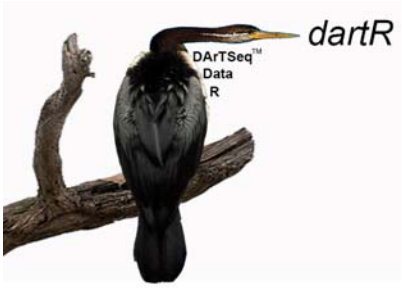
Wildlife Genomics@UC
wildlifegenomics.biomatix.org

UNIVERSITY OF CANBERRA

SNP Analysis using dartR



Interrogating and Manipulating Data

Version 2

Copies of the latest version of this tutorial are available from:

The Institute for Applied Ecology
University of Canberra ACT 2601
Australia

Email: georges@aerg.canberra.edu.au

dartR is a collaboration between the University of Canberra, CSIRO and Diversity Arrays Technology, and is supported with funding from the ACT Priority Investment Program, CSIRO and the University of Canberra.

Contents

Session 1: Interrogating a genlight object 4

Examine attributes 4

Examine metrics 5

Use adegenet accessors 5

Exercises 5

Use dartR report functions 6

Example: Reporting on Call Rate 6

Where have we come? 9

Session 2. Data Manipulation 10

Overview 10

An early cautionary note 10

Removing populations and individuals 11

Dropping populations, individuals and loci 11

Selection on an individual metric 11

Replacing the pop variable 11

Recalculating locus metrics 11

Recode tables 12

Tidy up the workspace 13

Where have we come? 14

Session 1: Interrogating a genlight object

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

Examine attributes

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

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

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

 // 250 genotypes,  255 binary SNPs, size: 751.3 Kb
 7868 (12.34 %) missing data

 // Basic content
   @gen: list of 250 SNPbin
   @ploidy: ploidy of each individual  (range: 2-2)

 // Optional content
   @ind.names:  250 individual labels
   @loc.names:  255 locus labels
   @loc.all:  255 alleles
   @position: integer storing positions of the SNPs
   @pop: population of each individual (group size range: 1-11)
   @other: a list containing: loc.metrics  latlong  ind.metrics
```

Note: If it is a SNP dataset, the ploidy of each individual will be reported as (range 2-2). If it is a SilicoDArT dataset, the ploidy will be reported as (range: 1-1)

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

Displayed are the number of genotypes (individuals/specimens/samples), the size of the genlight object, and the number of missing values. The ploidy value should be 2-2 for SNP data for a diploid organism (dartR does not have support for polyploid organisms), so if it is something else, you have a problem with your data. SilicoDArT presence absence data has the ploidy set to 1-1.

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

If any of the optional content slots indicated above are missing, consider running

```
gl <- gl.compliance.check(gl)
```

which will render the genlight object compliant with dartR.

Examine metrics

To access metadata directly you can use commands of the form

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

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

Remind yourself of the variables in the metadata using

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

and

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

Use adegenet accessors

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

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

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

Exercises

1. Copy the genlight object `testset.gl` to a working genlight object `gl`.
2. How many loci are represented in this dataset?
3. How many individuals have been scored?
4. Are the individuals assigned to populations and if so, how many populations? What are the names of the populations?

	5. Examine the genotypes for the first 5 individuals for the first 10 loci. 6. How are missing values represented? 7. Examine the structure of the dataset in a smear plot. What can you say about allelic dropout? 8. Redo activities 2-7 with your own data.

Use dartR report functions

Various report scripts provide summaries for SNP and SilicoDArT datasets, often as a prelude to filtering. Note: Do not assign the output of the report to your genlight object or you will overwrite it. The common ones that are used as a prelude to filtering are listed below.

`gl.report.callrate`	summarises CallRate values
`gl.report.reproducibility`	summarises repAvg (SNP) or reproducibility (SilicoDArT) values
`gl.report.monomorphs`	provides a count of polymorphic and monomorphic loci
`gl.report.secondaries`	provides a count of loci that are secondaries, that is, loci that reside on the one sequence tag.
`gl.report.rdepth`	reports the estimate of average read depth for each locus
`gl.report.hamming`	reports on Hamming distances between sequence tags
`gl.report.overshoot`	reports loci for which the SNP has been trimmed along with the adaptor sequence
`gl.report.taglength`	reports a frequency tabulation of sequence tag lengths

Example: Reporting on Call Rate

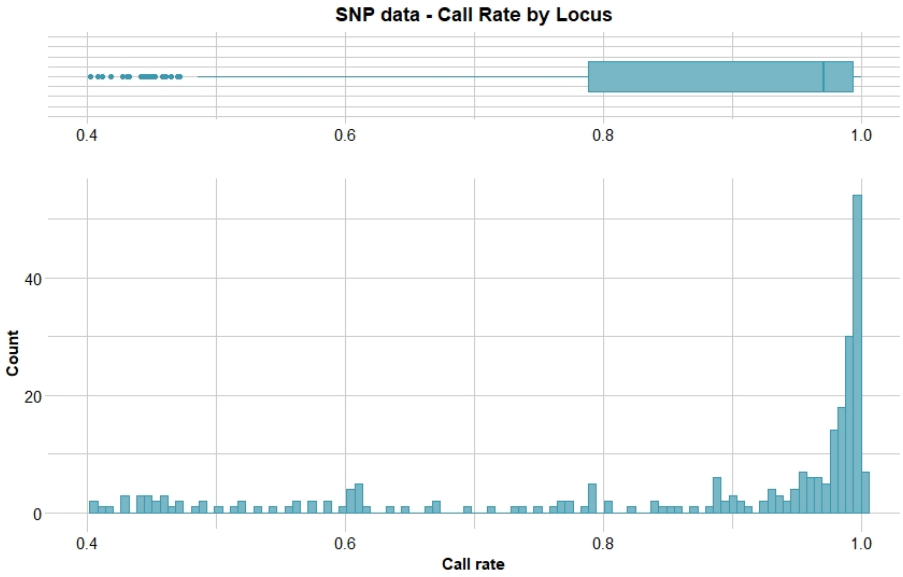
As an example, let us consider reporting on call rate across loci.

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

```
Starting gl.report.callrate
Processing SNP data
Reporting Call Rate by Locus
No. of loci = 255
No. of individuals = 250
Minimum      :  0.401932
1st quantile :  0.7879225
Median       :  0.971014
Mean         :  0.8634915
3r quantile  :  0.993237
Maximum      :  1
Missing Rate Overall:  0.12
```

```
Quantile Threshold Retained Percent Filtered Percent
1      100% 1.0000000        7     2.7      248    97.3
2       95% 0.9990340       19     7.5      236    92.5
3       90% 0.9971010       34    13.3      221    86.7
4       85% 0.9961350       44    17.3      211    82.7
5       80% 0.9951690       53    20.8      202    79.2
6       75% 0.9932370       67    26.3      188    73.7
7       70% 0.9911108       77    30.2      178    69.8
8       65% 0.9884060       91    35.7      164    64.3
9       60% 0.9855070      103    40.4      152    59.6
10      55% 0.9803862      115    45.1      140    54.9
11      50% 0.9710140      128    50.2      127    49.8
12      45% 0.9577778      140    54.9      115    45.1
13      40% 0.9389374      153    60.0      102    40.0
14      35% 0.9021252      166    65.1       89    34.9
15      30% 0.8606764      178    69.8       77    30.2
16      25% 0.7879225      191    74.9       64    25.1
17      20% 0.6672464      204    80.0       51    20.0
18      15% 0.6039616      216    84.7       39    15.3
19      10% 0.5151690      229    89.8       26    10.2
20       5% 0.4506281      242    94.9       13     5.1
21       0% 0.4019320      255   100.0        0     0.0
Saving the ggplot to session tempfile
Saving tabulation to session tempfile
Completed: gl.report.callrate
```

The report is fairly self-evident. The table provides you with a basis for deciding an appropriate Call Rate to use as a threshold for filtering loci. For example, if you reject all loci that fail to call in any individual, you will lose all but 30.9% of your loci. That is too stringent. However, a threshold of 0.95 might be appropriate, with 67.8% of loci retained. The plot might make the decision simpler.



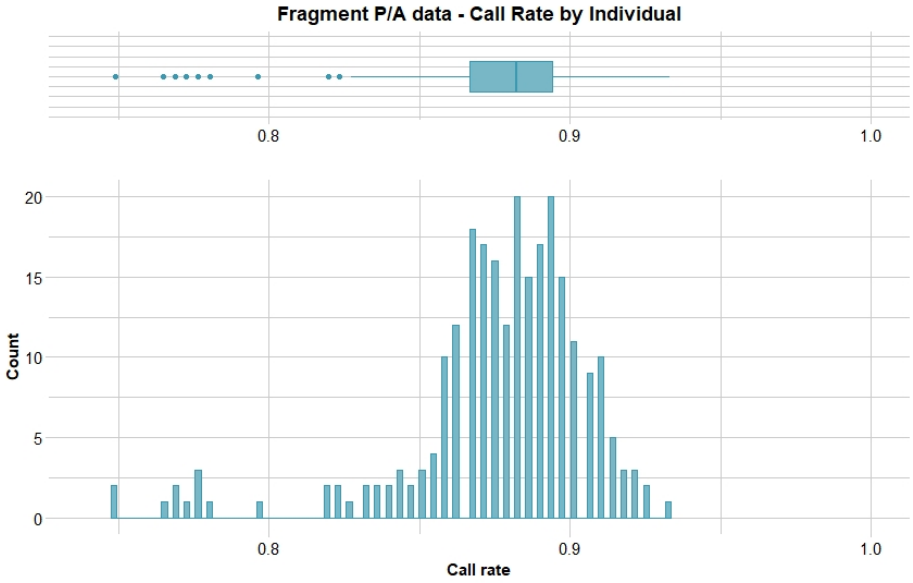
You can consider call rate across individuals using

```
gl.report.callrate(testset.gl, method='ind')
```

```
Starting gl.report.callrate
Processing SNP data
Reporting Call Rate by Individual
No. of loci = 255
No. of individuals = 250
Minimum      :  0.7490196
1st quantile :  0.8666667
Median       :  0.8823529
Mean         :  0.8765804
3r quantile  :  0.8941176
Maximum      :  0.9333333
```

```
Missing Rate Overall:  0.12

Quantile Threshold Retained Percent Filtered Percent
1      100% 0.9333333        1     0.4      249    99.6
2       95% 0.9137255       14     5.6      236    94.4
3       90% 0.9058824       33    13.2      217    86.8
4       85% 0.9019608       44    17.6      206    82.4
5       80% 0.8980392       59    23.6      191    76.4
6       75% 0.8941176       79    31.6      171    68.4
7       70% 0.8941176       79    31.6      171    68.4
8       65% 0.8901961       96    38.4      154    61.6
9       60% 0.8862745      111    44.4      139    55.6
10      55% 0.8823529      131    52.4      119    47.6
11      50% 0.8823529      131    52.4      119    47.6
12      45% 0.8784314      143    57.2      107    42.8
13      40% 0.8745098      159    63.6       91    36.4
14      35% 0.8705882      176    70.4       74    29.6
15      30% 0.8705882      176    70.4       74    29.6
16      25% 0.8666667      194    77.6       56    22.4
17      20% 0.8627451      206    82.4       44    17.6
18      15% 0.8588235      216    86.4       34    13.6
19      10% 0.8466667      225    90.0       25    10.0
20       5% 0.8213725      237    94.8       13     5.2
21       0% 0.7490196      250   100.0        0     0.0
Saving the ggplot to session tempfile
Saving tabulation to session tempfile
Completed: gl.report.callrate
```

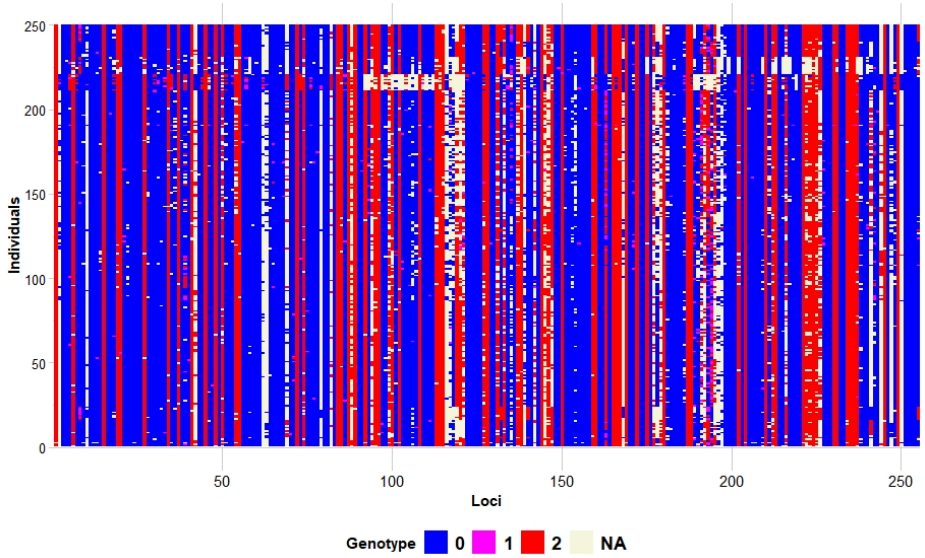


Again, the printout and plot give you a basis for selecting a threshold for filtering individuals with a poor call rate.

An alternative way of visualizing the call rate across individuals and loc is to examine a smear plot.

```
gl.smearplot(gl)
```

SNP loci are shown along the x-axis and individuals/specimens along the y-axis. Missing data are shown as white, so it can be seen that some loci (vertical white stripes) have an exceptionally low call rate, and some individuals have a poor call rate for some loci (horizontal white bands).



Try this for yourself using the sample SilicoDart dataset

```
gl.report.callrate(gs)

gl.report.callrate(gs, method='ind')

gl.smearplot(gs)
```

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

Generate reports for `reprocubility, rdepth, secondaries, overshoot, taglength, monomorphs` and other locus metadata in `gl`. Try it on your own data.

Where have we come?

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

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

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

Session 2. Data Manipulation

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

Overview

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

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

An early cautionary note

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

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

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

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

Even subsetting is potentially problematic. For example

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

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

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

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

Removing populations and individuals

Dropping populations, individuals and loci

The following set of functions apply to populations and individuals.

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

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

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

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

`gl <- gl.drop.loc()` remove listed loci

`gl <- gl.keep.loc()` keep only the listed loci

Selection on an individual metric

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

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

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

Replacing the pop variable

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

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

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

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

Try this with your own data.

Recalculating locus metrics

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

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

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

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

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

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

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

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

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

`gl <- gl.define.pop()` create a new population for listed individuals; their old population assignments will be discarded.

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

`gl <- gl.rename.pop()` `rename a population.`

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

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

Recode tables

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

`gl <- gl.make.recode.pop()` make a recode table based on existing population labels. You will need to edit the second column of the recode table to specify the new labels to apply.

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

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

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

to recompute the locus metrics.

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

Tidy up the workspace

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

```
rm(newguys)
```

Where have we come?

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

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

Ende