Basic usage of utility functions in GBScleanR

Tomoyuki Furuta

March 25, 2021

Contents

Introduction

The GBScleanR package has been mainly developed to conduct error correction on genotype data obtained via NGS-base genotyping methods such as RAD-seq and GBS. Nevertheless, several quality check procedure and data filtering are highly encouraged to improve correction acculacy. Therefore, this package also provide the functions for data quality check and filtering with some data visualization functions to help filtering procedure. In this document, we walk through the utility functions implemented in GBScleanR to introduce a basic usage. An error correction procedure for GBS data of a biparental population is described in another vignette.

Prerequisites

This package internally uses the following packages.
- ggplot2
- dplyr
- tidyr
- GWASTools
- SNPRelate
- SeqArray

To install them all, run the codes below.

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("GWASTools")
BiocManager::install("SNPRelate")
BiocManager::install("SeqArray")
install.packages("ggplot2")
install.packages("dplyr")
install.packages("tidyr")
```

You can install GBScleanR from the local source file with the following code.

```
install.packages("path/to/source/GBScleanR.tar.gz", repos = NULL, type = "source")
```

The code below let you install the package from the github repository.

```
if (!requireNamespace("devtools", quietly = TRUE))
    install.packages("devtools")
devtools::install_github("")
```

To load the package.

```
library("GBScleanR")
```

Data format conversion and object instantiation

The main class of the GBScleanR package is gbsrGenotypData which inherits the GenotypeData class in the GWASTools package. The gbsrGenotypeData class object has three slots: data, snpAnnot, and scanAnnot. The data slot holds genotype data as a gds.class object which is defined in the gdsfmt package while snpAnnot and scanAnnot contain objects storing annotation information of SNPs and samples, which are the SnpAnnotationDataFrame and ScanAnnotationDataFrame objects defined in the GWASTools package. See the vignette of GWASTools for more detail. GBScleanR follows the way of GWASTools in which a unique genotyping instance (genotyped sample) is called "scan".

As mentioned above, the gbsrGenotypeData class requires genotype data in the gds.class object which enable us quick access to the genotype data without loading the whole data on RAM. At the beginning of the processing, we need to convert data format of our genotype data from VCF to GDS. This conversion can be achi eved using gbsrVCF2GDS as shown below.

```
gbsrVCF2GDS(vcf_fn = "./data/gbs_nbolf2.vcf.gz", # Path to the input VCF file.
            out_fn = "./data/gbs_nbolf2.gds") # Path to the output GDS file.
```

Our sample dataset contains genotype information of 816 samples with 20224 markeres.
This size of data takes a few seconds for conversion.
The larger the data size, the longer the running time.

```
exec_time <- system.time({
  gbsrVCF2GDS(vcf_fn = "./data/gbs_nbolf2.vcf.gz", # Path to the input VCF file.
              out_fn = "./data/gbs_nbolf2.gds") # Path to the output GDS file.
})
exec_time
```

```
##    user  system elapsed
##  15.312   0.189  15.514
```

Once we converted the VCF to the GDS, we can create the gbsrGenotypeData instance for our data.

```
gdata <- loadGDS("./data/gbs_nbolf2.gds")
```

If your samples have non autosomal chromosomes such as X and Y chromosomes or mitochondrial one, please pass the named list to define which chromosome is which type of non autosomal chromosome. * This argument can be specified but no effect in the current implementation. This will work in a future release.

```
# Not run.
gdata <- loadGDS("./data/gbs_nbolf2.gds",
                 non_autosomes =  list(X = 13,
                                       Y = 14,
                                       M = 15)) # M indicates mitochondrial chromosome.
```

Some getter functions allow you to retrieve basic information of genotype data, e.g. number of SNPs and samples, chromosome names, physical position of SNPs and alleles.

```
nScan(gdata) # Number of samples
```

```
## [1] 816
```

```
nSnp(gdata) # Number of SNPs
```

```
## [1] 20224
```

```
head(getChromosome(gdata)) # Indices of chromosome ID of all markers
```

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

```
head(getChromosome(gdata, name = TRUE)) # Chromosome names of all markers
```

```
## [1] 1 1 1 1 1 1
## Levels: 1 2 3 4 5 6 7 8 9 10 11 12
```

```
getChromosome(gdata, levels = TRUE) # Unique set of chromosome names
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

```
head(getPosition(gdata)) # Position (bp) of all markers
```

```
## [1] 19357 19395 38474 38477 38508 38510
```

```
head(getAlleleA(gdata)) # Reference allele of all markers
```

```
## [1] "C" "G" "G" "T" "T" "T"
```

```
head(getAlleleB(gdata)) # Alternative allele of all markers
```

```
## [1] "A" "C" "A" "C" "C" "C"
```

```
head(getSnpID(gdata)) # SNP IDs
```

```
## [1] 1 2 3 4 5 6
```

```
nScan(gdata) # Number of samples
```

```
## [1] 816
```

```
head(getScanID(gdata)) # sample IDs
```

```
## [1] "F2_1900" "F2_1901" "F2_1902" "F2_1903" "F2_1904" "F2_1905"
```

`getGenotype` is a function in `GWASTools` but works for `gbsrGenotypeData` too.

```
g <- getGenotype(gdata) # Genotype calls in which 0, 1, and 2 indicate the number of reference allele.
```

Calculate summary statitics

`countGenotype` and `countRead` are class methods of `gbsrGenotypeData` and they summarize genotype counts and read counts both per SNP and per sample.

```
gdata <- countGenotype(gdata)
gdata <- countRead(gdata)
```

The returned values from the methods are stored in `snpAnnot` and `scanAnnot` slots. We cannot extract the data with directly specifing the slots but via the `pData` method.

```
gdata@snpAnnot
```

```
## An object of class 'SnpAnnotationDataFrame'
##   snps: 1 2 ... 20224 (20224 total)
##   varLabels: snpID chromosome ... countReadAlt (17 total)
##   varMetadata: labelDescription
```

```
gdata@scanAnnot
```

```
## An object of class 'ScanAnnotationDataFrame'
##   scans: 1 2 ... 816 (816 total)
##   varLabels: scanID validScan ... countReadAlt (11 total)
##   varMetadata: labelDescription
```

```
head(pData(gdata@snpAnnot), n = 3)
```

```
##   snpID chromosome chromosome.name position alleleA alleleB validMarker ploidy
## 1     1          1               1    19357       C       A        TRUE      2
## 2     2          1               1    19395       G       C        TRUE      2
## 3     3          1               1    38474       G       A        TRUE      2
##   countGenoRef countGenoHet countGenoAlt countGenoMissing countAlleleRef
## 1           30            0           18              768             60
## 2           30            0           18              768             60
## 3            0            1          813                2              1
##   countAlleleAlt countAlleleMissing countReadRef countReadAlt
## 1             36               1536           33           28
## 2             36               1536           33           28
## 3           1627                  4            9         8342
```

```
head(pData(gdata@scanAnnot), n = 3)
```

```
##    scanID validScan countGenoRef countGenoHet countGenoAlt countGenoMissing
## 1 F2_1900      TRUE         7290         2958         2741             7235
## 2 F2_1901      TRUE         8161         2217         2098             7748
## 3 F2_1902      TRUE         8259         2270         1971             7724
##   countAlleleRef countAlleleAlt countAlleleMissing countReadRef countReadAlt
## 1          17538           8440              14470        77160        40039
## 2          18539           6413              15496        89518        42546
## 3          18788           6212              15448        78809        36072
```

These summary statistics can be visualized via ploting functions. With the values obtained via `countGenotype`, we can plot histgrams of missing rate (Figure 1), heterozygosity (Figure 2), reference allele frequency (Figure 3) as shown below.

```
hist(gdata, stats = "missing") # Histgrams of missing rate
```

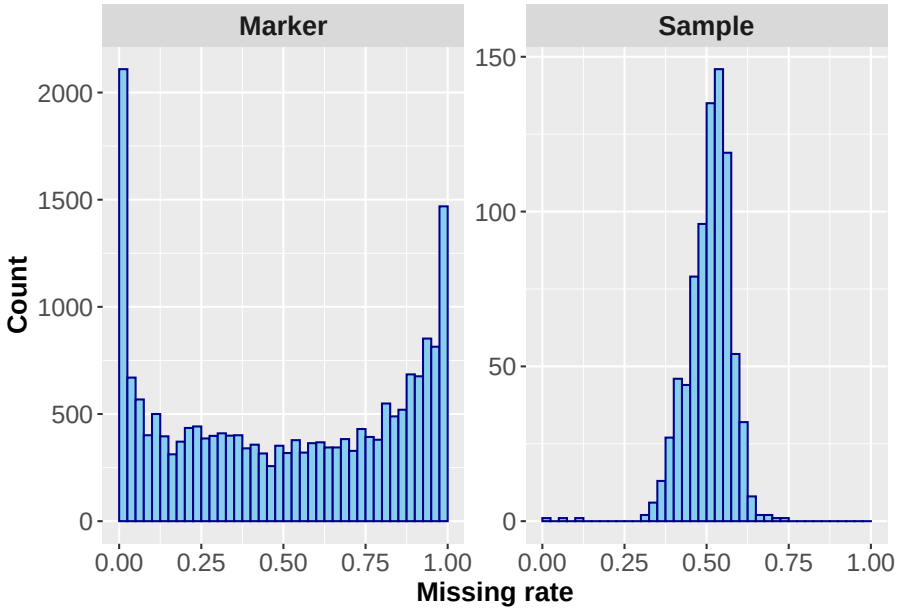


Figure 1: Missing rate per marker and per sample.

```
hist(gdata, stats = "het") # Histgrams of heterozygosity
```

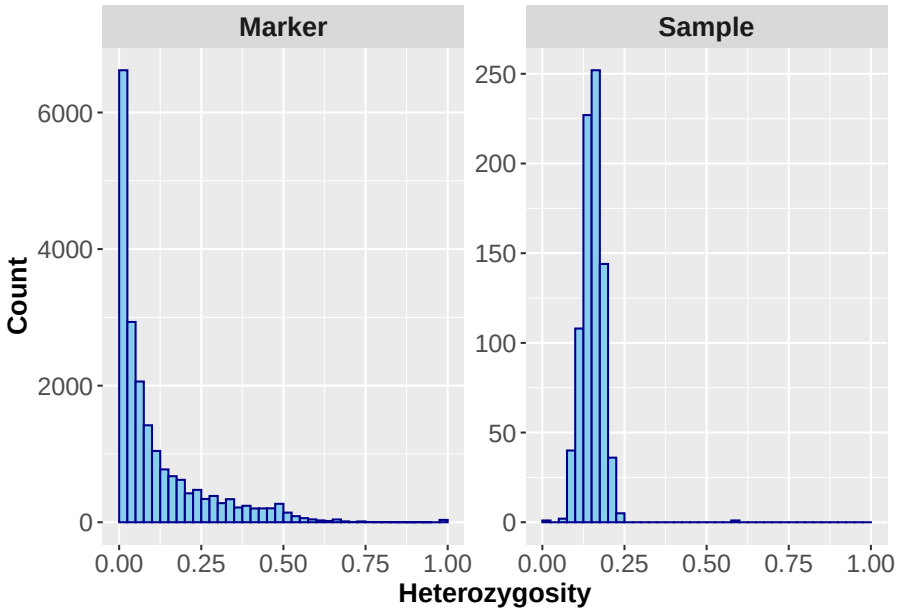


Figure 2: Heterozygosity per marker and per sample.

```
hist(gdata, stats = "raf") # Histgrams of reference allele frequency
```

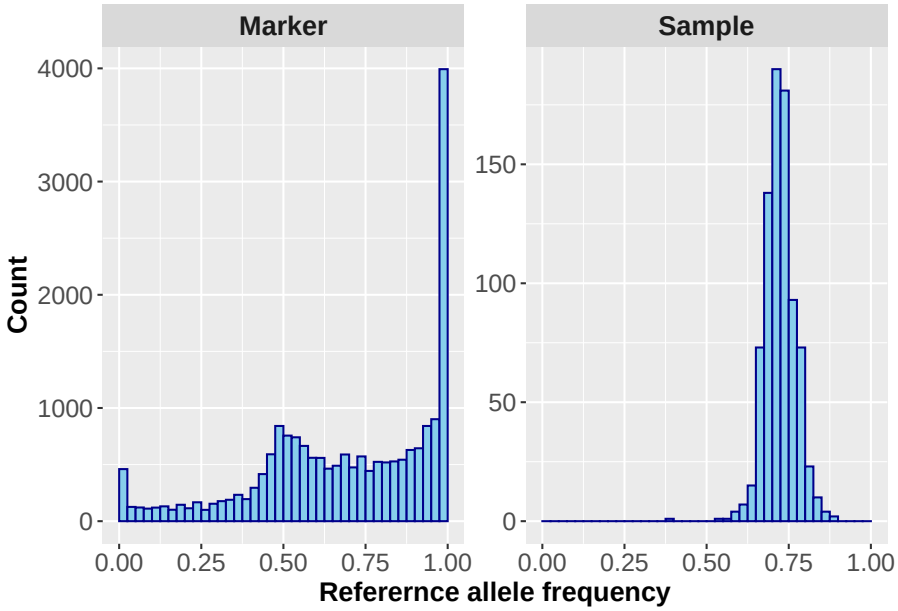


Figure 3: Reference allele frequency per marker and per sample.

With the values obtained via `countRead`, we can plot histgrams of total read depth (Figure 4), allelic read depth (Figure 5), reference read frequency (Figure 6) as shown below.

```
hist(gdata, stats = "dp") # Histgrams of total read depth
```

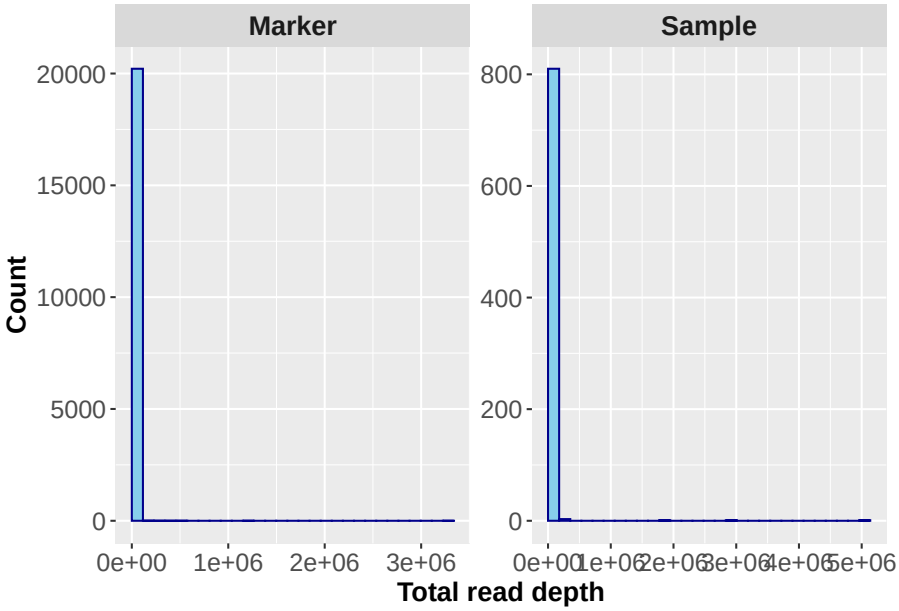


Figure 4: Total read depth per marker and per sample.

```
hist(gdata, stats = "ad_ref") # Histgrams of allelic read depth
```

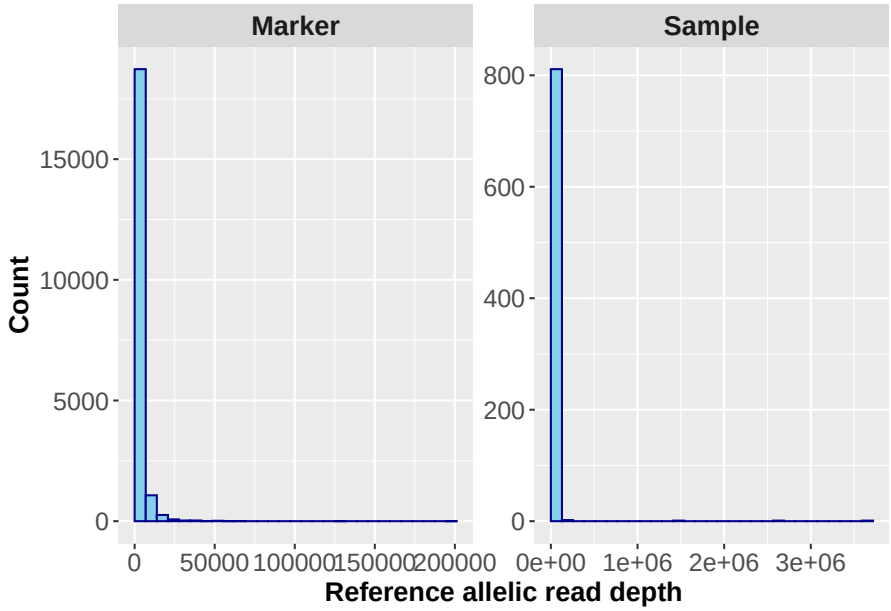


Figure 5: Reference read depth per marker and per sample.

```
hist(gdata, stats = "ad_ref") # Histgrams of allelic read depth
```

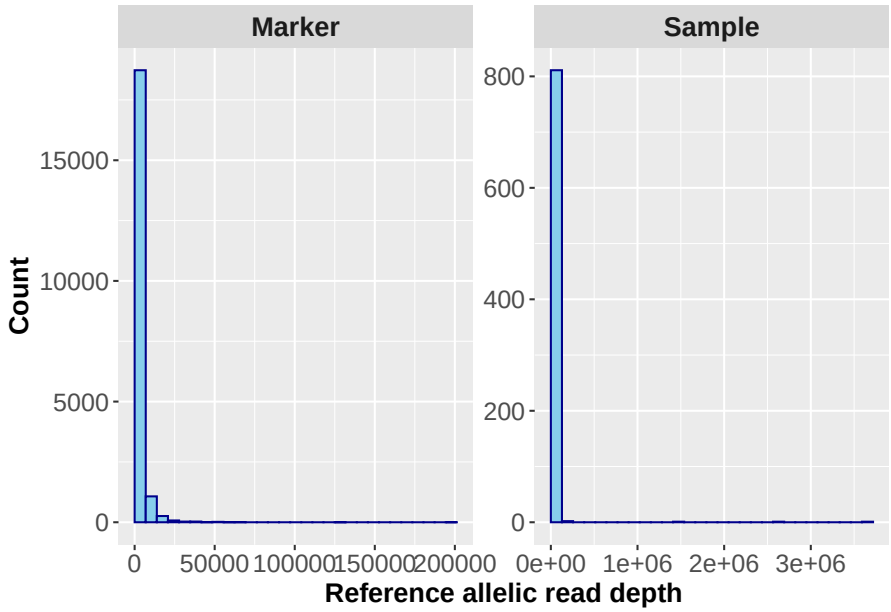


Figure 6: Alternative read depth per marker and per sample.

```
hist(gdata, stats = "rrf") # Histgrams of reference allele frequency
```

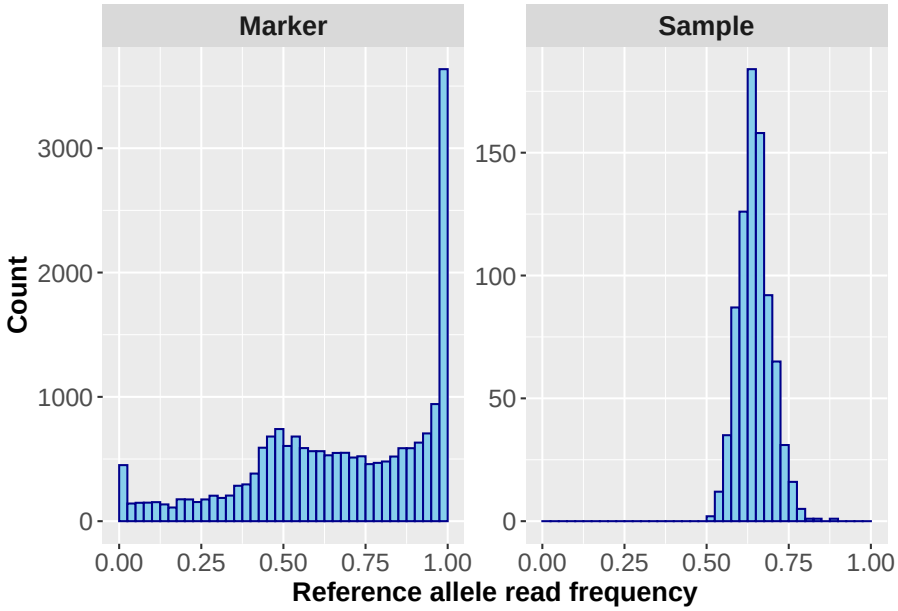


Figure 7: Reference read per marker and per sample.

In addition to `countGenotype` and `countRead`, we can get mean, sd, and quantile of read counts per marker and per sample. Unlike `countRead`, this function first normalize read counts by dividing each read count of both alleles at a marker in a sample by the total read count of the sample followed by multiplying it by 10^6 to be read counts per million. This normalization allow us to compare read data distributions obtained for the samples without concern for absolute differences in total read counts between samples. This calculation takes a longer time than those by `countGenotype` and `countRead`.

```
gdata <- calcReadStats(gdata, q = 0.5)
```

The values specified for the "q" argument are passed to the "quantile" function internally to get quantiles. The "q" argument accepts a numeric vector and has `NULL` as default which let the function return no quantile.

To plot those statistics, we can also use `hist`.

```
hist(gdata, stats = "mean_ref") # Histgrams of mean allelic read depth
```

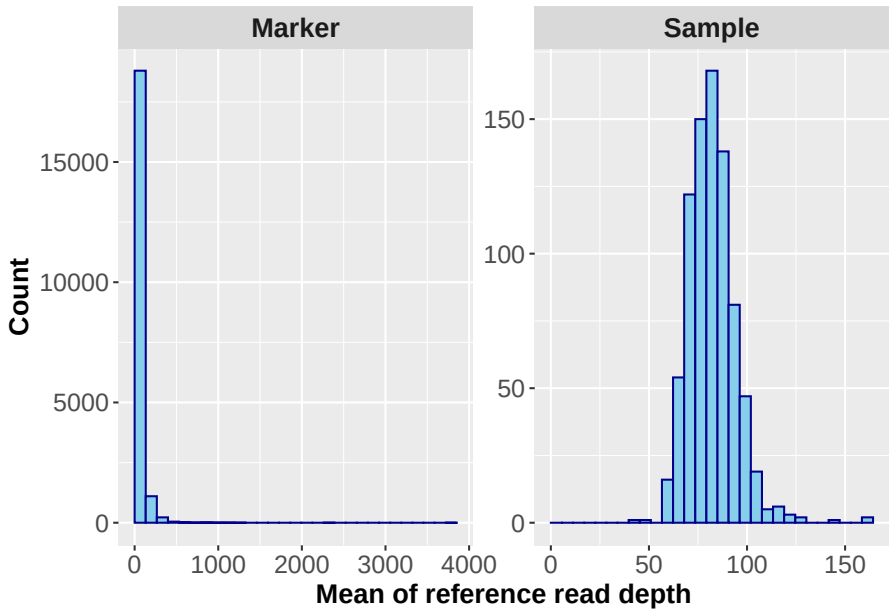


Figure 8: Mean of reference read depth per marker and per sample.

```
hist(gdata, stats = "mean_ref") # Histgrams of mean allelic read depth
```

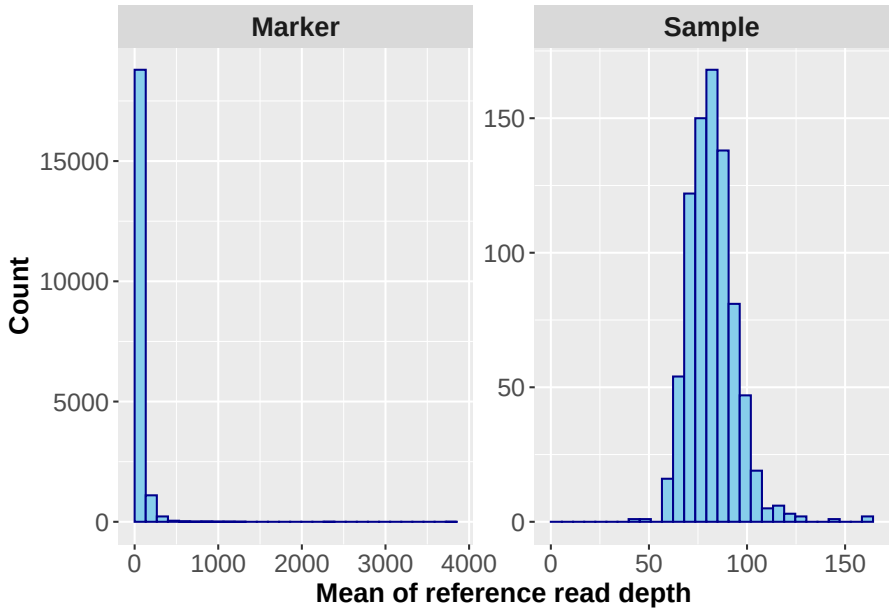


Figure 9: Mean of alternative read depth per marker and per sample.

```
hist(gdata, stats = "sd_ref") # Histgrams of standard deviation of read depth
```

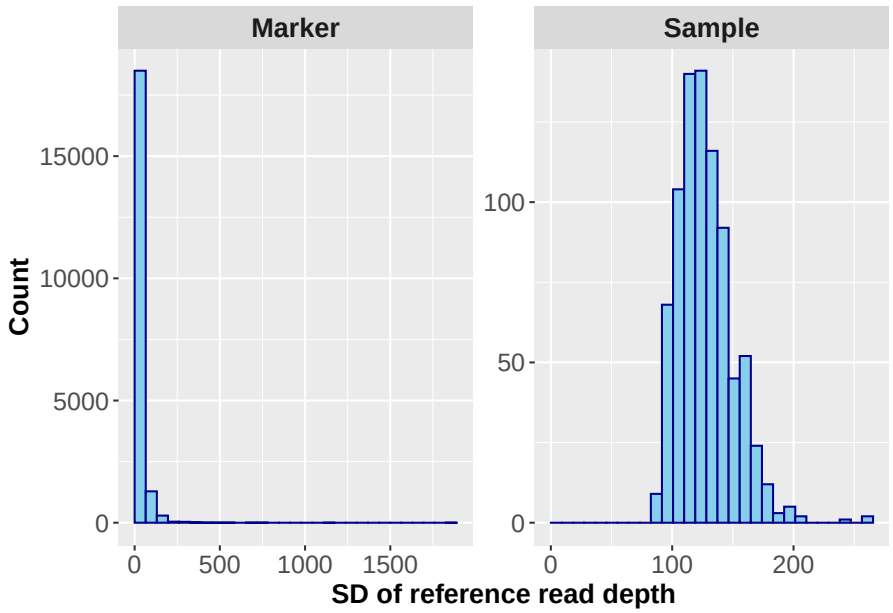


Figure 10: SD of reference read depth per marker and per sample.

```
hist(gdata, stats = "sd_ref") # Histgrams of standard deviation of read depth
```

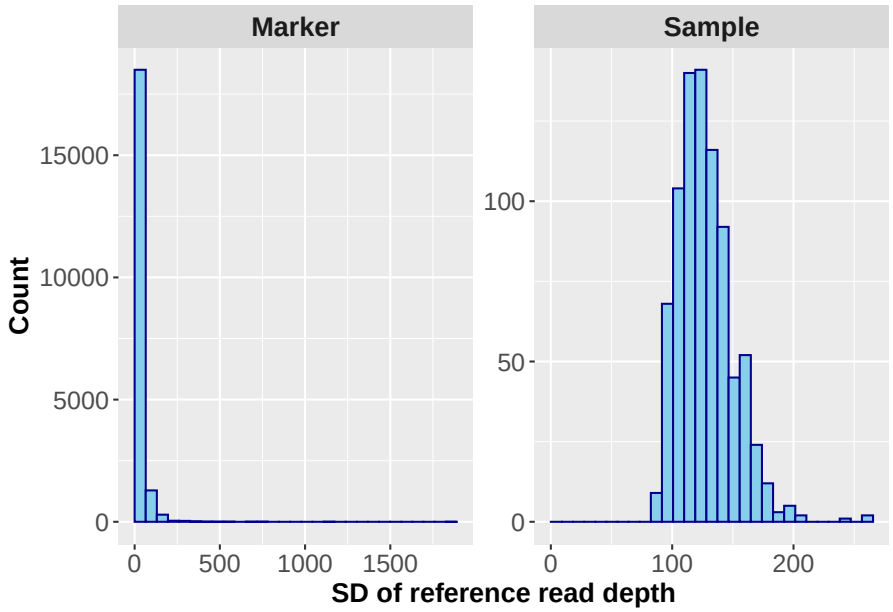


Figure 11: SD of alternative read depth per marker and per sample.

```
hist(gdata, stats = "qtile_ref", q = 0.5) # Histgrams of quantile of read depth
```

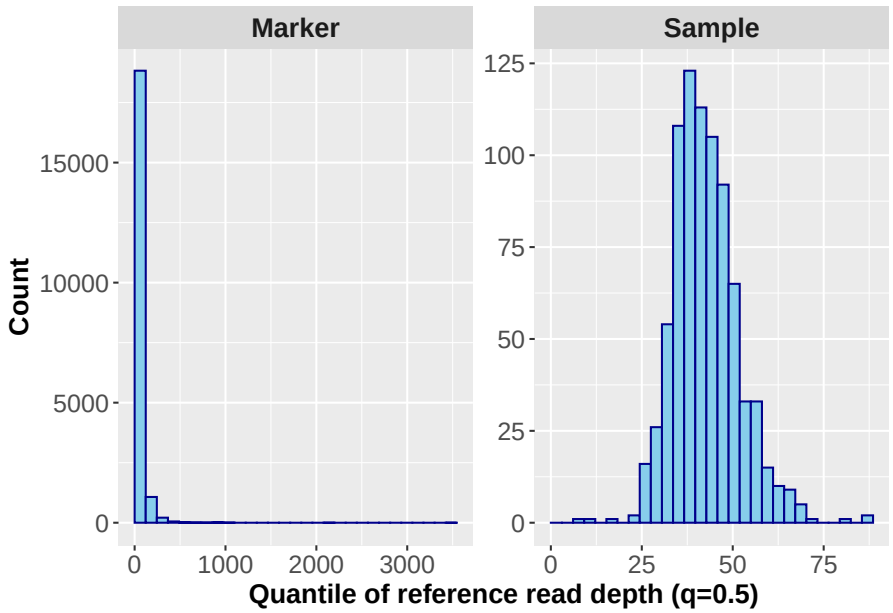


Figure 12: Quantile of reference read depth per marker and per sample.

```
hist(gdata, stats = "qtile_ref", q = 0.5) # Histgrams of quantile of read depth
```

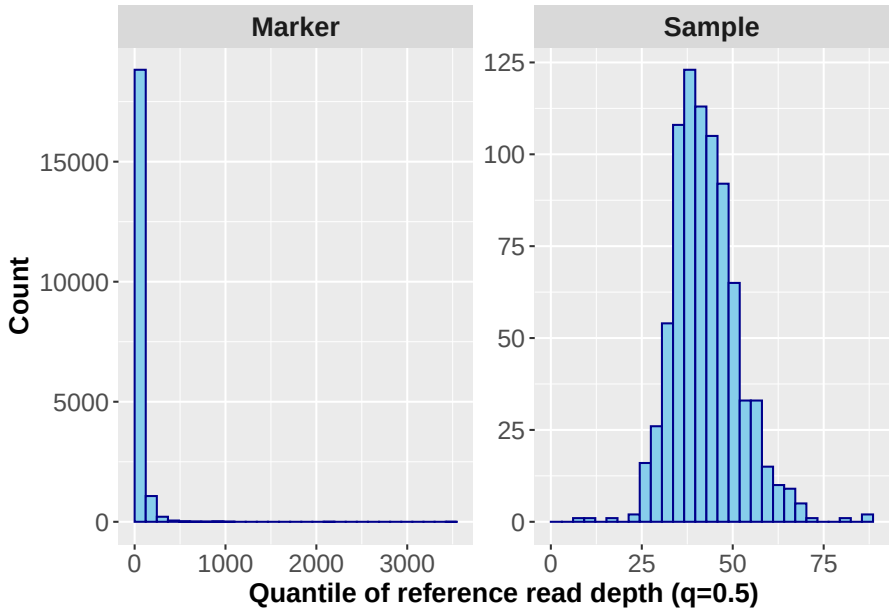
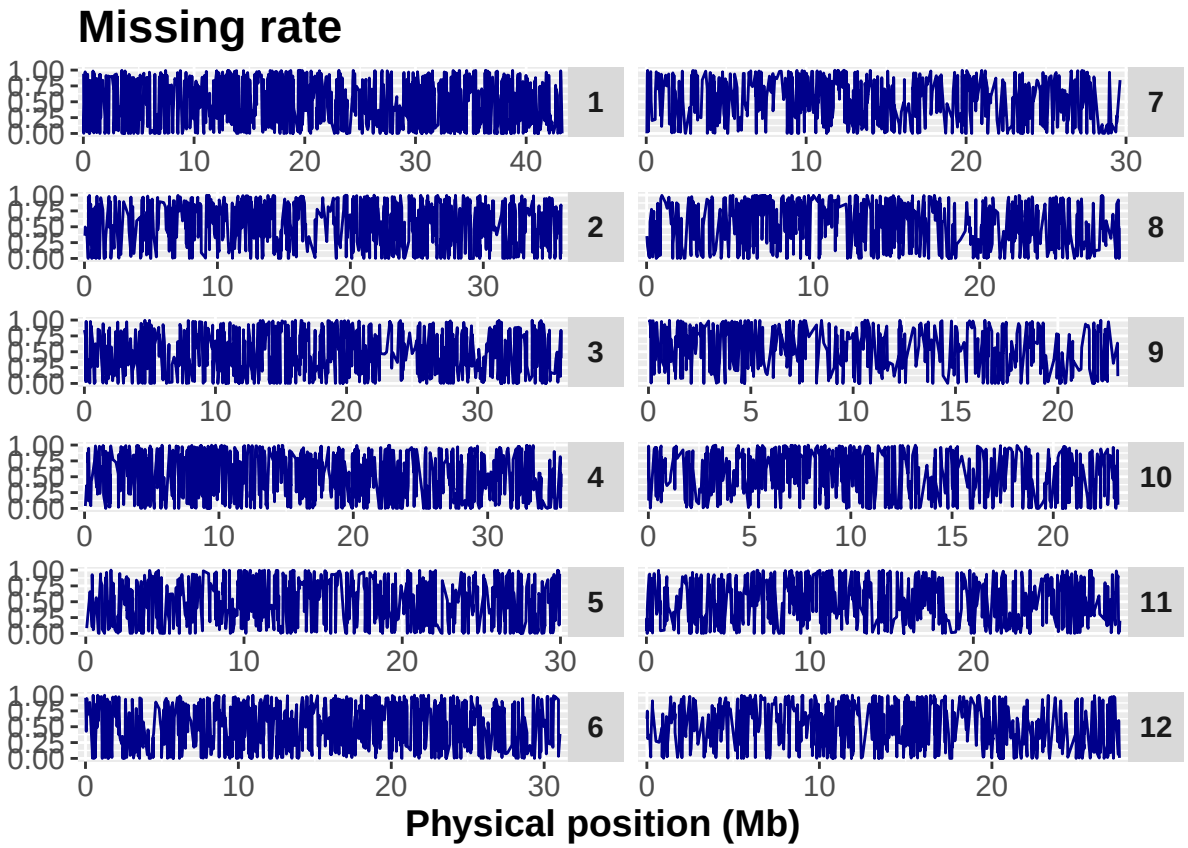


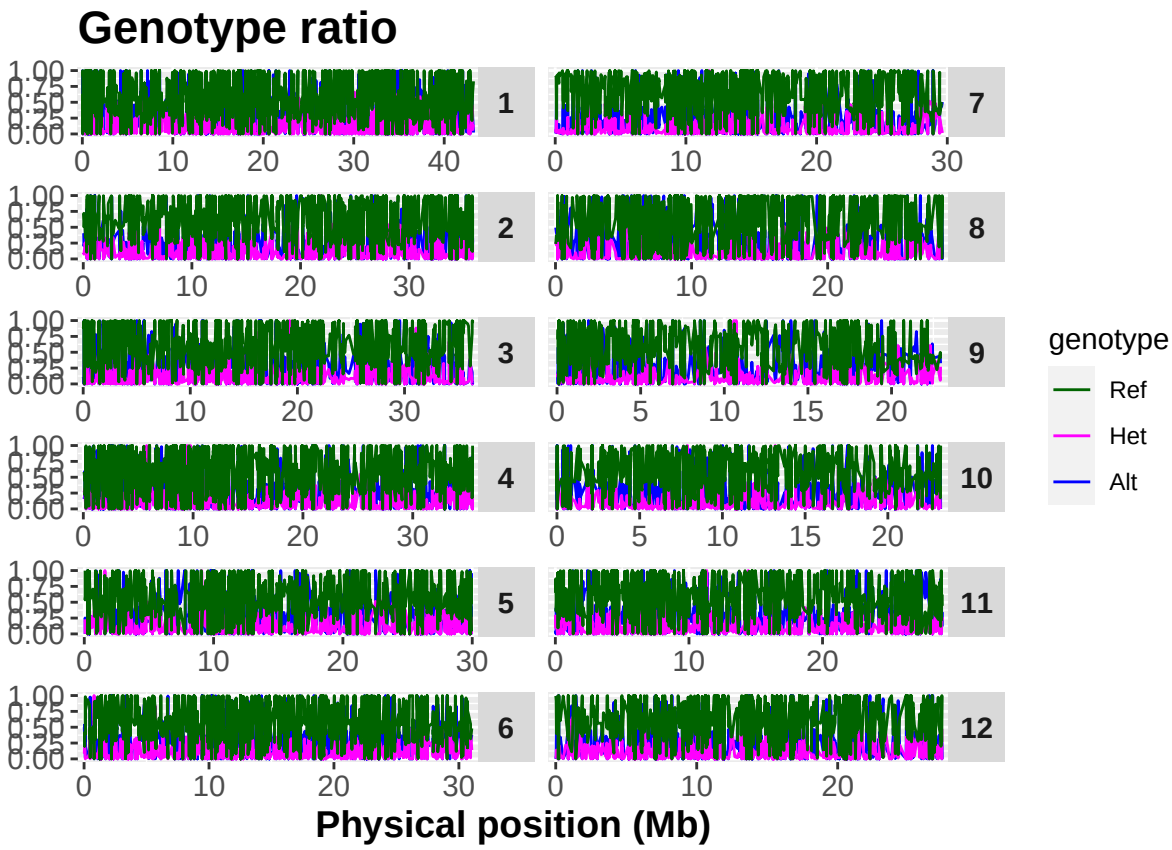
Figure 13: Quantile of alternative read depth per marker and per sample.

`plot()` and `pairs()` provide other ways to visualize statistics. `plot()` draws a line plot of a specified statistics per marker along each chromosome. `pairs()` give us a two-dimensional scatter plot to visualize relationship between statistics.

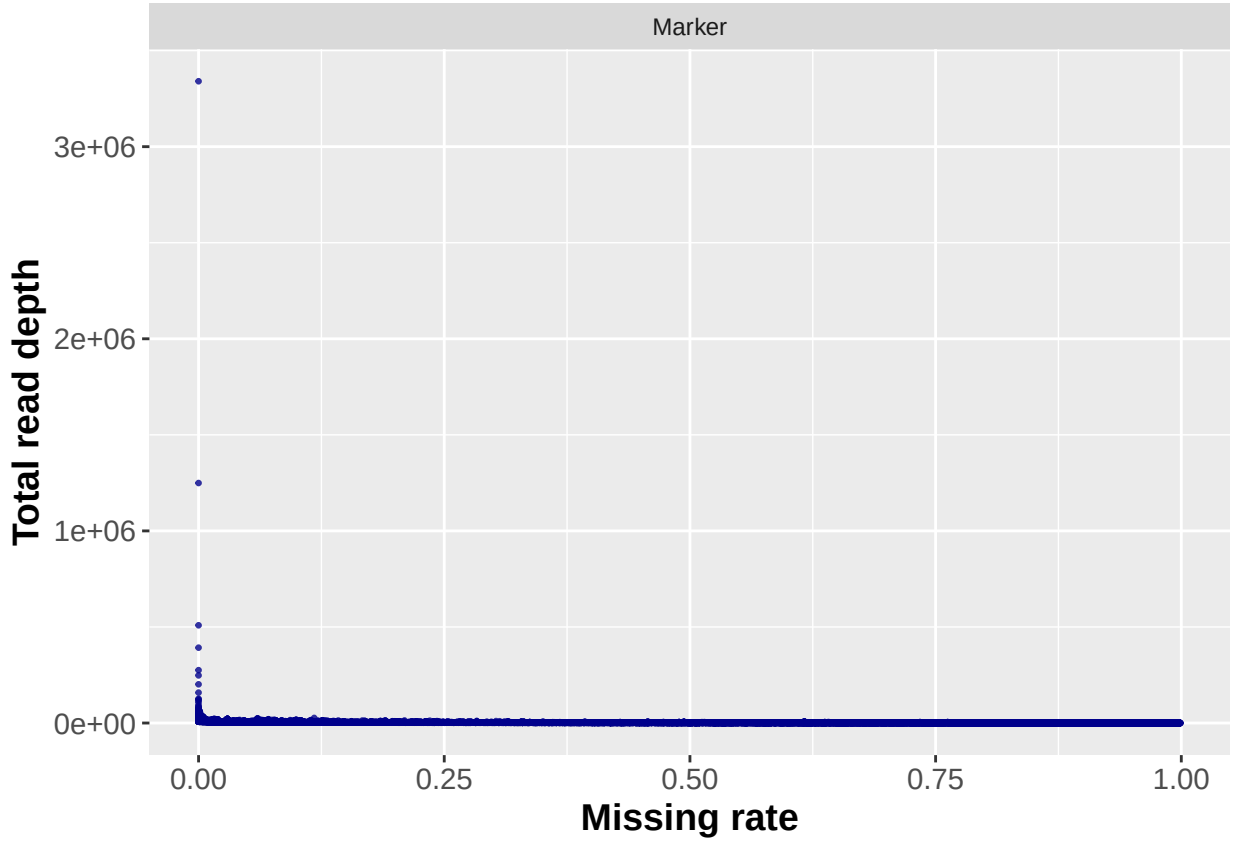
```
plot(gdata, stats = "missing", coord = c(6, 2)) # coord controls the number of rows and columns of face
```




```
plot(gdata, stats = "geno", coord = c(6, 2)) # coord controls the number of rows and columns of facets.
```




```
pairs(gdata, stats1 = "missing", stats2 = "dp")
```



The statistics obtained via `countGenotype`, `countReat`, and `calcReadStats` are sotred in the `snpAnnot` and `scanAnnot` slots. They can be retrieved using getter functions as follows.

```
head(getCountGenoRef(gdata, target = "snp")) # Reference genotype count per marker
```

```
## [1]  30  30   0 812 813 813
```

```
head(getCountGenoRef(gdata, target = "scan")) # Reference genotype count per sample
```

```
## [1] 7290 8161 8259 8185 7802 7864
```

```
head(getCountGenoHet(gdata, target = "snp")) # Heterozygote count per marker
```

```
## [1] 0 0 1 2 1 1
```

```
head(getCountGenoHet(gdata, target = "scan")) # Heterozygote count per sample
```

```
## [1] 2958 2217 2270 2776 2272 2661
```

```
head(getCountGenoAlt(gdata, target = "snp")) # Alternative genotype count per marker
```

```
## [1]  18  18 813   0   0   0
```

```
head(getCountGenoAlt(gdata, target = "scan")) # Alternative genotype count per sample
```

```
## [1] 2741 2098 1971 2321 2399 2854
```

```
head(getCountGenoMissing(gdata, target = "snp")) # Missing count per marker
```

```
## [1] 768 768   2   2   2   2
```

```
head(getCountGenoMissing(gdata, target = "scan")) # Missing count per sample
```

```
## [1] 7235 7748 7724 6942 7751 6845
```

```
head(getCountAlleleRef(gdata, target = "snp")) # Reference allele count per marker
```

```
## [1]   60   60    1 1626 1627 1627
```

```
head(getCountAlleleRef(gdata, target = "scan")) # Reference allele count per sample
```

```
## [1] 17538 18539 18788 19146 17876 18389
```

```
head(getCountAlleleAlt(gdata, target = "snp")) # Alternative allele count per marker
```

```
## [1]   36   36 1627    2    1    1
```

```
head(getCountAlleleAlt(gdata, target = "scan")) # Alternative allele count per sample
```

```
## [1] 8440 6413 6212 7418 7070 8369
```

```
head(getCountAlleleMissing(gdata, target = "snp")) # Missing allele count per marker
```

```
## [1] 1536 1536    4    4    4    4
```

```
head(getCountAlleleMissing(gdata, target = "scan")) # Missing allele count per sample
```

```
## [1] 14470 15496 15448 13884 15502 13690
```

```
head(getCountReadRef(gdata, target = "snp")) # Reference read count per marker
```

```
## [1]   33   33    9 8332 8329 8332
```

```
head(getCountReadRef(gdata, target = "scan")) # Reference read count per sample
```

```
## [1] 77160 89518 78809 96211 76591 88000
```

```
head(getCountReadAlt(gdata, target = "snp")) # Alternative read count per marker
```

```
## [1]   28   28 8342   15   16   13
```

```
head(getCountReadAlt(gdata, target = "scan")) # Alternative read count per sample
```

```
## [1] 40039 42546 36072 45967 38218 46892
```

```
head(getCountRead(gdata, target = "snp")) # Sum of reference and alternative read counts per marker
```

```
## [1]   61   61 8351 8347 8345 8345
```

```
head(getCountRead(gdata, target = "scan")) # Sum of reference and alternative read counts per sample
```

```
## [1] 117199 132064 114881 142178 114809 134892
```

```
head(getMeanReadRef(gdata, target = "snp")) # Mean of reference allele read count per marker
```

```
## [1]  17.172321  17.172321   8.578077 138.728032 138.699253 138.752117
```

```
head(getMeanReadRef(gdata, target = "scan")) # Mean of reference allele read count per sample
```

```
## [1] 63.82621 64.76572 64.77250 61.26700 65.53853 61.44615
```

```
head(getMeanReadAlt(gdata, target = "snp")) # Mean of Alternative allele read count per marker
```

```
## [1]  13.87532  13.87532 138.82800  12.63413  12.43686  13.25711
```

```
head(getMeanReadAlt(gdata, target = "scan")) # Mean of Alternative allele read count per sample
```

```
## [1] 56.01453 68.98544 69.11611 59.11611 67.26274 59.00972
```

```
head(getSDReadRef(gdata, target = "snp")) # SD of reference allele read count per marker
```

```
## [1]  7.263394  7.263394  8.358193 52.130242 52.079679 52.134087
```

```
head(getSDReadRef(gdata, target = "scan")) # SD of reference allele read count per sample
```

```
## [1] 106.37888 111.16390 103.71444 102.55137 104.85645  92.59848
```

```
head(getSDReadAlt(gdata, target = "snp")) # SD of Alternative allele read count per marker
```

```
## [1]  6.423890  6.423890 52.259496  7.221757  7.592914  8.144763
```

```
head(getSDReadAlt(gdata, target = "scan")) # SD of Alternative allele read count per sample
```

```
## [1] 466.2758 873.0088 792.3877 749.9504 528.2615 719.8666
```

```
head(getQtileReadRef(gdata, target = "snp", q = 0.5)) # Quantile of reference allele read count per mar
```

```
## [1]  16.82957  16.82957  11.24847 137.55107 137.76099 137.55107
```

```
head(getQtileReadRef(gdata, target = "scan", q = 0.5)) # Quantile of reference allele read count per sa
```

```
## [1] 34.12999 30.28835 34.81864 28.13375 34.84047 29.65335
```

```
head(getQtileReadAlt(gdata, target = "snp", q = 0.5)) # Quantile of Alternative allele read count per m
```

```
## [1]  13.32064  13.32064 137.55107  12.52646  14.47869  12.70470
```

```
head(getQtileReadAlt(gdata, target = "scan", q = 0.5)) # Quantile of Alternative allele read count per
```

```
## [1] 25.59749 22.71626 26.11398 21.10031 26.13036 22.24002
```

```
head(getMAF(gdata, target = "snp")) # Minor allele frequency per marker
```

```
## [1] 0.3750000000 0.3750000000 0.0006142506 0.0012285012 0.0006142506
## [6] 0.0006142506
```

```
head(getMAF(gdata, target = "scan")) # Minor allele frequency per sample
```

```
## [1] 0.3248903 0.2570135 0.2484800 0.2792501 0.2834122 0.3127663
```

```
head(getMAC(gdata, target = "snp")) # Minor allele count per marker
```

```
## [1] 36 36  1  2  1  1
```

```
head(getMAC(gdata, target = "scan")) # Minor allele count per sample
```

```
## [1] 8440 6413 6212 7418 7070 8369
```

You can get the proportion of each genotype call with `prop = TRUE`.

```
head(getCountGenoRef(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.6250000 0.6250000 0.0000000 0.9975430 0.9987715 0.9987715
```

```
head(getCountGenoHet(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.000000000 0.000000000 0.001228501 0.002457002 0.001228501 0.001228501
```

```
head(getCountGenoAlt(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.3750000 0.3750000 0.9987715 0.0000000 0.0000000 0.0000000
```

```
head(getCountGenoMissing(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.94117647 0.94117647 0.00245098 0.00245098 0.00245098 0.00245098
```

The proportion of each allele counts.

```
head(getCountAlleleRef(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.6250000000 0.6250000000 0.0006142506 0.9987714988 0.9993857494
## [6] 0.9993857494
```

```
head(getCountAlleleAlt(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.3750000000 0.3750000000 0.9993857494 0.0012285012 0.0006142506
## [6] 0.0006142506
```

```
head(getCountAlleleMissing(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.94117647 0.94117647 0.00245098 0.00245098 0.00245098 0.00245098
```

The proportion of each allele read counts.

```
head(getCountReadRef(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.540983607 0.540983607 0.001077715 0.998202947 0.998082684 0.998442181
```

```
head(getCountReadAlt(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.459016393 0.459016393 0.998922285 0.001797053 0.001917316 0.001557819
```

Filtering and subsetting data

Based on the statistics we obtained, we can filter out less reliable markers and samples using `setSnpFilter` and `setScanFilter`.

```
# Not run
gdata <- setSnpFilter(
  id,   # Specify a character vector of snpID to be removed.
  missing = 1,   # Specify an upper limit of missing rate.
  het = c(0, 1),   # Specify a lower and an upper limit of heterozygosity rate.
  mac = 0,   # Specify a lower limit of minor allele count.
  maf = 0.05,   # Specify a lower limit of minor allele frequency.
  ad_ref = c(0, Inf),   # Specify a lower and an upper limit of reference allele count.
  ad_alt = c(0, Inf),   # Specify a lower and an upper limit of alternative allele count.
  dp = c(0, Inf),   # Specify a lower and an upper limit of total read count.
  mean_ref = c(0, Inf),   # Specify a lower and an upper limit of mean reference allele count.
  mean_alt = c(0, Inf),   # Specify a lower and an upper limit of mean alternative allele count.
  sd_ref = Inf,   # Specify a lower and an upper limit of SD of reference allele count.
  sd_alt = Inf    # Specify a lower and an upper limit of SD of alternative allele count.
)

gdata <- setScanFilter(
  id,   # Specify a character vector of snpID to be removed.
  missing = 1,   # Specify an upper limit of missing rate.
  het = c(0, 1),   # Specify a lower and an upper limit of heterozygosity rate.
  mac = 0,   # Specify a lower limit of minor allele count.
  maf = 0,   # Specify a lower limit of minor allele frequency.
  ad_ref = c(0, Inf),   # Specify a lower and an upper limit of reference allele count.
  ad_alt = c(0, Inf),   # Specify a lower and an upper limit of alternative allele count.
  dp = c(0, Inf),   # Specify a lower and an upper limit of total read count.
  mean_ref = c(0, Inf),   # Specify a lower and an upper limit of mean reference allele count.
  mean_alt = c(0, Inf),   # Specify a lower and an upper limit of mean alternative allele count.
  sd_ref = Inf,   # Specify a lower and an upper limit of SD of reference allele count.
  sd_alt = Inf   # Specify a lower and an upper limit of SD of alternative allele count.
)
```

`setCallFilter()` is another type of filtering which works on each genotype call. We can replace some genotype calls with missing. If you would like to filter out less reliable genotype calls supported by less than 5 reads, set the arguments as below.

```
gdata <- setCallFilter(gdata, dp_count = c(5, Inf))
```

If need to remove genotype calls supported by too many reads, which might be the results of mismapping from repetitive sequences, set as follows.

```
gdata <- setCallFilter(gdata, norm_dp_count = c(0, 1000))
gdata <- setCallFilter(gdata, norm_ref_count = c(0, 1000),
                       norm_alt_count = c(0, 800))
```

Usually reference reads and alternative reads show different data distributions. Thus, we can set the different thresholds for them via `norm_ref_count` and `norm_alt_count`. `setCallFilter()` also has arguments `scan_ref_qtile`, `scan_alt_qtile`, `snp_ref_qtile`, and `snp_alt_qtile` to filter out genotype calls based on quantiles of read counts per marker and per sample.

Here, let's filter out calls supported by less than 5 reads and then filter out markers having more than 10% of missing rate.

```
gdata <- setCallFilter(gdata, dp_count = c(5, Inf))
gdata <- setSnpFilter(gdata, missing = 0.1)
```

In addition to those statistics based filtering functions, `GBScleanR` provides filtering function based on relative marker positions. Markers locating too close each other usually have redundant information, especially if those markers are closer each other than the read length, in which case the markers are supported by completely (or almost) the same set of reads. To select only one marker from those markers, we can sue `thinMarker`. This function selects one marker having the least missing rate from each stretch with the specified length. If some markers have the least missing rate, select the first marker in the stretch.

```
thinMarker(gdata, range = 150) # Here we select only one marker from each 150 bp stretch.
```

```
## File: /home/ftom/hdd2/gbscleanr/data/gbs_nbolf2.gds (38.7M)
## +    [  ] *
## |--+ sample.id   { Str8 816 LZMA_ra(16.7%), 1.1K }
## |--+ snp.id   { Int32 20224 LZMA_ra(8.33%), 6.6K }
## |--+ snp.rs.id   { Str8 20224 LZMA_ra(15.2%), 37.9K }
## |--+ snp.position   { Int32 20224 LZMA_ra(43.8%), 34.6K }
## |--+ snp.allele   { Str8 20224 LZMA_ra(11.9%), 9.4K }
## |--+ genotype   { Bit2 816x20224 LZMA_ra(32.5%), 1.3M } *
## |--+ annotation   [  ]
## |  |--+ info   [  ]
## |  \--+ format   [  ]
## |     \--+ AD   [  ]
## |        |--+ data   { VL_Int 816x40448 LZMA_ra(11.5%), 3.6M }
## |        |--+ norm   { Float32 40448x816 LZMA_ra(6.13%), 7.7M }
## |        |--+ filt.scan   { Bit1 20224x816 LZMA_ra(22.3%), 449.9K }
## |        \--+ filt.data   { VL_Int 816x40448 LZMA_ra(6.63%), 2.1M }
## |--+ snp.chromosome.name   { Str8 20224 LZMA_ra(0.33%), 205B }
## |--+ snp.chromosome   { Int8 20224 LZMA_ra(0.82%), 173B }
## \--+ filt.genotype   { Bit2 816x20224 LZMA_ra(15.1%), 608.1K }
## An object of class 'SnpAnnotationDataFrame'
##   snps: 1 2 ... 20224 (20224 total)
##   varLabels: snpID chromosome ... qtileReadAlt0.5 (23 total)
##   varMetadata: labelDescription
## An object of class 'ScanAnnotationDataFrame'
##   scans: 1 2 ... 816 (816 total)
##   varLabels: scanID validScan
##   varMetadata: labelDescription
```

We can obtain the summary statistics using `countGenotype()`, `countRead()`, and `calcReadStats()` for only the SNPs and samples retained after filtering with the same codes we used before.

```
gdata <- countGenotype(gdata)
gdata <- countRead(gdata)
gdata <- calcReadStats(gdata)
```

`calcReadStats()` never calculate the normalized read counts again for the filtered data but gets mean, sd, and quantiles from the normalized values of the retained markers of samples.

We can check which markers and samples are retained after the filtering using `getValidSnp()` and `getValidScan()`.

```
head(getValidSnp(gdata))
```

```
## [1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

```
head(getValidScan(gdata))
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE
```

The class methods of `gbsrGenotypeData` basically work with only the markers and samples having `TRUE` in the returned values of `getValidSnp()` and `getValidScan()`, if you don't explicitly specify `valid = FALSE` as an argument of the class methods.

```
nSnp(gdata)
```

```
## [1] 3748
```

```
nSnp(gdata, valid = FALSE)
```

```
##  snps
## 20224
```

We can reset filtering as following.

```
gdata <- resetSnpFilters(gdata) # Reset the filter on markers
gdata <- resetScanFilters(gdata) # Reset the filter on samples
gdata <- resetCallFilters(gdata) # Reset the filter on calls
gdata <- resetFilters(gdata) # Reset all filters
```

To save the filtered data, we can create the subset GDS file containing only the retained data.

```
subset_gdata <- subsetGDS(gdata,
                          out_fn = "./data/gbs_nbolf2_subset.gds",
                          snp_incl = getValidSnp(gdata),
                          scan_incl = getValidScan(gdata))
```

`out_fn` is the file path of the output GDS file storing the subset data. Users need to specify, for `snp_incl` and `scan_incl`, a logical vector indicating which markers and samples should be included in the subset. The functions `getValidSnp()` and `getValidScan` return a logical vector indicating which markers and samples are retained by `setSnpFilter()` and `setScanFilter()`. `subsetGDS` returns a new `gbsrGenotypeData` object for the subset.

Session information

```
sessionInfo()
```

```
## R version 4.0.4 (2021-02-15)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.2 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3
##
```

```
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
## [1] GBScleanR_1.0.0     GWASTools_1.36.0    Biobase_2.50.0
## [4] BiocGenerics_0.36.0
##
## loaded via a namespace (and not attached):
##  [1] tidyr_1.1.3            bit64_4.0.5            splines_4.0.4
##  [4] stats4_4.0.4           blob_1.2.1             GenomeInfoDbData_1.2.4
##  [7] GWASExactHW_1.01       yaml_2.2.1             pillar_1.5.1
## [10] RSQLite_2.2.4          backports_1.2.1        lattice_0.20-41
## [13] quantreg_5.85          glue_1.4.2             digest_0.6.27
## [16] XVector_0.30.0         GenomicRanges_1.42.0   colorspace_2.0-0
## [19] sandwich_3.0-0         htmltools_0.5.1.1      Matrix_1.3-2
## [22] conquer_1.0.2          pkgconfig_2.0.3        broom_0.7.5
## [25] SparseM_1.81           zlibbioc_1.36.0        purrr_0.3.4
## [28] scales_1.1.1           MatrixModels_0.5-0     tibble_3.1.0
## [31] mgcv_1.8-34            farver_2.1.0           generics_0.1.0
## [34] IRanges_2.24.1         ggplot2_3.3.3          ellipsis_0.3.1
## [37] cachem_1.0.4           formula.tools_1.7.1    survival_3.2-10
## [40] magrittr_2.0.1         crayon_1.4.1           memoise_2.0.0
## [43] evaluate_0.14          mice_3.13.0            fansi_0.4.2
## [46] operator.tools_1.6.3   nlme_3.1-152           SeqArray_1.30.0
## [49] tools_4.0.4            data.table_1.14.0      lifecycle_1.0.0
## [52] matrixStats_0.58.0     stringr_1.4.0          S4Vectors_0.28.1
## [55] munsell_0.5.0          gdsfmt_1.26.1          Biostrings_2.58.0
## [58] compiler_4.0.4         GenomeInfoDb_1.26.4    logistf_1.24
## [61] rlang_0.4.10           RCurl_1.98-1.3         grid_4.0.4
## [64] labeling_0.4.2         bitops_1.0-6           rmarkdown_2.7
## [67] DNAcopy_1.64.0         gtable_0.3.0           DBI_1.1.1
## [70] R6_2.5.0               zoo_1.8-9              knitr_1.31
## [73] dplyr_1.0.5            fastmap_1.1.0          bit_4.0.4
## [76] utf8_1.2.1             stringi_1.5.3          Rcpp_1.0.6
## [79] quantsmooth_1.56.0     vctrs_0.3.6            tidyselect_1.1.0
## [82] xfun_0.22              lmtest_0.9-38
```