is one way to deal with the challenges of independent lineage sorting. Method 1 has one line per population (OTU) and uses ambiguity codes (Swofford pers. comm.); Method 2 has two lines per population (OTU) to allow representation of both SNP states (Chifman & Kubatko, 2014).

`gl2snapp()` produces an input nexus file for BEAUti (part of the BEAST software package, Bouckaert et al., 2014) which can then be used to access the SNAPP package (Bryant, Bouckaert, Felsenstein, Rosenberg, & RoyChoudhury, 2012).

`gl2treemix()` produces an input file for treemix (Pickrell & Pritchard, 2012), which looks for stress between the data matrix and the ML tree, and reduces that stress by hypothesising migration events (gene flow or introgression).

`gl2phylonet()` produces an input file for Phylonet (Zhu, Wen, Yu, Meudt, & Nakhleh, 2018), which has options for analysing data not as a bifurcating tree, but as a network. It thus accommodates tokogeneic exchange. This script has been tested only so far as confirming the format of the input file for Phylonet. Let us know how you go.

`gl2fasta()` produces a fastA file in a number of formats suitable for phylogenetic analysis, in some cases, by concatenating the sequence tags. Methods 1 and 2 concatenate the sequence tags for analysis by Garli which uses a mutational model that has been generated from the base frequencies and ts/tv ratios. Method 1 replaces heterozygous sites with the relevant ambiguity code; method 2 selects one SNP state at random. Methods 3 and 4 combines only the SNP base information, which is suitable for use by programs like RAXML. Again, Method 1 replaces heterozygous sites with the relevant ambiguity code; method 2 selects one SNP state at random.

References

Bouckaert, R., Heled, J., Kühnert, D., Vaughan, T., Wu, C.-H., Xie, D., … Drummond, A. J. (2014). BEAST 2: A Software Platform for Bayesian Evolutionary Analysis. PLoS Computational Biology, 10(4), e1003537. doi:10.1371/journal.pcbi.1003537

Bryant, D., Bouckaert, R., Felsenstein, J., Rosenberg, N. A., & RoyChoudhury, A. (2012). Inferring Species Trees Directly from Biallelic Genetic Markers: Bypassing Gene Trees in a Full Coalescent Analysis. Molecular Biology and Evolution, 29(8), 1917–1932. doi:10.1093/molbev/mss086

Buneman, P. (1973). A note on the metric properies of trees. Journal of Combinatorial Theory, 17B, 48–50.

Chifman, J., & Kubatko, L. (2014). Quartet inference from SNP data under the coalescent model. Bioinformatics, 30, 3317–3324.

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

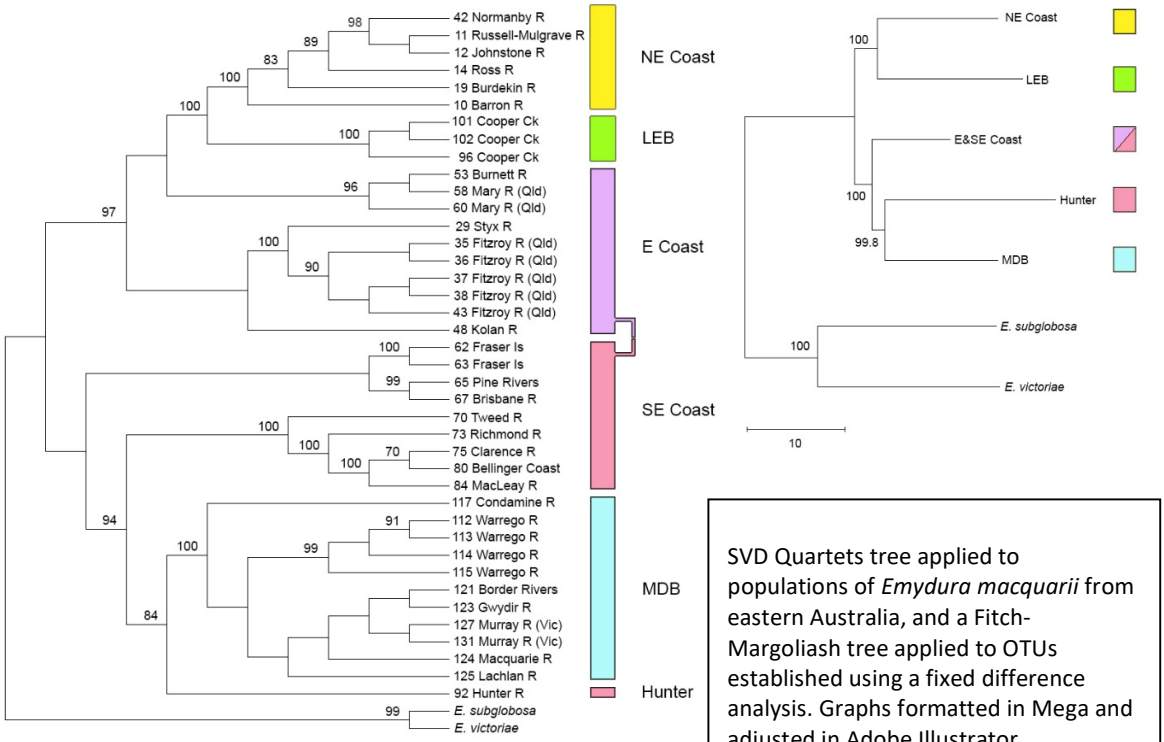
Felsenstein, J. (2004). Inferring Phylogenies. Sunderdland, MA USA: Sinauer Associates.

Pickrell, J. K., & Pritchard, J. K. (2012). Inference of population splits and mixtures from genome-wide allele frequency data. PLoS Genetics, 8, e1002967.

Swofford, D. L. (2003). PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods). Version 4. Sunderland, Massachusetts, USA: Sinauer Associates.

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

Zhu, J., Wen, D., Yu, Y., Meudt, H. M., & Nakhleh, L. (2018). Bayesian inference of phylogenetic networks from bi-allelic genetic markers. PLOS Computational Biology, 14, e1005932.



SVD Quartets tree applied to populations of *Emydura macquarii* from eastern Australia, and a Fitch-Margoliash tree applied to OTUs established using a fixed difference analysis. Graphs formatted in Mega and adjusted in Adobe Illustrator.

Lesson 9: Admixture Analysis

Overview

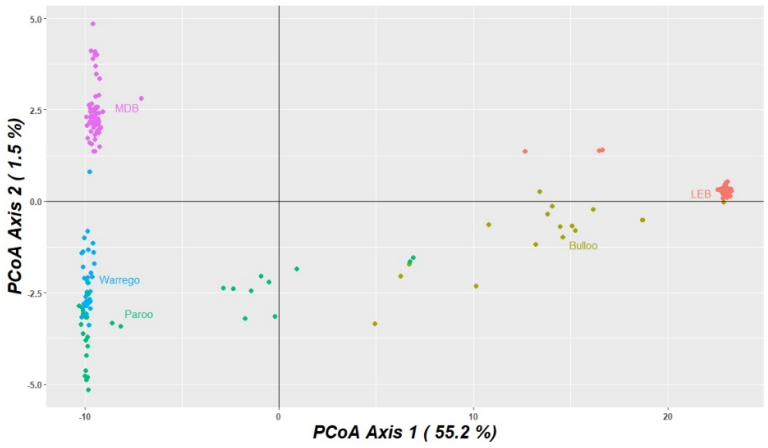
Support for admixture analysis covers a number of approaches. Perhaps the best place to start is to examine structure among the sampled individuals with principal coordinates analysis (PCoA)(Gower, 1966). Ordination with PCA, PCoA and MDS essentially treat each locus as a variable, construct linear combinations of these original variables in order of the percentage of variation among individuals that they explain. This enables the visualization of the structure in the dataset in one, two, three or rarely more dimensions. Noise variation is pushed down to deeper dimensions which are disregarded.

When two populations diverge, they wander apart in ordinated space and their separation is represented in one linear dimension of the final ordination. Independent divergence of three populations will be represented in two ordinated dimensions, and so on. In this context, contemporary admixture will be evident as individuals falling between the two parental populations. An example is shown where individuals from the Murray-Darling Basin, the Paroo, Barcoo and Lake Eyre Basin are plotted in an ordinated space of two dimensions.

PCoA

So a first step is to plot individuals in an ordinated space.

```
MDB_LEB <- loadRDS(file="MDB_LEB.Rdata")
pcoa <- gl.pcoa(MDB_LEB)
gl.pcoa.plot(pcoa, MDB_LEB, xaxis=1, yaxis=2, labels="pop")
```



You can interrogate the plot using

```
gl.pcoa.plot(pcoa, MDB_LEB, xaxis=1, yaxis=2,
    labels="interactive")
plotly()
```

NewHybirds

For a quantitative analysis, we recommend NewHybrids (Anderson & Thompson, 2002). NewHybrids employs a statistical method for identifying species hybrids using data on multiple, unlinked markers which does not require that allele frequencies be known in the parental species. However, advance knowledge of the parental species, uncontaminated by introgressed individuals, increases the performance of the software.

The probability model used is one in which parentals and various classes of hybrids (F1s, F2s, and various backcrosses) form a mixture from which the sample is drawn. Using the framework of Bayesian model-based clustering, NewHybrids computes, using Markov chain Monte Carlo, the relative likelihood (posterior probability) that each individual belongs to each of the distinct hybrid classes.

NewHybrids is limited to ca 200 loci because of memory constraints, but this is not as serious a constraint as one might expect. You need relatively few loci to identify hybridization. We choose the best 200 available loci on the basis of their being different and fixed in each of the two nominated parental populations, and then on avgPIC (a DArT statistic measuring information content, akin to an even-ness score).

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

If you have installed it, then by specifying the installation location, gl.nhybrids will run NewHybrids and pass the results back to R. If you do not specify an installation location, gl.nhbyrids() will simply produce an input file for NewHybrids.

```
gl.nhybrids(MDB_LEB,p0=parent0,p1=parent2,
    outpath=getwd(),outfile="MBD_LEB.nhy",
    nhyb.directory="C:/workspace/R/NewHybsPC",BurnIn=10000,
    sweeps=10000)
```

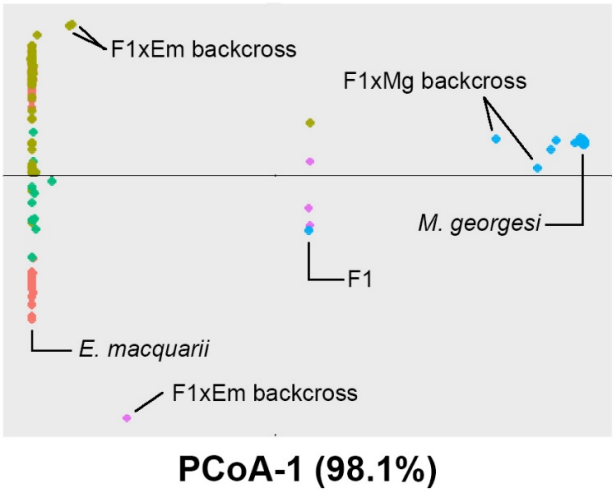
The output is a csv file in which all individuals are listed, and their assignment to one of the parental populations is indicated, if the individuals are not admixed. Admixed individuals will be assigned a posterior probability of membership of one of the classes, F1, F2, backcross of F1 to parental 1, backcross of F1 to parental 2.

There is no value in going further than this, because the likelihood bins for more distant crosses overlap to the point that the results are not that useful.

The following table shows the results for hybridization and introgression between the introduced *Emydura macquarii* and the critically endangered *Myuchelys georgesi* in the Bellinger River.

pop	*Myuchelys*	*Emydura*	F1	F2	F1xMy	F1xEm	N
p0	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	5
Myuchelys (Bellinger)	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	110
Myuchelys (Bellinger)	0.0000	0.0000	0.0000	0.0000	1.0000	0.0000	2
Myuchelys (Kalang)	0.0000	0.0000	0.0000	0.0000	0.0000	1.0000	1
Myuchelys (Kalang)	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	3
Myuchelys (Taronga)	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	16
Myuchelys (Taronga)	0.0000	0.0000	1.0000	0.0000	0.0000	0.0000	1
Emydura (Bellinger)	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	51
Emydura (Kalang)	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	14
p1	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	63

Here are the combined results of the PCoA and NewHybrids designations, with a bit of help from Adobe Illustrator.



References

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

Gower, J. C. (1966). Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, *53*, 325–338.

Lesson 10: Population Assignment

Overview

Assigning individuals of unknown provenance to populations of known provenance is a challenging exercise, and several approaches have been suggested. Perhaps the simplest is to calculate the probabilities of yielding the observed genotype of the unknown individual given the observed allele frequencies in each the target population. Using this approach, the individual is assigned notionally to those populations for which this probability or likelihood is greatest; populations for which the probability or likelihood is lower than some level of significance are eliminated from further consideration. This approach was first applied in a study of microsatellite markers in bear populations (Paetkau, Slade, Burden, & Estoup, 2004) and subsequently applied using classical and Bayesian approaches to estimating probabilities (Blanchong, Scribner, & Winterstein, 2002; Gotz & Thaller, 1998).

Private Alleles

For various reasons, including not wishing to duplicate software options that are already available, we have taken an alternate approach. We first eliminate from consideration those target populations where a SNP allele is present in the unknown individual but not in the target – if there are alleles in the unknown individual not present in a particular population, then it is unlikely that the focal individual was drawn from that population.

```
x <- gl.assign(testset.gl, id="UC_00146", nmin=10,
alpha=0.05, t=1)
```

```
  0 EmmacMaclGeor
  1 EmmacBurnBara EmmacRichCasi EmmacTweeUki
  2 EmmacBrisWive EmmacMDBBowm EmmacMDBCudg EmmacMDBForb EmmacMDBMaci
    EmmacMDBMurrMung
  3 EmmacFitzAllig EmmacMDBCond EmmacMDBSanf EmmacRussEube
  4 EmmacBurdMist EmmacJohnWari EmmacRoss
  6 EmmacCoopEulb
  7 EmmacCoopCully
 16 EmmacCoopAvin
```

Note that the focal unknown individual shares all of its alleles only with EmmacMacGeor (0 private alleles), which is the population from which it was drawn for this example.

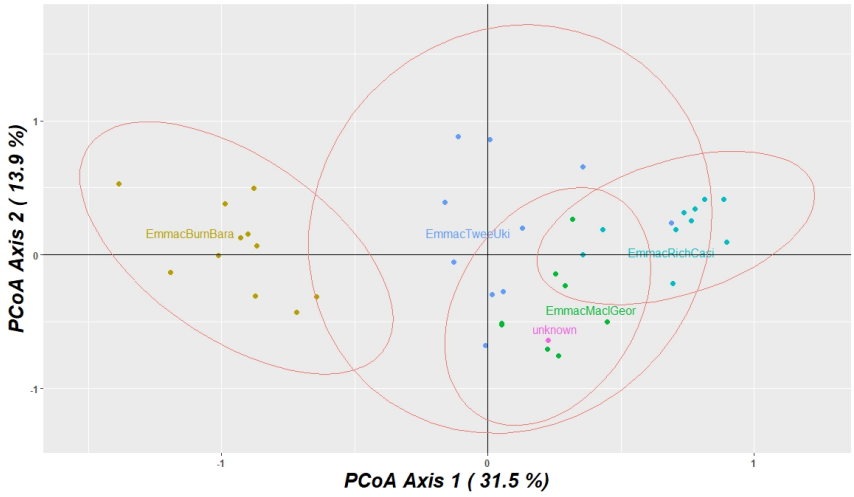
In many cases, examining private alleles in this way will narrow down the possible source populations considerably, and depending on the spatial resolution required for the assignment (say, Australia or New Guinea), may provide a satisfactory answer.

PCoA

Second, we examine the position of the unknown individual relative to the target populations in a reduced ordinated locus space using PCoA. This graphic representation is provided by gl.assign().

Addition of confidence ellipses then allows a decision to eliminate some populations from consideration as the source of the unknown individual (shown in pink).

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



Confidence Envelopes

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

The output of this analysis is as follows:

```
      Population   Index      CE Assign
2 EmmacMaclGeor -1.1294 -2.6193    yes
4  EmmacTweeUki -1.2056 -2.6193    yes
3 EmmacRichCasi -3.9405 -2.6193     no
1 EmmacBurnBara -7.8446 -2.6193     no
```

Index is a weighted log-likelihood score, the bigger it is, the more likely does the unknown belong. The CE is the log-likelihood of an individual residing on the confidence envelope. If Index > CE, then membership of the unknown to the respective population is unlikely.

A critical issue is whether the number of individuals in the target populations are sufficient to characterize them in the critical considerations made here. Is the sample size sufficient to support the identification of a private allele in the unknown? Is it sufficient to confidently construct confidence ellipses in the PCoA plot? Is it sufficient to provide a robust estimate of the distribution of individuals

along each axis of the ordination in order to adequately estimate the likelihood of the unknown on that axis? We have set a default of nmin=10 (2n = 20), but this is a matter of judgement that needs to be considered when planning a study.

References

Blanchong, J. A., Scribner, K. T., & Winterstein, S. R. (2002). Assignment of individuals to populations: Bayesian methods and multi-Locus genotypes. Journal of Wildlife Management, 66, 321–329.

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

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

Lesson 11: Relatedness

Overview

Assessing relatedness among individuals is not straightforward, not least of all because of a distinction between genealogical relatedness (one is related to ones descendents) and genetic relatedness (the degree of genetic similarity attributable to descent). Given sufficient generations, a proportion of your descendants (i.e. genealogical relatives) will have inherited none of your alleles at any locus (i.e. genetically unrelated). That is the nature of recombination and independent assortment of chromosomes.

So when we talk of relatedness, we refer to genetic relatedness.

It is a complex area of analysis, well developed in breeding of domestic animals, and draws upon mixed linear models best analysed in packages like ASreml 4 (**https://www.vsni.co.uk/software/asreml/**).

These analyses are best performed on organisms for which there are substantial genomic data, such as humans or cattle.

We restrict our attention in dartR to generating a G-matrix of genetic relatedness, and on visualization tools. We also provide advice on converting your SNP data to a form that can be used by ASreml.

The G matrix

The G matrix is the genetic (or genomic) relationship matrix (GRM) and is estimated from the SNP data by the procedure described by Yang et al. (2010). It is used extensively in the breeding industry for obtaining estimates of trait heritability and genome-wide association studies (Yang et al., 2017).

$$G_{jk} = \frac{1}{m} \sum_{i=1}^{m} \frac{(x_{ij} - 2p_i)(x_{ik} - 2p_i)}{2p_i(1 - p_i)}$$

where G is the relatedness between individuals j and k; xij is the number of copies of the minor allele for the j-th individual at locus i; pi is the frequency of the minor allele for locus i; m is the number of loci.

G is a measure of how much the SNP scores covary $(x_{ij} - 2p_i)(x_{ik} - 2p_i)$ relative to what is expected on average [expected heterozygosity $2p_i(1 - p_i)$], averaged over all loci.

G is a relative measure of the covariance between individuals averaged over all loci.

Graphical representation

The relationships among individuals represented by the G-matrix can be visualized as a network using iGraph software in R. A network graph is comprised of nodes

for the individuals and links between the nodes that represent the relationships between them.

By judicious choice of a threshold for the strength of relationship, patterns can be seen in the network plot.

As an example, we present data for individuals of the endangered freshwater turtle Myuchelys georgesi from the Bellinger River of central NSW. The data are held in a saved genlight object called Mygeo.

The data contain SNP genotypes for wild animals, for animals in a captive colony held at Taronga Zoo, and three clutches of their offspring. The wild population include some F1 hybrids with the introduced Emydura macquarii and some individuals showing evidence of introgression detected by a NewHybrids analysis. So quite a mixed bag.

The wild population can be further divided into individuals sampled before a devastating virus attacked the population, and those sample post virus.

The first step in the analysis is to generate a G-matrix of genetic relatedness among individuals.

```
Mygeo <- loadRDS("Mygeo.gl")
G <- gl.grm(Mygeo)
```

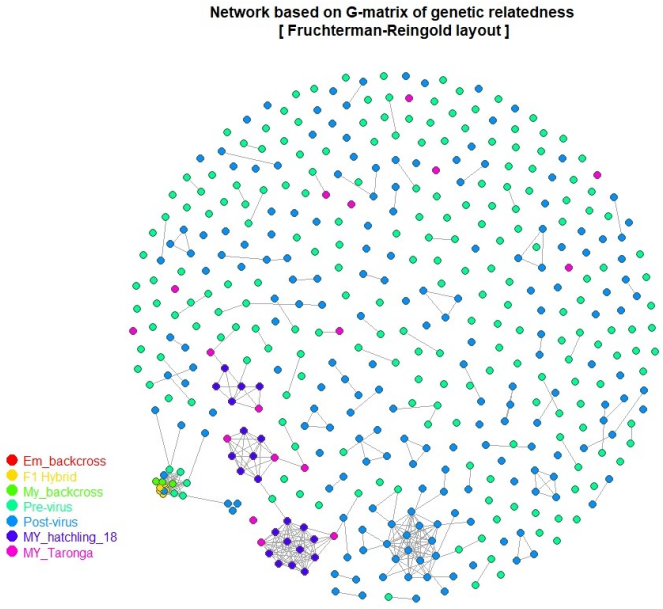
This matrix is very large because of the number of individuals involved, but we can look at the first 10 individuals to see what the data looks like.

```
G[1:10,1:10]
```

The pairwise values are the genetic relatedness between individuals at the head of the rows and columns, and the diagonal contains a measure of inbreeding.

To visualize these data, we use the script

```
gl.grm.network(G, Mygeo)
```



The key to producing a network graph of this type is to adjust the threshold for relationships to be displayed as links to a value that has sparse connections for the bulk of the population. In this way, the stronger than average relationships become readily visible.

Here we have the three families of hatchlings and their parents (only mum in one case) clearly represented as clusters. The F1 hybrids and introgressed animals pull out as an isolated cluster. There appears to be no clear difference in relationship pre and post virus, but there is one cluster in the post virus individuals that is worthy of further detailed examination.

Linking to ASreml

Linking across to the R package ASreml could not be easier. Simply convert your SNP data in your genlight object to a matrix and rescale the SNP scores.

```
r <- as.matrix(gl) - 1
```

References

Yang, J., Benyamin, B., McEvoy, B.P., Gordon, S., Hendeers, A.K., Nyhold, D.R., Madden, P.A., Heath, A.C., Martin, N.G., Montgomery, G.W. and Goddard, M.E. (2010). Common SNPs explain a large proportion of the heritability for human height. Nat. Genetics 42:565-569.

Yang, J., Zeng, J., Goddard, M.E., Wray, N.R. and Visscher, P.M. (2017). Concepts, estimation and interpretation of SNP-based heritability. Nature Genetics volume 49:1304–1310.

Lesson 12: Population genetics

Overview

dartR supports a number of basic functions to calculate population genetic statistics. It is beyond this workshop to give a deeper introduction to those concepts and discussions. An important aim in population genetics is to study population structure, which in turn is the result of often important population processes, such as reproduction, inbreeding, dispersal and others. Knowledge on those processes are often important for the conservation of species. Currently there are the following functions:

`gl.report.hwe()` reports Hardy-Weinberg-Equilibrium over loci for all individuals and over population

`gl.filter.hwe()` filters loci that are in depature of Hardy-Weinberg-Equilibrium

`gl.report.heterozygosity()` calculates the observed heterozygosity of populations

`gl.report.pa.pop` calculates the number of private alleles and fixed alleles between pairs of populations

`gl.diversity` Calculates several diversity indices within populations such as number of alleles, Shannon diversity and heterozygosity (alpha) and between pairs of populations such as mutual information and Jost-D. Also plots the so-called q-profile for populations. See Sherwin et al. 2017.

`gl.ibd()` performs and isolation by distance analysis based in populations. Pairwise Euclidean distances are regressed (using a Mantel-test) against pairwise genetic distances (Fst)

`gl.fst.pop()` Calculates pairwise fsts between populations

`gl.amova()` Performs an amova [beta]

Exercises

1. Report observed heterozygosity for the data set `foxes.gl`
2. Study the pairwise fixed differences between foxes from WA and NSW? Which pair of population has the highest number of private alleles?
3. Remove individuals from Tasmania and ACT from the data set and calculate Shannon diversity (one_D_alpha) for all populations. Which population has the highest Shannon diversity (meant to be a measurement of overall evolvability)
4. Calculate pairwise D (using gl.diveristy) and pairwise Fst using stammpfst function from package StAMMP)

Landscape genetics

Load the possum data set.

```
possums.gl
```

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.gl, projected = TRUE)
```

Exercises

1. Study the possum genlight object (how many individuals per population)

2. Overall how many loci are in the data set?

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

For the 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.

We need landscape first [better a resistance surface]. We will use the in-built example landsape.som

```
plot(landsape.sim)
```

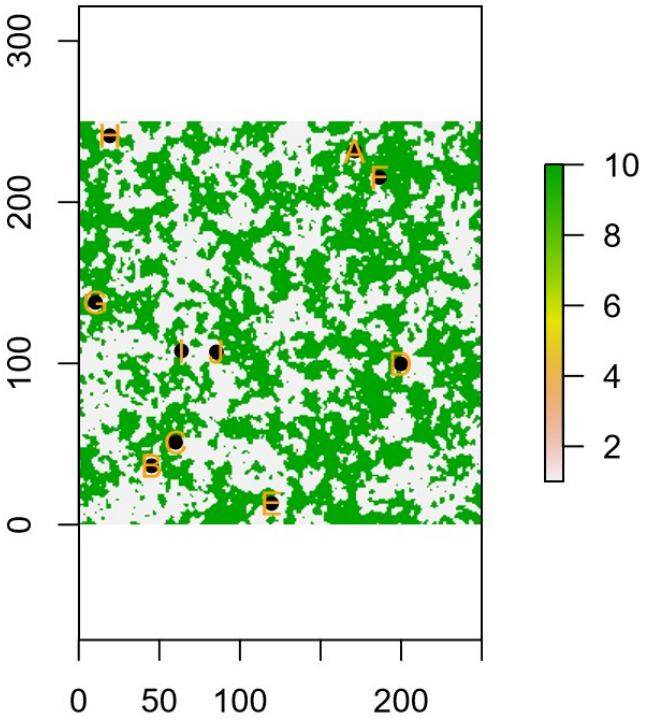
to perform a landscape genetic analysis

Some R “magic”…

We calculate the population centers via:

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

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



The calculation of Euclidean distances between the population is simply via the dist function:

```
coords <- cbind(xs, ys)
eucl <- as.matrix(dist(coords))
```

For the calculation of costdistances we use function of the package PopGenReport:

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

As before we calculate pairwise Fst's between population using the StAMPP package

```
gd <- as.matrix(as.dist(gl.fst.pop(possums.gl)))
```

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  0.048
## 2 Gen ~Euclidean | cost -0.2502 0.833
```

In the package PopGenReport is a convenience function that performs all the necessary steps in one go. 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.gl)
glc <- genleastcost(pgi, fric.raster = landscape.sim, gen.distance = "D", NN = 8, pathtype = "leastcost")
```

`gl2shp()` a simple function that allows to export genlight object with coordinates in (@other$latlong) to be exported as shp or kml file.

We can also create a simple maps using ggmap and stamen maps (based on openstreetmaps)

Be aware you need the latest version of ggmap. You might need to use

```
library(devtools)
install_github("dkahle/ggmap")

library(ggmap)
#define extent
bb <- matrix(c(100, -50,170,0), 2,2)
#get map from osm
stamen <- get_stamenmap(bb,zoom=4, extent = "device")
#plot map with points
df <- foxes.gl@other$ind.metrics
ggmap(stamen)+ geom_point(aes(x=lon, y=lat, col=pop(foxes.gl)), data=df)
```

Using simulation functions and resampling methods

We can use simulation techniques to test the effect of sample size on our findings. For example in a study on the population structure 50 individuals per population were sampled. Often we would like to know If 20 individuals would have been enough to find a similar population structure (e.g. similar levels of pairwise Fsts).

Here we can “downsample” our populations, rerun the fst calculations (repeat the procedure 100 times) and check the distribution of pairwise fsts.

In another example we want to check our ability to identify hybrids between two populations. A simulation approach would create hybrids between individuals and then run a genomic relatedness matrix between individuals to identify hybrids.

The following functions are now in dartR to support such simulations:

`gl.sim.offsping()` simulates the a number of offsprings between fathers and mothers (which are provided as genlight objects)

`gl.sim.ind()` simulates individuals based on the allele frequency and number of loci provided by a genlight object.