DARTRBOOK

Bernd Gruber, Arthur Georges, Carlo Pacioni, Luis Mijangos

Table of contents

Preamble

This is a Quarto book and the following page is just for convenience when writing the book.

(so just ignore)

Exercise

Note that there are five types of callouts, including: `note`, `warning`, `important`, `tip`, and `caution`.

NOTICE! Thank you for noticing this **new notice**! Your noticing it has been noted, and *will be reported to the authorities*!

NOTICE!

Thank you for noticing this **new notice**! Your noticing it has been noted, and *will be reported to the authorities*!

Next

```
cat("sdaafsadfsdafasd")
```

```
sdaafsadfsdafasd
```

```
1+1
```

```
[1] 2
```

As demonstrated by Gruber et al. (2018) we can do the following. see also Mijangos et al. (2022)

dartRverse

Rationale

We have developed a new suit of packages called “dartRverse” packages, that will replace the previous dartR package, which soon will be no longer supported. The new suit of packages are meant to be a 1:1 replacement and only in some instance very minor changes will be needed to update previous code. For example we reworked on the “output” of all functions, now being able to save figures in specified folders, which can be easier recovered and manipulated for further tweaking. The main reasons for splitting dartR into several packages and developing the dartRverse suit of packages was difficulty to maintain dartR, because of limits imposed by CRAN, but also limits due the the long time to test new function, when compiling the package. Therefore we were forced to split the package into several smaller packages. This has several advantages and in the best of all cases no disadvantages.

- easier maintenance
- faster development
- less dependencies on other packages
- easier to become a contributor
- have your own package developed that can be branded as part of the dartRverse

The main paradigm was also that for our users nothing (or at as little as possible) needed to change and all existing code and scripts should still work. Also the installation process was meant to be straightforward and finally the coexistance between dartR and dartRverse packages should be possible (for the interim until dartR will be no longer supported).

Installation

install dartRverse

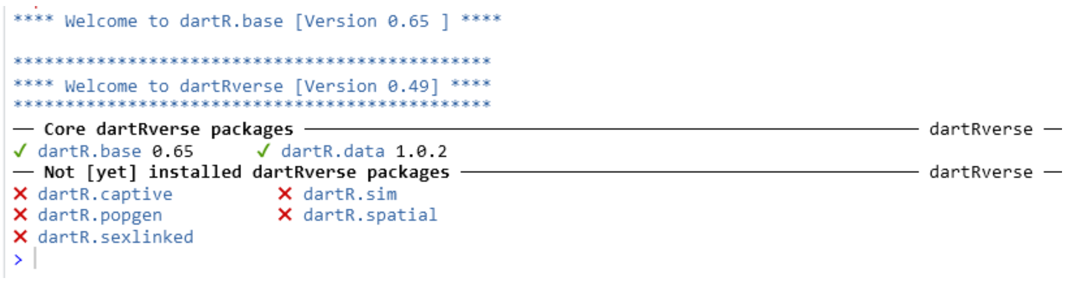
```
#installs the necessary bioconductor packages
install.packages("devtools")
install.packages("BiocManager")
BiocManager::install("SNPRelate")

#install dartRverse (dartRverse) & core (dartR.base, dartR.data)
install.packages("dartRverse")
```

The dartRverse package is the first package to install and its only purpose is to support the installation of the other dartRverse packages. It will also be the first port of call to provide links to tutorials and other documentation.

If everything works well this should install two more packages with form the core version of dartRverse, nameley dartR.base and dartR.data. Those core packages have all the main function of old dartR package that deals with input, conversion, reporting and filtering of data. Also some base function to analysse data (e.g. PCoA, Fst) are included.

Once the dartRverse package has been installed we can load the package to check which part of the dartRverse has been installed.

```
library(dartRverse)
```



```
**** Welcome to dartR.base [Version 0.65 ] ****

***********************************************
**** Welcome to dartRverse [Version 0.49] ****
***********************************************
── Core dartRverse packages ──────────────────────────────────── dartRverse ──
✓ dartR.base 0.65       ✓ dartR.data 1.0.2
── Not [yet] installed dartRverse packages ───────────────────── dartRverse ──
✗ dartR.captive         ✗ dartR.sim
✗ dartR.popgen          ✗ dartR.spatial
✗ dartR.sexlinked
> |
```

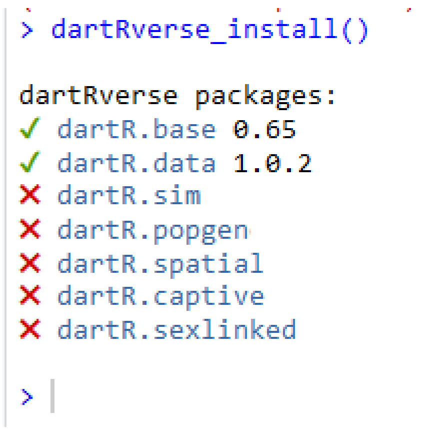
install additional packages

Now we can install the additional packages that are part of the dartRverse. Depending on your needs you can install all of them or only the one you are interested.

For example if you are interested in additional functions to analyse population structure (e.g. run STRUCTURE or faststructure) you can install the dartR.popgen package. If you are interested in functions that support the simulation of data you can install the dartR.sim package.

You can check which packages are avaialbel and which of them you have installed by typing:

```
dartRverse::dartRverse_install()
```



```
> dartRverse_install()

dartRverse packages:
✓ dartR.base 0.65
✓ dartR.data 1.0.2
✗ dartR.sim
✗ dartR.popgen
✗ dartR.spatial
✗ dartR.captive
✗ dartR.sexlinked

> |
```

The currently available packages are:

- dartR.sim (simulate SNP data)
- dartR.popgen (run population genetic analysis)
- dartR.spatial (run landscape genetic analysis)
- dartR.captive (estimate relatedness, support captive breeding)

- dartR.sexlinked (identify sexlinked markers, not ready yet)

So to install the dartR.sim simply type:

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

Github repositories

We make all of our packages available via CRAN. The reason for this is that CRAN packages follow a stringent testing before they are allowed to be uploaded to CRAN and therefore are more likely to contain less errors then packages that are available on other repositories. Nevertheless we also make our packages available during development via github.

You can find the repositories under: https://github.com/green-striped-gecko/dartR.base [for the dartR.base package] and equivalent for the other packages.

The reason why you might want to install a package from github is that you want to use the latest version of the package. However, you should be aware that the packages on github are not tested and therefore might contain errors. Also the packages on github might be updated on a daily basis and therefore might change without notice.

We use different branches and they are reflecting different stages of development and majurity.

- main (the main branch, which is equivalent to the current CRAN version)
- dev (development branch, which has the latest functions that might be in the next CRAN version, but have not been tested yet)
- dev_name (these are branches of our main developers and are used for testing and development of new functions. Installing functions from here might cause problems and should only be done if you know what you are doing)

dartRverse supports the installation of github version of the packages using the following syntax:

```
dartRverse_install(package = "dartR.base, repo = "github", branch =
"dev")
```

This install the dev branch of dartR.base from CRAN. All main and dev branches are tested if they can be installed (and some additional error checks via): https://github.com/green-striped-gecko/dartRverse

Please note that you should provide the package repository (github/cran), the branch (main, dev,dev_name) and version number in case you want to report a bug. You can use the github methods to report issues or use our google group: https://groups.google.com/d/forum/dartr.

Using legacy dartR

For whatever reason you might want to use legacy dartR, instead of the dartRverse packages [e.g. for initial testing]. The good news is that both packages can be installed next to each other, but you need to make sure you detach the other package. This can be done in Rstudio via the package tab by unticking the packages. Please note the order you unclick packages is important. So first the non-core packages (dartR.sim, dartR.popgen, dartR.captive, dartR.spatial), then dartRverse followed by the core packages dartR.base and finally dartR.data. If you do not follow the correct order you get a warning message and it will not detach the package. If you want to use code type:

```
detach("package:dartR.sim", unload = TRUE)
detach("package:dartR.sexlinked", unload = TRUE)
detach("package:dartR.spatial", unload = TRUE)
detach("package:dartR.captive", unload = TRUE)
detach("package:dartR.popgen", unload = TRUE)
detach("package:dartRverse", unload = TRUE)
detach("package:dartR.base", unload = TRUE)
detach("package:dartR.data", unload = TRUE)
```

Now you can load the legacy dartR package and use it as before.

```
library(dartR)
```

To unload dartR you can use the Packages tab as described before or the following code:

```
detach("package:dartR", unload = TRUE)
detach("package:dartR.data", unload = TRUE)
```

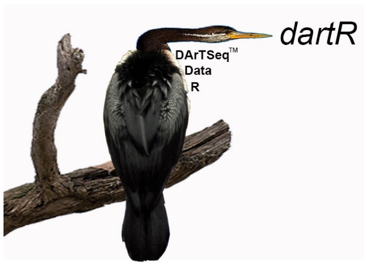
using dartRverse

To use dartRverse you can simply load the package and use it as before.

```
library(dartRverse)
```

1 Basic filtering

by Arthur Georges, Bernd Gruber & Luis Mijangos



1.1 Session 1: Basic Filtering

1.1 Overview

Calling SNPs in genotyping by sequencing is not an exact science. Judgements need to be made at various points in the workflow to increase the likelihood of a correct call at a particular locus. Diversity Arrays Pty Ltd (DArT) 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 to increase the reliability of the data retained in your SNP or SilicoDArT dataset. For each filter parameter offered by dartR, we provide both a report function and a filtering function (e.g. gl.report.reproducibility us associated with `gl.filter.reproducability`). The reporting function provides descriptive statistics for the parameter of choice. Inspection of these outputs will assist you to identify appropriate filter thresholds. Hence, as a standard workflow, it is a good idea to run the gl.report functions in advance of applying each gl.filter function to provide a foundation for selecting thresholds. Several filters are available to improve the quality of the data represented in your genlight object. Some of these are specific to Diversity Arrays data (e.g. `gl.filter.reproducibility`), but most will work on any data provided they are in the appropriate genlight format. The basic ones, in no particular order, are:

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

`gl <- gl.filter.taglength()` |filter out loci for which the tag length is less that a threshold |
`gl <- gl.filter.overshoot()` | filter out loci where the SNP location lies outside the trimmed sequence tag |

The order of filtering can be important and requires some thought. As an example, 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. The filtering steps may also change depending on the analysis you intend to run. There is no one-fits-all approach to SNP filtering.

1.1 Filtering on reproducibility

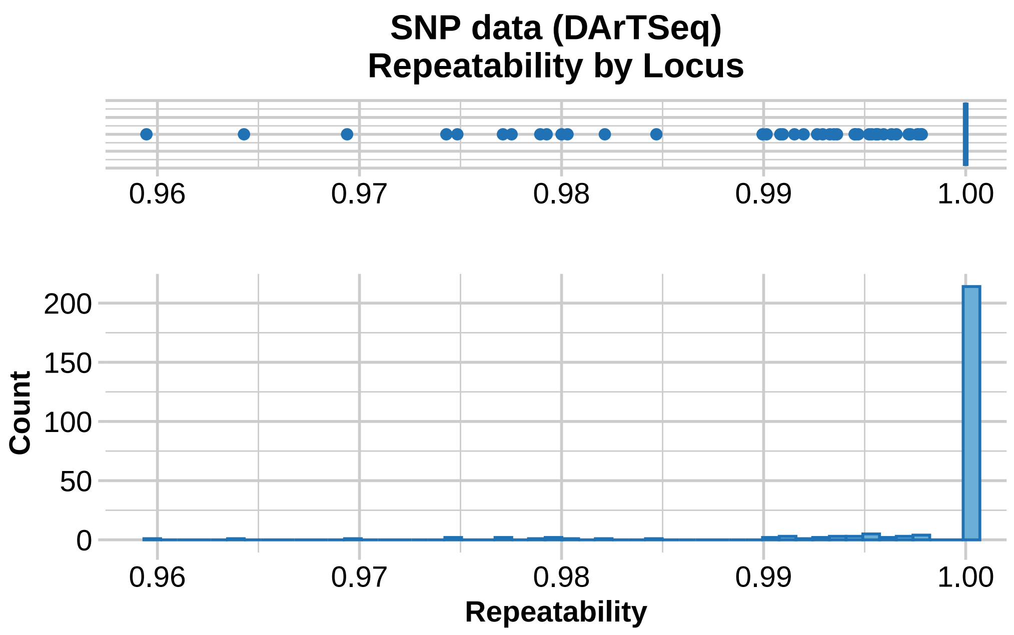
As an additional quality control, a selection of samples is processed twice by Diversity Arrays Technology using independent adaptors from DNA through to allelic calls. Scoring consistency of these technical replicates is referred to as reproducibility. Let's examine the distribution of reproducibility measures (RepAvg) in our dataset.

Loading the package (assuming you have installed dartRverse - see chapter dartRverse)

```
library(dartRverse)
```

```
gl.report.reproducibility(testset.gl , verbose = 1)
```

```
Starting gl.report.reproducibility
  Reporting Repeatability by Locus
  No. of loci = 255
  No. of individuals = 250
    Minimum      :  0.959459
    1st quartile :  1
    Median       :  1
    Mean         :  0.9981525
    3r quartile  :  1
    Maximum      :  1
    Missing Rate Overall:  0.12
```



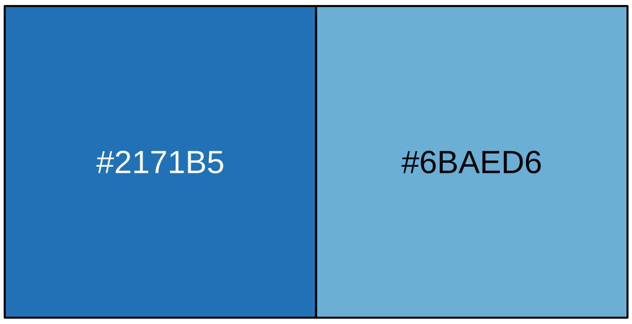
```
   Quantile Threshold Retained Percent Filtered Percent
1      100%  1.000000      214    83.9       41    16.1
2       95%  1.000000      214    83.9       41    16.1
3       90%  1.000000      214    83.9       41    16.1
4       85%  1.000000      214    83.9       41    16.1
5       80%  1.000000      214    83.9       41    16.1
6       75%  1.000000      214    83.9       41    16.1
```

```
7       70%  1.000000      214    83.9        41    16.1
8       65%  1.000000      214    83.9        41    16.1
9       60%  1.000000      214    83.9        41    16.1
10      55%  1.000000      214    83.9        41    16.1
11      50%  1.000000      214    83.9        41    16.1
12      45%  1.000000      214    83.9        41    16.1
13      40%  1.000000      214    83.9        41    16.1
14      35%  1.000000      214    83.9        41    16.1
15      30%  1.000000      214    83.9        41    16.1
16      25%  1.000000      214    83.9        41    16.1
17      20%  1.000000      214    83.9        41    16.1
18      15%  0.997674      217    85.1        38    14.9
19      10%  0.994536      230    90.2        25     9.8
20       5%  0.984694      243    95.3        12     4.7
21       0%  0.959459      255   100.0         0     0.0
Completed: gl.report.reproducibility
```

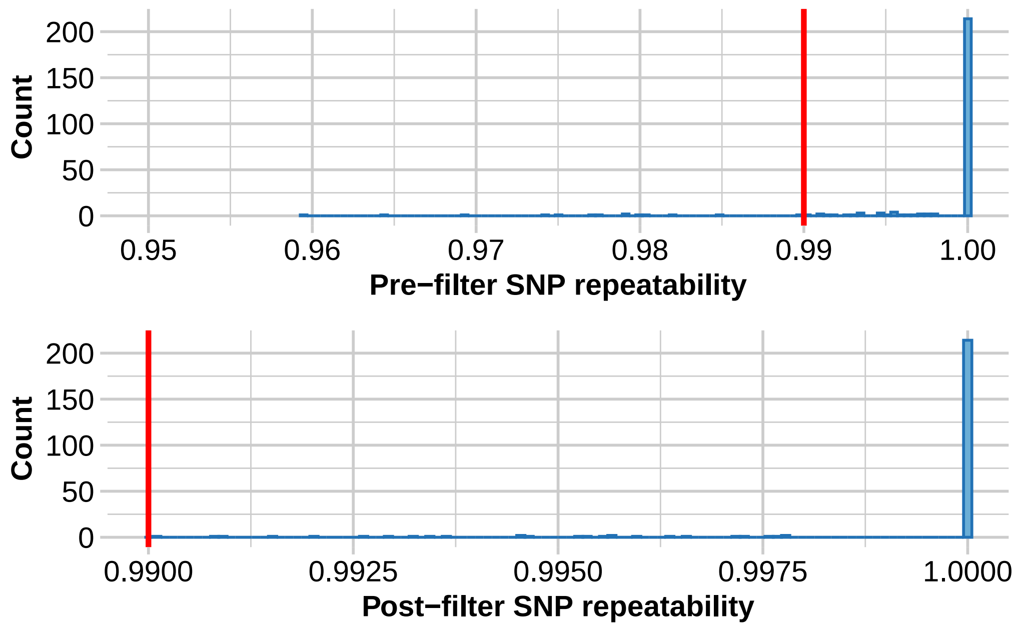
We now filter on selecting a threshold value of 0.99.

```
gl <- gl.filter.reproducibility( testset.gl, threshold=0.99, verbose = 1)
```

```
Starting gl.select.colors
  Warning: Number of required colors not specified, set to 9
  Library: RColorBrewer
  Palette: brewer.pal
  Showing and returning 2 of 9 colors for library RColorBrewer : palette Blues
```



```
Completed: gl.select.colors
Starting gl.filter.reproducibility
```



```
Completed: gl.filter.reproducibility
```

Only 14 loci out of 255 were removed.

Exercise

1. Just in case you have accidentally modified the genlight object gl, recreate it by copying it from `testset.gl`.

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

2. Filter gl using the filters listed above. Request a report first to inform your choice of threshold. Be sure to set plot=TRUE to examine the distribution of each parameter and optionally `smearplot=TRUE` to examine the smear plot.

1.1 Filtering on call rate

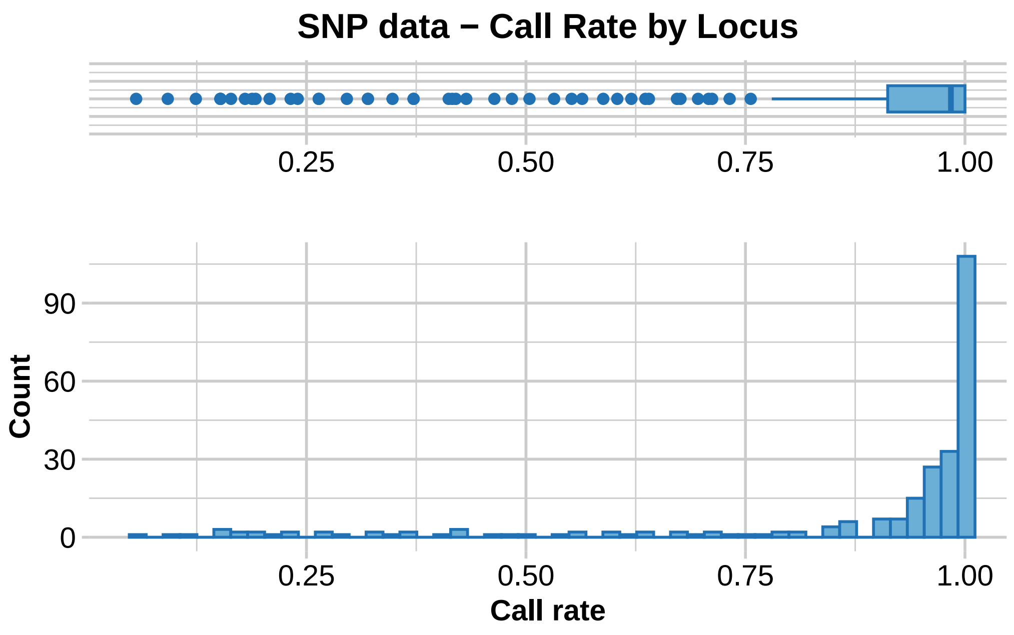
A filter for Call Rate can be applied to loci and to individuals. When filtering on loci, only those for which the associated SNP is called in at least a specified proportion will be retained. When filtering on individuals, only those individuals for which a specified percentage of loci are scored for a SNP polymorphism will be retained.

Filtering on Loci

Recall that Call Rate for SNPs can arise from two sources. The first source is where a missing value arises because the sequence tag bearing the target SNP cannot be amplified – there has been a mutation at one or both of the restriction sites. The second source of missing values is where the read depth is insufficient to make a reliable call on the SNP. Either way, the SNP is not called and is recorded as NA. For presence-absence data (i.e. SilicoDArT), the sequence tag is recorded as having been amplified (presence) or not (absence). A missing value arises when it is not possible to determine if the sequence tag has been amplified or not, so in that sense it is true missing data. A first step in filtering on Call Rate is to examine the distribution of Call Rates across loci. We use the function `gl.report.callrate` to yield the following output and graph.

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

```
Starting gl.report.callrate
  Processing genlight object with SNP data
  Reporting Call Rate by Locus
  No. of loci = 255
  No. of individuals = 250
    Minimum      :  0.056
    1st quartile :  0.912
    Median       :  0.984
    Mean         :  0.8765804
    3r quartile  :  1
    Maximum      :  1
    Missing Rate Overall:  0.1234
```



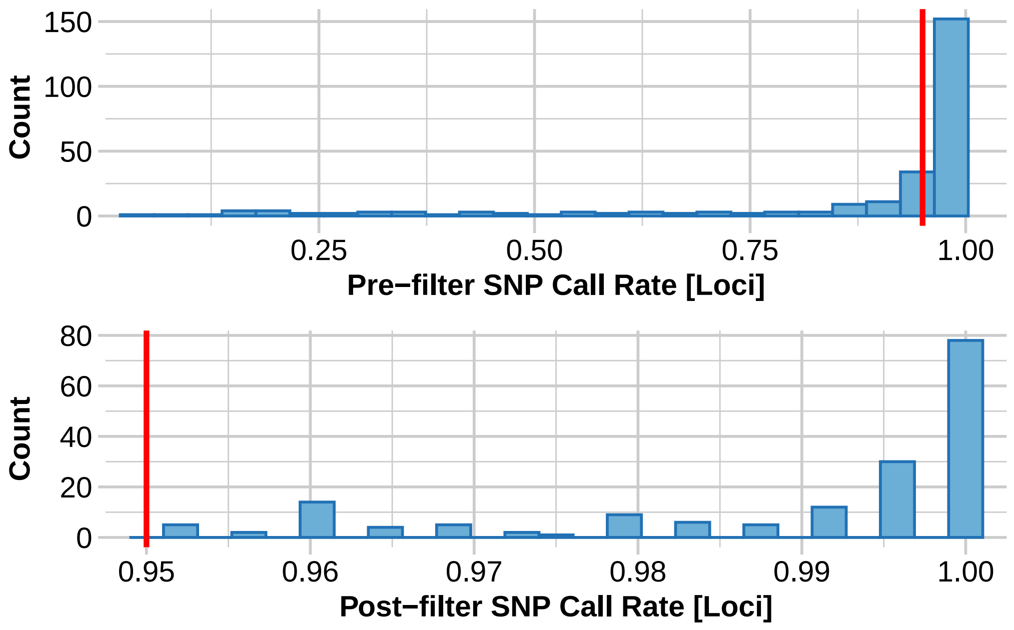
```
Completed: gl.report.callrate
```

Here you can see that the call rate for most loci is close to 100%, but that there is a tail of loci for which the call rate is exceptionally poor. In this case, we might choose to filter out loci for which the call rate is less than 95% (0.95).

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

```
Starting gl.filter.callrate
  Processing genlight object with SNP data
  Warning: Data may include monomorphic loci in call rate
```

```
                  calculations for filtering
  Recalculating Call Rate
  Removing loci based on Call Rate, threshold = 0.95
```



```
Completed: gl.filter.callrate
```

The results of the filtering are shown as a before-after plot. For most purposes, this filtering of poorly called loci is likely to be satisfactory.

Filtering on Individuals

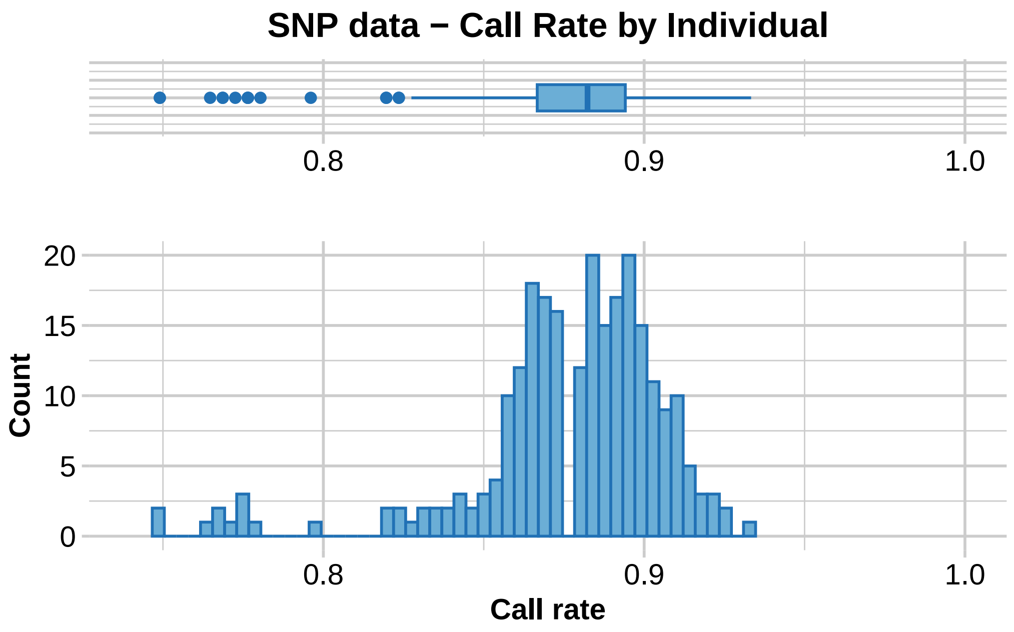
A second way of filtering on Call Rate is to remove individuals that have sequenced particularly poorly. This may occur if the associated samples are degraded in comparison with other samples. We again first report the Call Rate, this time for individuals.

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

```
Starting gl.report.callrate
  Processing genlight object with SNP data

  Reporting Call Rate by Individual
  No. of loci = 255
  No. of individuals = 250
    Minimum      :  0.7490196
    1st quartile :  0.8666667
    Median       :  0.8823529
    Mean         :  0.8765804
    3r quartile  :  0.8941176
    Maximum      :  0.9333333
    Missing Rate Overall:  0.1234

Listing 30 populations and their average CallRates
  Monitor again after filtering
```

```
         Population CallRate  N
1      EmmacBrisWive   0.8839 10
2      EmmacBurdMist   0.8808 10
3      EmmacBurnBara   0.8859 11
4      EmmacClarJack   0.8596  5
5      EmmacClarYate   0.8769  5
6      EmmacCoopAvin   0.7682 10
7     EmmacCoopCully   0.9122 10
8      EmmacCoopEulb   0.8702 10
9     EmmacFitzAllig   0.8973 10
10     EmmacJohnWari   0.8929 10
11     EmmacMaclGeor   0.8806 11
12     EmmacMaryBoru   0.8843  6
13     EmmacMaryPetr   0.8892  4
14      EmmacMDBBowm   0.8824 10
15      EmmacMDBCond   0.8855 10
16      EmmacMDBCudg   0.8878 10
17      EmmacMDBForb   0.8766 11
18      EmmacMDBGwyd   0.9050  9
19      EmmacMDBMaci   0.8773 10
20 EmmacMDBMurrMung   0.8890 10
21      EmmacMDBSanf   0.8914 10
22     EmmacNormJack   0.8725  6
23     EmmacNormLeic   0.8863  1
24     EmmacNormSalt   0.8706  1
25     EmmacRichCasi   0.8757 10
26         EmmacRoss   0.8706 10
27     EmmacRussEube   0.8612 10
28      EmmacTweeUki   0.8773 10
29     EmsubRopeMata   0.8345  5
30     EmvicVictJasp   0.8361  5

Listing 20 individuals with the lowest CallRates
  Use this list to see which individuals will be lost on filtering by individual
  Set ind.to.list parameter to see more individuals
   Individual  CallRate
1     AA063722 0.7490196
2     AA063726 0.7490196
3     AA063732 0.7647059
4     AA063720 0.7686275
5     AA063712 0.7686275
6     AA063708 0.7725490
7     AA063718 0.7764706
8     AA063710 0.7764706
9     AA063714 0.7764706
10    AA063716 0.7803922
11    AA032760 0.7960784
12    UC_00210 0.8196078
13    UC_00259 0.8196078
14    AA018494 0.8235294
15    UC_00206 0.8235294
16    AA019164 0.8274510
17    UC_00209 0.8313725
```

```
18    UC_00254 0.8313725
19    AA019159 0.8352941
20   UC_00126c 0.8352941

)
```



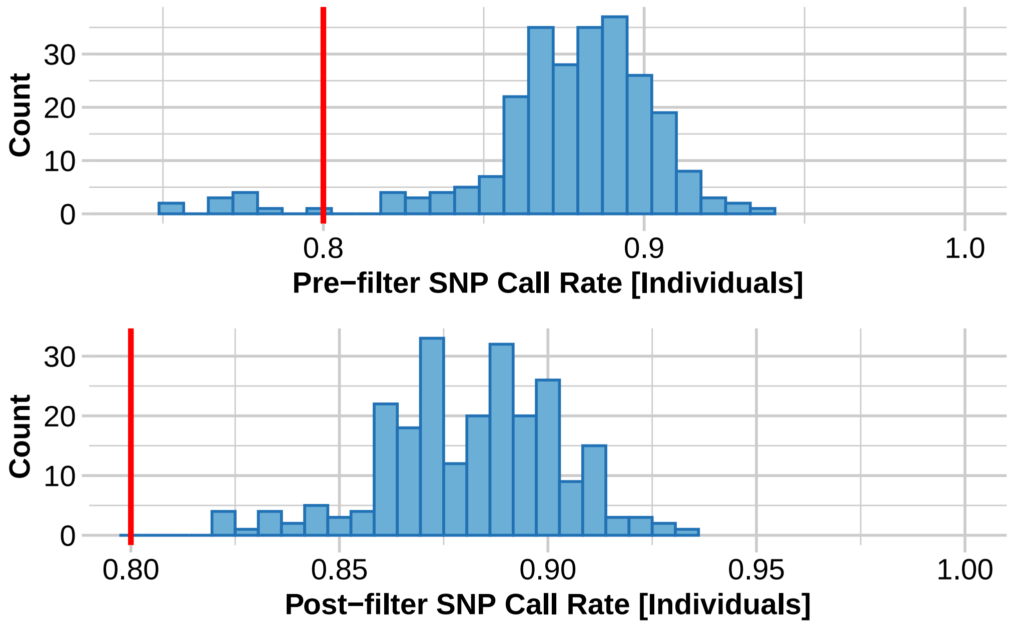
```
Completed: gl.report.callrate
```

The output will include a list of populations and the Call Rate averaged across individuals, and a list of the top worst individuals in terms of their call rate. This will allow you to make a reasoned judgement on the impact of filtering out individuals. The graph tells the story. In the absence of information to the contrary, a threshold of 80% (0.80) would seem appropriate for filtering individuals on Call Rate.

We execute the filter with a threshold of 0.80.

```
gl <- gl.filter.callrate(testset.gl, threshold=0.80, method='ind')
```

```
Starting gl.filter.callrate
  Processing genlight object with SNP data
  Warning: Data may include monomorphic loci in call rate
                    calculations for filtering
  Recalculating Call Rate
  Removing individuals based on Call Rate, threshold = 0.8
  Individuals deleted (CallRate <=  0.8 ):
AA032760[EmmacMDBMaci], AA063718[EmmacCoopAvin], AA063720[EmmacCoopAvin], AA063722[EmmacCoopAv
```



```
  Note: Locus metrics not recalculated
  Note: Resultant monomorphic loci not deleted
Completed: gl.filter.callrate
```

This statement filters out individuals with a Call Rate across loci of less than 80%. It conveniently lists the individuals that have been removed and the populations to which they belong so you can assess the impact of the filtering.

1.1 Filter Seconaries

Sequence tags can contain more than one callable SNP marker. Because of their close proximity, these multiple SNPs within a single sequence tag (referred to in dartR as 'secondaries' are likely to be strongly linked (inherited together). This is problematic for many analyses, so one might wish to filter out the multiple SNPs to leave only one per sequence tag. Diversity Arrays Technology include multiple SNPS in a single sequence tag each as separate records in the data provided with your report. The decision becomes, which SNP to retain, and which SNPs to discard. One strategy is to leave the filtering of secondaries until last, so that you are considering only those secondaries that have survived the earlier filtering on call rate, reproducibility and read depth. You can then choose one from the surviving secondaries at random (method='random') or based on comparisons of reproducibility (RepAvg) and polymorphism information content (PIC) (method='best'). The call is:

```
gl.report.secondaries(gl)
```

```
Starting gl.report.secondaries
  Processing genlight object with SNP data
Counting ....
  Warning: No loci with secondaries, no plot produced
  Total number of SNP loci scored: 255
    Number of secondaries: 0
Completed: gl.report.secondaries
```

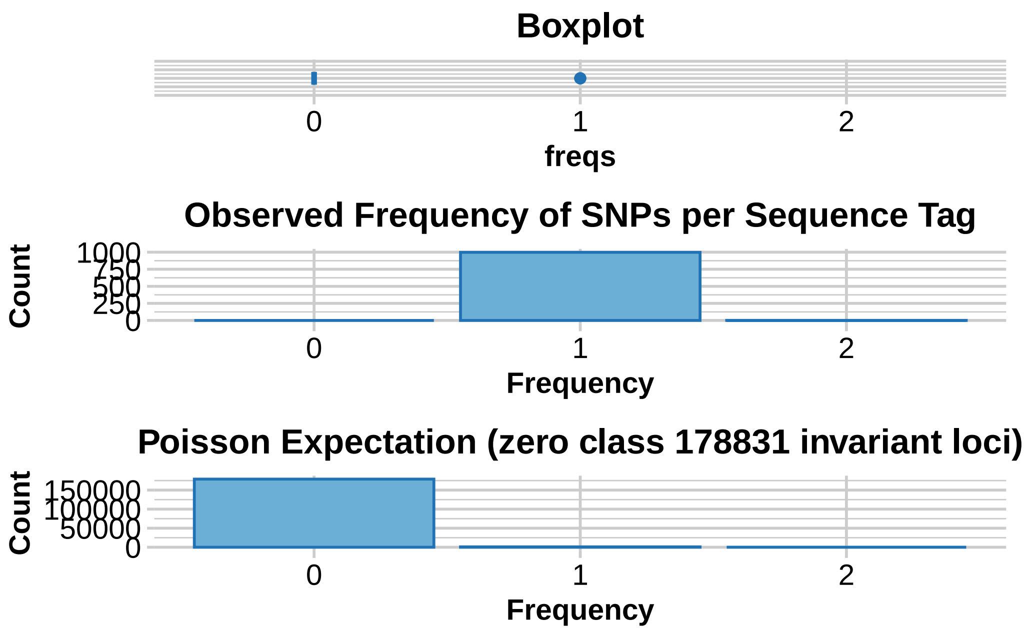
```
               Param       Value
1       n.total.tags   255.00000
2 n.SNPs.secondaries     0.00000
3   n.invariant.tags          NA
4 n.tags.secondaries          NA
5          n.inv.gen 15224.00000
6       mean.len.tag    60.70196
7         n.invariant          NA
8             Lambda          NA
```

The output is a bit more complex than one might expect. The total number of SNPs scored in the dataset gl is 119,414 residing on 70,942 sequence tags. So there are many sequence tags with more than one SNP (indeed 28,769). Filtering on secondaries will remove 48,477 SNPs. The function models the frequency spectrum of SNPs per sequence tag with a zero truncated Poisson Distribution (strictly the distribution is a compound Poisson as lamda varies) and uses that distribution to estimate the unknown zero class – the number of sequence tags expected to contain no SNP (that is, the invariant sequence tags). This value of 7,317,732 can be used to correct the denominator for estimates of heterozygosity.

This estimation is best shown in a graph. (**Arthur?** the original example with gl does not give a plot)

```
gl.report.secondaries(bandicoot.gl, verbose=1)
```

```
Starting gl.report.secondaries
The column 'TrimmedSequence' was not found in loc.metrics
 Mean tag length assumed to be 69
```



```
  Total number of SNP loci scored: 1000
   Number of sequence tags in total: 999
   Estimated number of invariant sequence tags: 178831
   Number of sequence tags with secondaries: 1
   Number of secondary SNP loci that would be removed on
            filtering: 1
   Number of SNP loci that would be retained on filtering: 999
   Number of invariant sites in sequenced tags: 67931
   Mean length of sequence tags: 69
   Total Number of invariant sites (including invariant sequence
            tags): 12407270
Completed: gl.report.secondaries
```

```
                Param        Value
1        n.total.tags 9.990000e+02
2 n.SNPs.secondaries 1.000000e+00
3   n.invariant.tags 1.788310e+05
4 n.tags.secondaries 1.000000e+00
5           n.inv.gen 6.793100e+04
6        mean.len.tag 6.900000e+01
7         n.invariant 1.240727e+07
8              Lambda 5.570726e-03
```

Note that the estimate of the zero class involves an iterative process that does not always converge to a satisfactory solution for lamda. In this case it did. Note also that the estimate of the zero class can have a very substantial error associated with it, especially if the count for class 0 exceeds the count for the class 1. Useful, but not infallible. Having examined the report, filtering out the secondaries is done using

```
gl <- gl.filter.secondaries(gl,method="random")
```

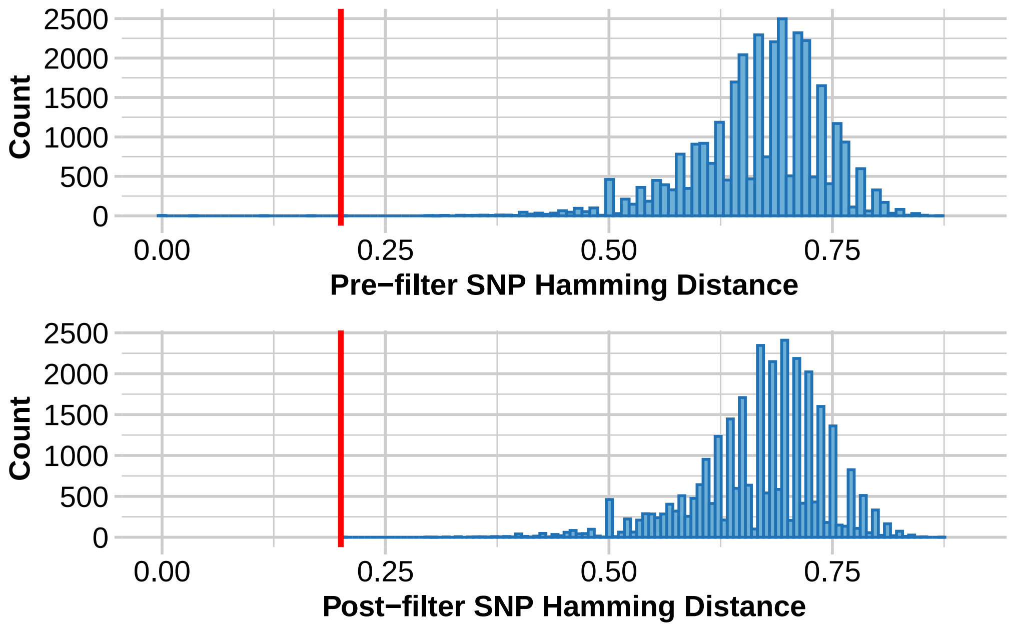
```
Starting gl.filter.secondaries
  Processing genlight object with SNP data
  Selecting one SNP per sequence tag at random
Completed: gl.filter.secondaries
```

1.1 filtering on Hamming Distance

Hamming distance is a measure of sequence similarity between sequence tags. If two sequence tags (69 bp) differ by only a couple of base pairs then suspicion is aroused as to whether they have arisen by sequencing error. Diversity Arrays Technology have already filtered on a Hamming distance of 3 bp. Rarely you might want to go further, and the `gl.filter.hamming` function gives you this option.

```
gl <- gl.filter.hamming(testset.gl)
```

```
Starting gl.filter.hamming
  Processing genlight object with SNP data
  Filtering loci with a Hamming Distance of less than 0.2
```



```
Completed: gl.filter.hamming
```

Exercise

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

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

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

Exercise

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

Note that many functions have a `mono.rm` and `recalc` parameters that allow you to remove monomorphic loci or recalculate metrics on the fly. It is not a fatal error to forget to recalculate the locus metrics because dartR scripts will detect if they have not been recalculated and rectify this before they a particular locus metric is needed.

1.1 Where have we come?

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

- Filter on call rate, reproducibility, secondaries, read depth and hamming distance.
- Recalculate locus metrics after deleting individuals or populations as part of the filtering process.

- Filter out resultant monomorphic loci.

Having played with the various filters, you should also have a good appreciation of how to use the `gl.report` functions to select appropriate thresholds for filtering and how to introduce a sequence of filtering steps into your workflows.

1.2 Session 2: Nuances of Filtering

1.2 Pre-dartR Filtering

Diversity Arrays Technology

Sequences generated by Diversity Arrays Technology are processed using proprietary analytical pipelines before the report (containing the SNP and SilicoDArT markers) is provided to the client and the data are passed to dartR.
Poor quality sequences are first filtered, applying more stringent selection criteria to the barcode region than to the rest of the sequence (minimum barcode Phred score 30, pass percentage 75; minimum whole-read Phred score 10, pass percentage 50). In that way, assignment of the sequences to specific samples in the sample disaggregation step is very reliable. In a report based on full sequence volume (high), approximately 2,500,000 (+7%) sequences per sample are identified and used in marker calling.
These sequences are truncated to 69 bp (including some adaptor sequence where the fragments were shorter than 69 bp) and aggregated into clusters by the DART fast clustering algorithm. A Hamming distance threshold of 3 bp is used as a threshold for identifying distinct sequence, taking advantage of the fixed fragment length.
The sequences are then error-corrected using an algorithm that corrects a low-quality base (Phred score <20) to a corresponding high-quality singleton tag (Phred score >25); where there is more than one distinct high-quality tag, the sequence with the low-quality base is discarded. Identical sequences are then collapsed. These error-corrected sequences are analysed using DART software (DArT GS14 pipeline) to output candidate SNP markers. SNP markers are identified within each cluster by examining parameters calculated for each sequence across all samples – primarily average and variance of sequencing depth, the average counts for each SNP allele and the call rate (proportion of samples for which the marker is scored).
Where three sequences survive filtering to this point, the two variants with the highest read depth are selected. The final average read depth per locus depends on the genome size, the degree of representation delivered by the chosen restriction enzymes and the sequencing intensity (sequence volume).
As an additional quality control, a selection of samples (30 to 100% of samples depending on the number of plates submitted) is processed twice using independent adaptors from DNA through to allelic calls. These are technical replicates. Scoring consistency (reproducibility or more strictly, repeatability) is used as the main selection criterion for high quality/low error rate markers. The DART analysis pipelines have been tested against hundreds of controlled crosses to verify Mendelian behaviour of the resultant SNPs and fine tune the parameter thresholds of the DArT GS14 pipeline as part of the commercial operations. The bottom line is that, although analytical pipelines are proprietary and commercial in confidence, a substantial amount of filtering of sequence tags is undertaken before you receive the SNP and SilicoDArT markers and report from Diversity Arrays Technology, filtering that ensures quality outcomes in terms of the veracity of the resultant SNPs. Package dartR picks up the analysis at this point. The additional filtering that you may choose to undertake using dartR depends upon the research questions and other considerations as outlined below.

Stacks

A full protocol for calling SNPs from reduced representation data has been published – Rochette, N. & Catchen, J. (2017). Deriving genotypes from RAD-seq short-read data using Stacks. Nature Protocols, 12(12), 2640–2659. Please refer to that article and the Stacks V2 Manual.

Package dartRverse picks up the analysis after the SNPs have been called in Stacks. The additional filtering that you may choose to undertake using dartR depends upon the research questions and other considerations as outlined below.

1.2 Filtering Order

It is often unclear as to what order the filtering steps should take in your dartR workflow. Does one filter on call rate by individual first then call rate by locus or the other way around? There is no correct answer to this, as it depends on whether you value retaining loci over retaining individuals. Usually, you would not want to lose individuals from your dataset, so filtering out those loci will low call rates would come first, filtering on individuals second, though this is not always the case. A starting point for considering a workflow might be:

1. Optionally filter on read depth if the Diversity Arrays threshold is considered too lenient.
2. Filter out loci with a reproducibility below a particular threshold (say 0.98)
3. Filter out loci with a call rate below a particular threshold (say 0.95)
4. Filter out individuals with a call rate below a particular threshold (say 0.80)
5. Optionally filter on Hamming Distance if the Diversity Arrays threshold is considered too lenient.
6. Filter out secondaries (all but one SNP retained per sequence tag)
7. Optionally filter for monomorphs created by the removal of earlier individuals.

An example of a filtering sequence might be:

```
gl <- gl.filter.rdepth(gl)
gl <- gl.filter.reproducibility(gl)
gl <- gl.filter.callrate(gl, method="loc")
gl <- gl.filter.callrate(gl, method="ind")
gl <- gl.filter.secondaries(gl)
gl <- gl.filter.monomorphs(gl)
```

1.2 Filtering Thresholds

Strategy

The take-home message from this section is to be careful not to over-filter. The objective is to get an appropriate balance between signal to noise ratio, not to eliminate noise altogether at the expense of also taking out some signal. This balance will depend on sequence intensity (volume in the context of genome size), sample quality and most importantly, downstream applications.
There is no right answer. A good strategy is to undertake an exploratory analysis first, whereby you experiment with filtering. Perhaps start with very stringent filtering, examine the impact on the final analysis, then progressively reduce the stringency to select the minimal filtering regime that still delivers a stable outcome. Play with your data to get a feel

for the balance between signal to noise ratio, in the context of the analyses you propose to subsequently undertake.

What is sequence volume?

Three services are routinely offered by Diversity Arrays Technology, differing in the volume of sequence generated. For genomes greater in size than 1 Gbp, sequence volume of 2.5 million is recommended (high). For genomes ranging from 300 Mbp to 1 Gbp, sequence volume of 1.2 million is recommended (medium). For genomes less than 300 Mbp, sequence volume of 800,000 is recommended (low). The average read depth of the sequence tags depends upon the genome size, the sequencing volume (low, medium, high), the proportion of the genome that is represented after the restriction enzymes are applied and size selection, and the proportion of mis-targeted sequence in the sample (e.g. arising from bacterial contamination). Average read depth per sequence tag is calculated by dartR using `gl.report.rdepth()`.

Average read depth can be obtained using:

```r
mean(gl@other$loc.metrics$rdepth)
```

```
[1] 16.75202
```

This is reported also by `gl.report.basics()`.

1.2 Filtering on Reproducibility

As mentioned above, a proportion of samples is processed twice, using independent adaptors, from DNA through to allelic calls. These are technical replicates and their scoring consistency (repeatability) is used as the main selection criterion for high quality/low error rate markers. Reproducibility values may depend on the number of plates run in a particular service, because the proportion of samples selected for technical replication will vary (usually 60-70%, but as high as 100% (partial plate) or as low as 30%).

While it is tempting to filter on 100% reproducibility, this is typically over-kill. One needs to appreciate that the reproducibility of technical replicates can depend on the quality of the input DNA. If it is contaminated, say with bacterial DNA or a blood parasite, then the target sequence volume can be dramatically reduced, the error rate inflated and the reproducibility technical replicates across samples for a locus can be substantially affected. Furthermore, the presence of inhibitors can affect the efficiency of restriction enzymes in a context specific way, again leading to inflation of the error rate and systematic differences between technical replicates.

These factors, contamination and presence of PCR inhibitors, have a greater influence on heterozygous than homozygous sites, and so differentially affect loci that are more polymorphic than others. Furthermore, experience shows that 90% of the resultant error rates occur in 10% of samples; filtering too stringently will lead to loss of loci that are otherwise reproducible for 90% of samples.

Diversity Arrays Technology have already used technical replicates in their earlier pipelines to select reliable markers to yield a basic level of reproducibility in the data provided to you.

So by all means filter on reproducibility, but consider the consequential loss of loci and do not filter so stringently as to decimate your locus count, as this can lead to systematic biases and unnecessary loss of informative loci. You should carefully consider the plot of the distribution of reproducibility values in the exploratory analysis (gl.report.reproducibility).

1.2 Filtering on Call Rate

When filtering loci on Call Rate, one needs to be aware failure to call a SNP at a given locus is typically a result of a mutation at one or both of the restriction sites. That is, a missing value typically represents a null allele, captured in the companion SilicoDArT dataset. In the SNP dataset, highly polymorphic regions of the genome, those with an abundance of informative SNPs are also most prone to null alleles. Hence, SNPs in highly polymorphic regions will be differentially eliminated by overzealous filtering on Call Rate.

Again, by all means filter on Call Rate, but consider the consequential loss of loci and do not filter so stringently as to decimate your locus count, as this can lead to systematic biases (highly polymorphic loci differentially eliminated) and unnecessary loss of informative loci. Carefully consider the plot of the distribution of Call Rate values in the exploratory analyses (gl.report.callrate). Depending upon the subsequent analyses (see below), a Call Rate threshold of 0.90 might be considered appropriate for filtering loci.

There is no strict rule when filtering individuals on Call Rate. One should examine which individuals are likely to be lost for a given threshold and consider the impact of their loss on the subsequent analysis and interpretation. Of course, if individuals have exceptionally low Call Rates, their deletion would be highly recommended. If such individuals are of particular importance to the study, re-extraction and re-running the samples should be considered.

Some analyses are intolerant of any missing values (e.g. classical Principal Components Analysis). Removing all missing values is often too wasteful of data – one must either remove all loci that contain one or more missing values, or remove all individuals that contain one or more missing values. An alternative is to impute the missing values. There are a number of approaches to imputation.

- Global Imputation: Missing values are replaced by an estimate of the allelic value generated from the average allele frequencies for all the data taken collectively.
- Local Imputation: Missing values are replaced by an estimate of the allelic value generated from the average allele frequencies or from a Hardy-Weinberg model at that locus in the population from which the individual was drawn.
- Nearest Neighbour: Missing values are replaced by the value at the corresponding locus for the nearest neighbouring individual (nearest using Euclidean genetic distance)
- Random: Missing data are substituted by random values (0, 1 or 2).

The appropriate statement is:

```r
gl <- gl.impute(gl)
```

Exercise

Try this for yourself on genlight object `testset.gl` by first filtering on Call Rate by locus (`threshold = 0.90`) and reporting the Call Rate on the resultant data, then repeating the report after imputation.

1.2 Filtering on Read Depth

Read depth has a considerable influence on the veracity of SNP calls. Read depth estimates for sequence tags generated from high volume sequencing typically ranges from 3 to 1000 for single copy sequence. The low values arise because of chance variation in the coverage of particular bases. The high values arise because of the differential efficiency of the PCR steps. Under some circumstances, it might be sensible to push the lower threshold for read depth higher than has been used by the Diversity Arrays Technology in their report. Analyses that rely heavily on the accuracy of calls of heterozygotes may require a higher threshold for read depth, say 10x, for example. The call is

```
gl.filter.rdepth(gl,threshold=10)
```

Of course, how far you can push the read depth threshold depends very much on the sequencing volume of the service taken in the context of the genome size. You should examine the average read depth for your data in making this decision.

Influence of Planned Analyses

Possibly the most influential consideration on filtering is the purpose for which the filtered data set is to be used.

If you are planning to generate **high resolution linkage maps**, then highly stringent filtering is warranted, and Diversity Arrays Technology would adjust their pipelines accordingly upon request. Similarly, if you are contemplating an analysis of **relatedness or inbreeding**, then stringent filtering might be warranted. In any case, it would be wise to start out with high stringency and then progressively relax the stringency and examine the impact on outcomes.

If your focus is on identifying **sex linked markers**, which rely heavily on identifying markers that are heterozygous in XY individuals and homozygous in XX individuals, then a filtering threshold for read depth of 10x would be sensible. Sequencing volume in the context of genome size will provide an upper limit to how far you can push the read depth threshold, so in some cases, a high sequence volume service will need to be requested. Because the pipelines of Diversity Arrays Technology depend in part on Mendelian behaviour in the selection of SNP markers, and sex-linked loci do not behave in Mendelian fashion, you might also ask them to relax the stringency of some aspects of their filtering in generating the report.

The proof is in the pudding when searching for sex linked markers, so if you get the balance of considerations wrong, the cost lies in the number of false positives that will be generated, and additional work at the validation stage.

If your focus is on **Genome-wide Association Studies (GWAS)** or identifying **loci under selection**, then noise in the data will not associate with phenotype, and so filtering can be less stringent in order to maximize the chances of identifying promising associations.

Principal Coordinates Analysis PCA (and to a lesser extent, **Principal Components Analysis**, PCoA), rely on fully populated data matrices (no missing values). Classical PCA for example, cannot easily accommodate missing values. To overcome this, a balance must be struck between filtering on Call Rate and imputing the values of those missing values that remain. More stringent filtering on Call Rate, and less imputation; but more stringent filtering on Call Rate, more loss of potentially useful information or valuable samples. Refer to Georges et al. (2023, BioxRiv https://doi.org/10.1101/2023.03.22.533737) for further discussion of this point.

Conclusion

There is no hard and fast rule to guide decisions on filtering SNP datasets prior to a substantive analysis. The decisions are based on comparing the distribution of the parameters to be filtered (using one of the gl.report functions) and the purpose to which the filtered dataset is to be put. Sequencing volume can constrain options, and the likelihood of some level of contamination of samples or presence of inhibitors of the restriction enzymes can have a bearing on the decisions. It is important not to over filter because of the risk of introducing a level of systematic bias and because of the unnecessary loss of informative loci or individuals. An experimental approach is recommended, whereby different filtering regimes are tried (from stringent to less stringent) and checked for influence on analysis outcomes.

1.2 Where have we come?

The above Session was designed to give you an overview of the considerations that come into play when filtering your data using dartR. Having completed this Session, you should:

- Appreciate to some degree the pre-processing that is undertaken by Diversity Arrays Technology before you receive your report.
- Understand some of the considerations that come into play in deciding the order in which to apply dartR filters.
- Be able to apply the filtering in a nuanced manner to avoid over-filtering, which depends of course on the particular research question you are addressing as much as attributes of the data.

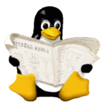
Exercise 1: Filtering the Platypus dataset

- Examine the attributes of the dataset platypus.gl paying particular attention to the presence of monomorphic loci and loci or individuals with all NA scores.
- Use the gl.report functions in combination with the gl.filter functions to devise and execute a series of filtering steps to yield a reliable set of SNP markers.
- How many SNP markers did you start with, and how many did you end up with?
- Which filtering steps generate monomorphic markers, and how does this influence when you filter for monomorphic markers?
- Check the number of individuals remaining in each of the populations defined for the dataset. Has your filtering potentially compromised subsequent analyses by decimating particular populations [maybe do a before-after comparison].
- Draft a section for a possible materials and methods section of a paper outlining your filtering strategy and its implementation.

Exercise 2: Filtering the Turtle dataset

- Repeat the analyses of Exercise 1 on the dataset testset.gl.

1.2 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
- O'Leary, S.J., Puritz, J.B., Willis, S.C., Hollenbeck, C.M. and Portnoy, D.S. (2018). These aren't the loci you're looking for: Principles of effective SNP filtering for molecular ecologists. Molecular Ecology 27: 3193-3206.

References

Gruber, Bernd, Peter J Unmack, Oliver F Berry, and Arthur Georges. 2018. "Dartr: An r Package to Facilitate Analysis of SNP Data Generated from Reduced Representation Genome Sequencing." *Molecular Ecology Resources* 18: 691–99. https://doi.org/10.1111/1755-0998.12745.

Mijangos, Jose Luis, Bernd Gruber, Oliver F Berry, Carlo Pacioni, and Arthur Georges. 2022. "dartR v2: An Accessible Genetic Analysis Platform for Conservation, Ecology and Agriculture." *Methods in Ecology and Evolution*. https://doi.org/10.1111/2041-210X.13918.