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.

```r
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.

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

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

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

To load the package.

```r
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("../inst/extdata/sim_pop.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] 102
```

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

```
## [1] 100
```

```
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
```

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

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

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

```
## [1] 1266164 1270080 2537850 2779885 2983182 3047595
```

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

```
## [1] "G" "G" "G" "G" "G" "G"
```

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

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

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

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

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

```
## [1] "Founder1"     "Founder2"     "G3_1_1x1_1_1" "G3_1_1x1_1_2" "G3_1_1x1_1_3"
## [6] "G3_1_1x1_1_4"
```

`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 ... 100 (100 total)
##   varLabels: snpID chromosome ... countReadAlt (17 total)
##   varMetadata: labelDescription
```

```
gdata@scanAnnot
```

```
## An object of class 'ScanAnnotationDataFrame'
##   scans: 1 2 ... 102 (102 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  1266164       G       A        TRUE      2
## 2     2          1               1  1270080       G       A        TRUE      2
## 3     3          1               1  2537850       G       A        TRUE      2
##   countGenoRef countGenoAlt countGenoMissing countGenoHet countAlleleRef
## 1           14           10               78            0             28
## 2           29           26                3           44            102
## 3           30           35                9           28             88
##   countAlleleAlt countAlleleMissing countReadRef countReadAlt
## 1             20                156           18           11
## 2             96                  6          173          173
## 3             98                 18          118          129
```

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

```
##         scanID validScan countGenoRef countGenoHet countGenoAlt
## 1     Founder1      TRUE           78            0            0
## 2     Founder2      TRUE            0            1           73
## 3 G3_1_1x1_1_1      TRUE           16           44           16
##   countGenoMissing countAlleleRef countAlleleAlt countAlleleMissing
## 1               22            156              0                 44
## 2               26              1            147                 52
## 3               24             76             76                 48
##   countReadRef countReadAlt
## 1          298            0
## 2            1          301
## 3          143          130
```

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.

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

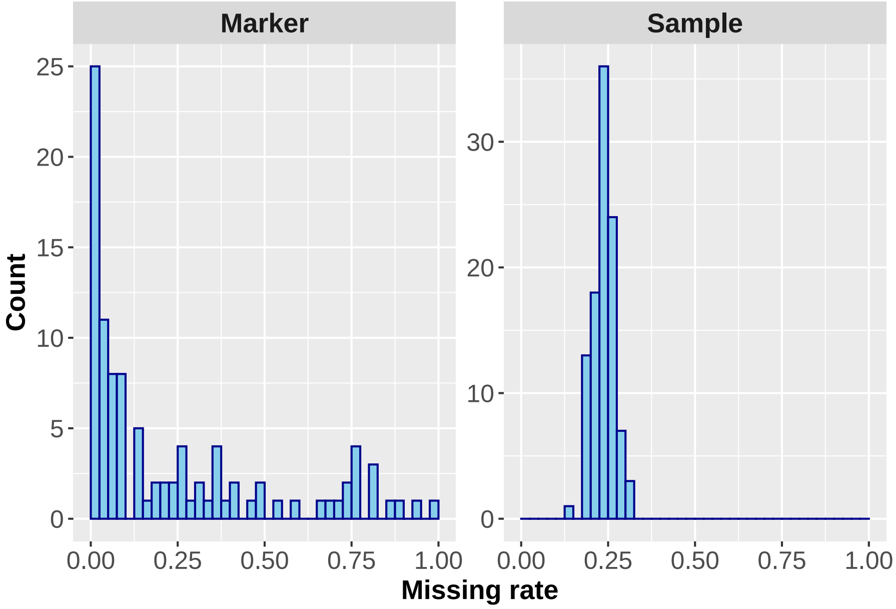


Figure 1: Missing rate per marker and per sample.

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

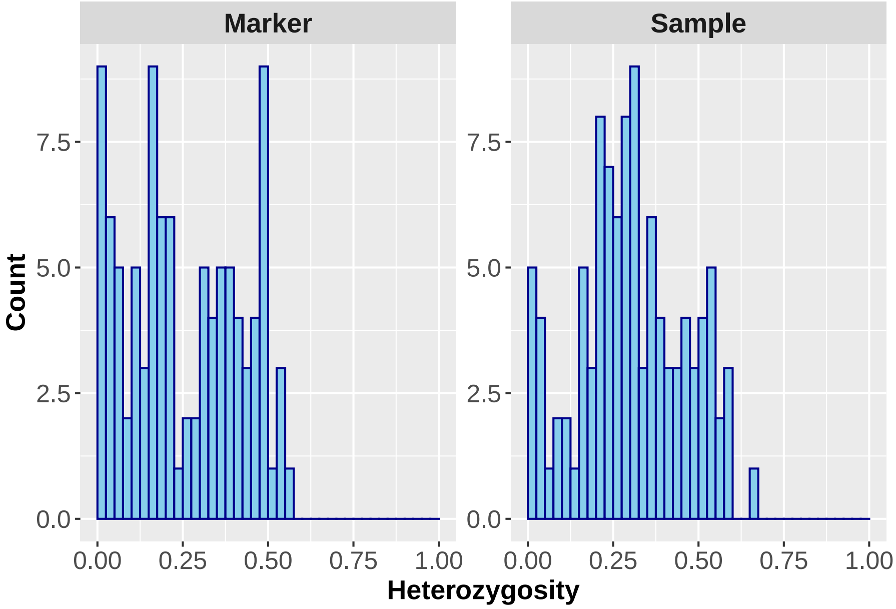


Figure 2: Heterozygosity per marker and per sample.

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

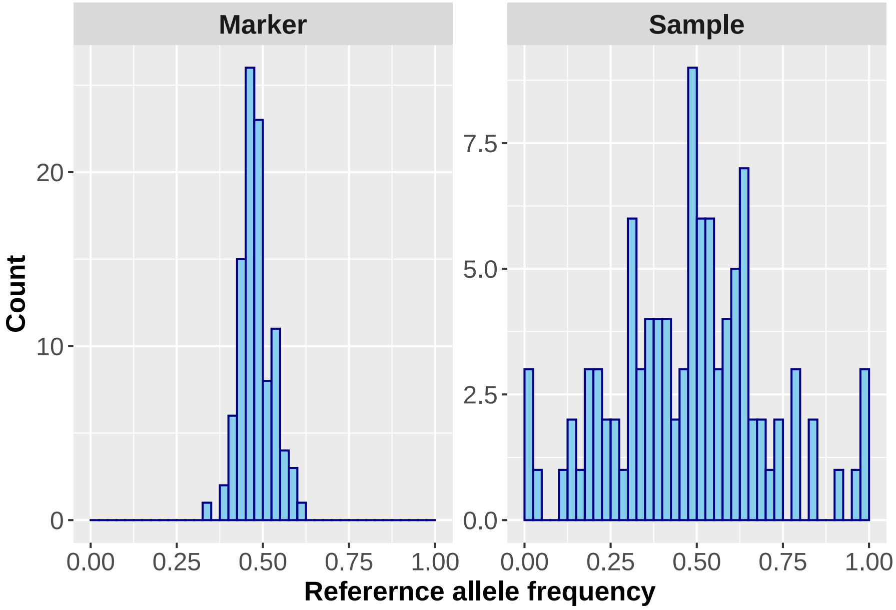


Figure 3: Reference allele frequency per marker and per sample.

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.

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

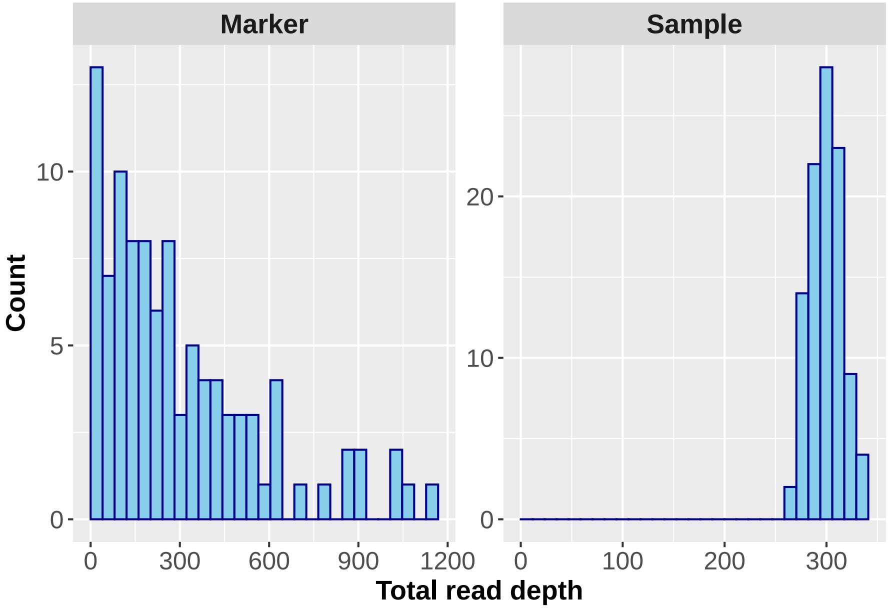


Figure 4: Total read depth per marker and per sample.

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

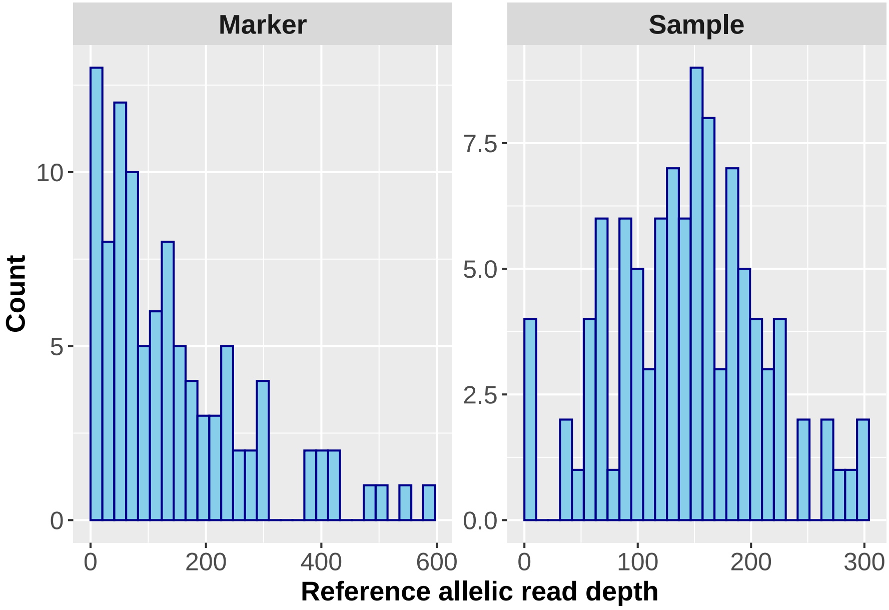


Figure 5: Reference read depth per marker and per sample.

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

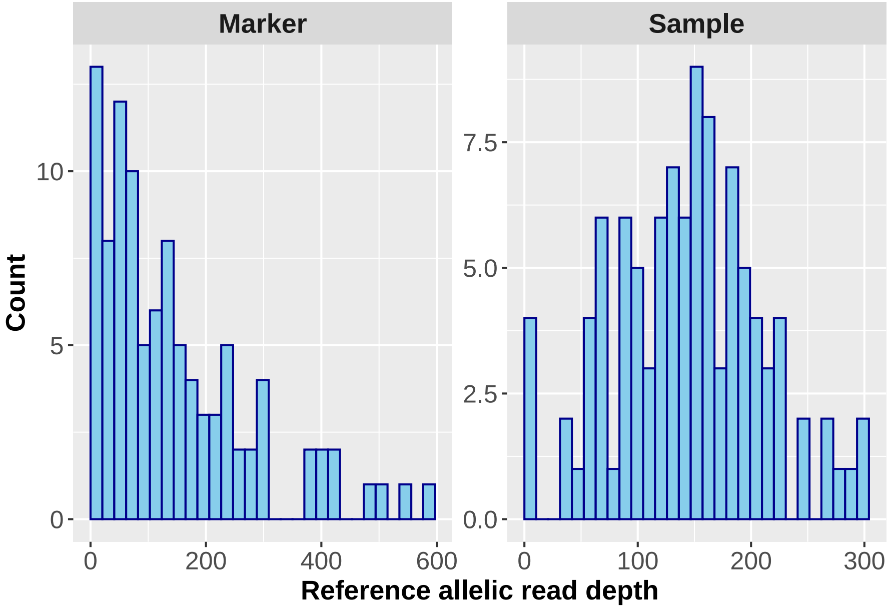


Figure 6: Alternative read depth per marker and per sample.

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

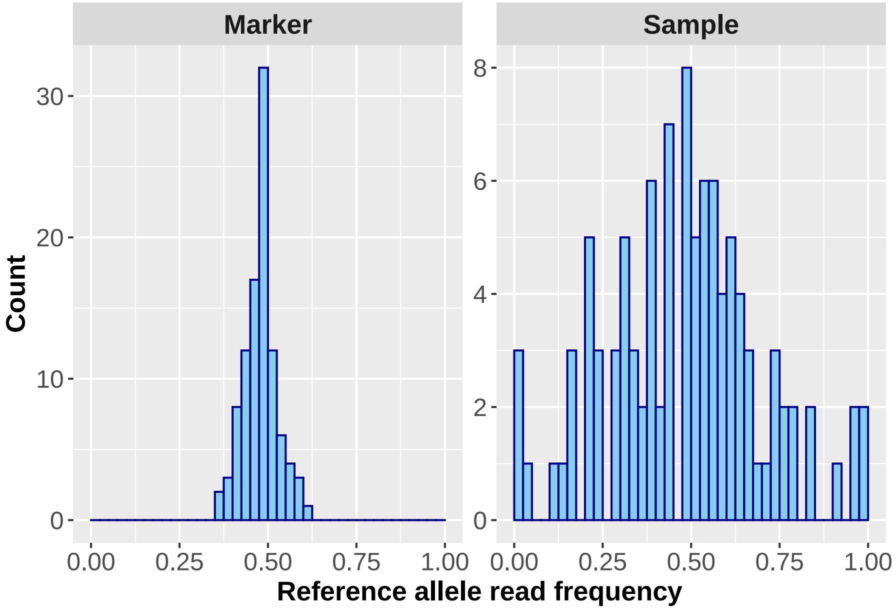


Figure 7: Reference read per marker and per sample.

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 10ˆ6 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`.

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

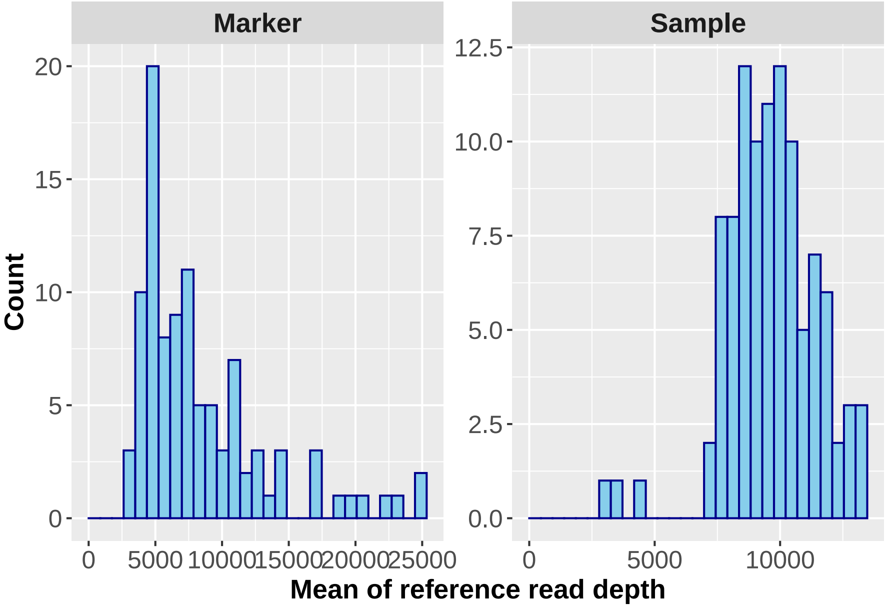


Figure 8: Mean of reference read depth per marker and per sample.

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

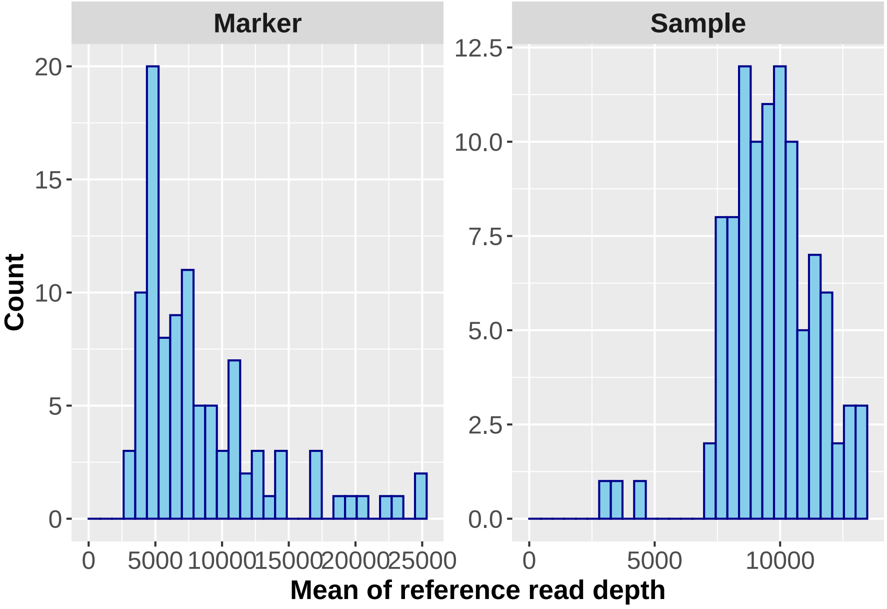


Figure 9: Mean of alternative read depth per marker and per sample.

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

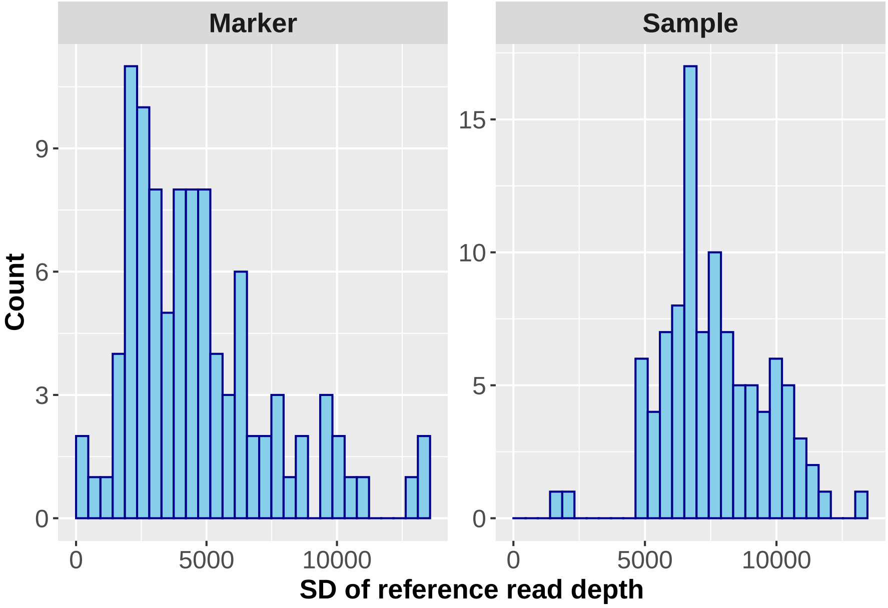


Figure 10: SD of reference read depth per marker and per sample.

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

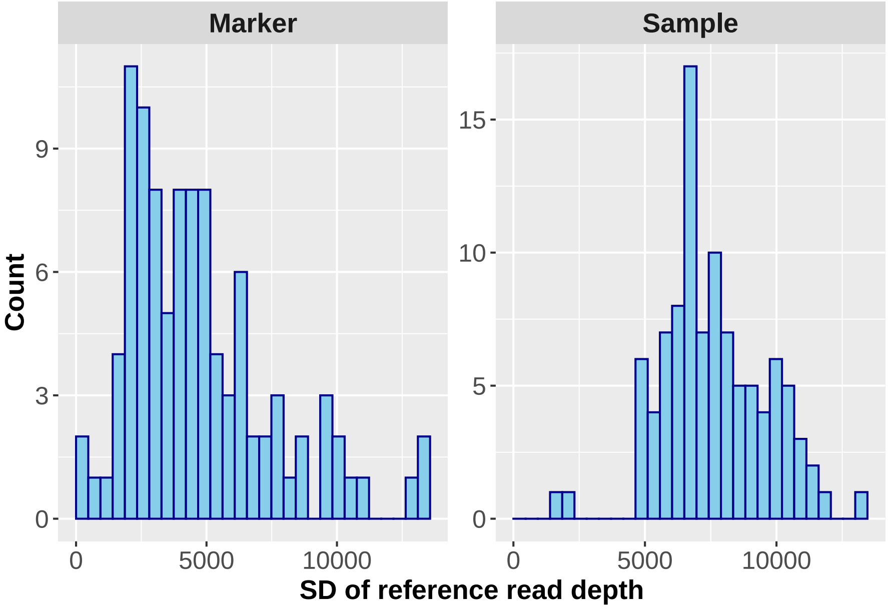


Figure 11: SD of alternative read depth per marker and per sample.

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

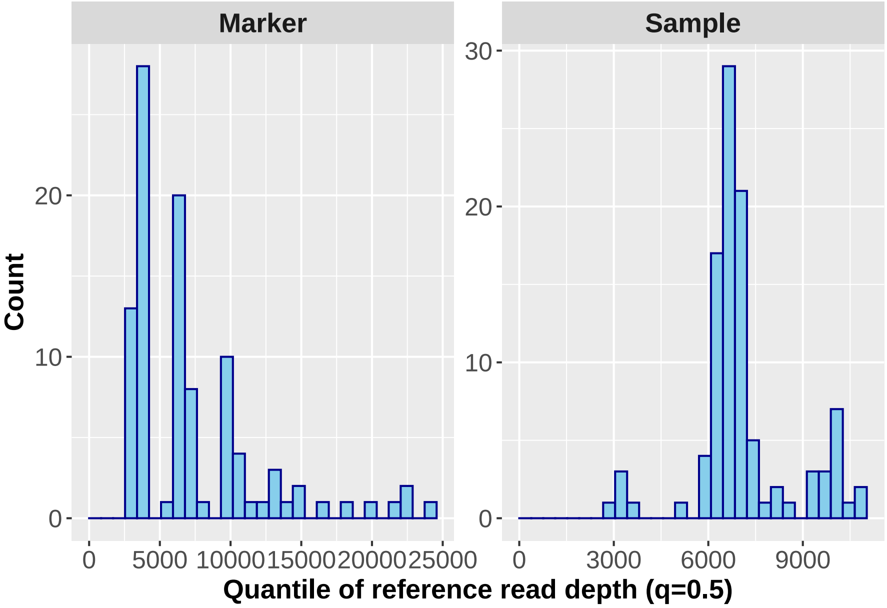


Figure 12: Quantile of reference read depth per marker and per sample.

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

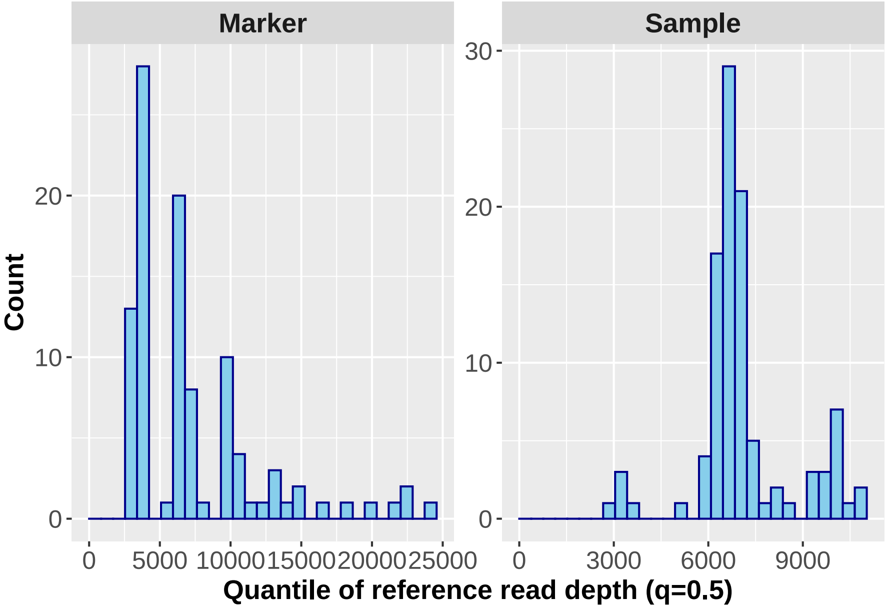
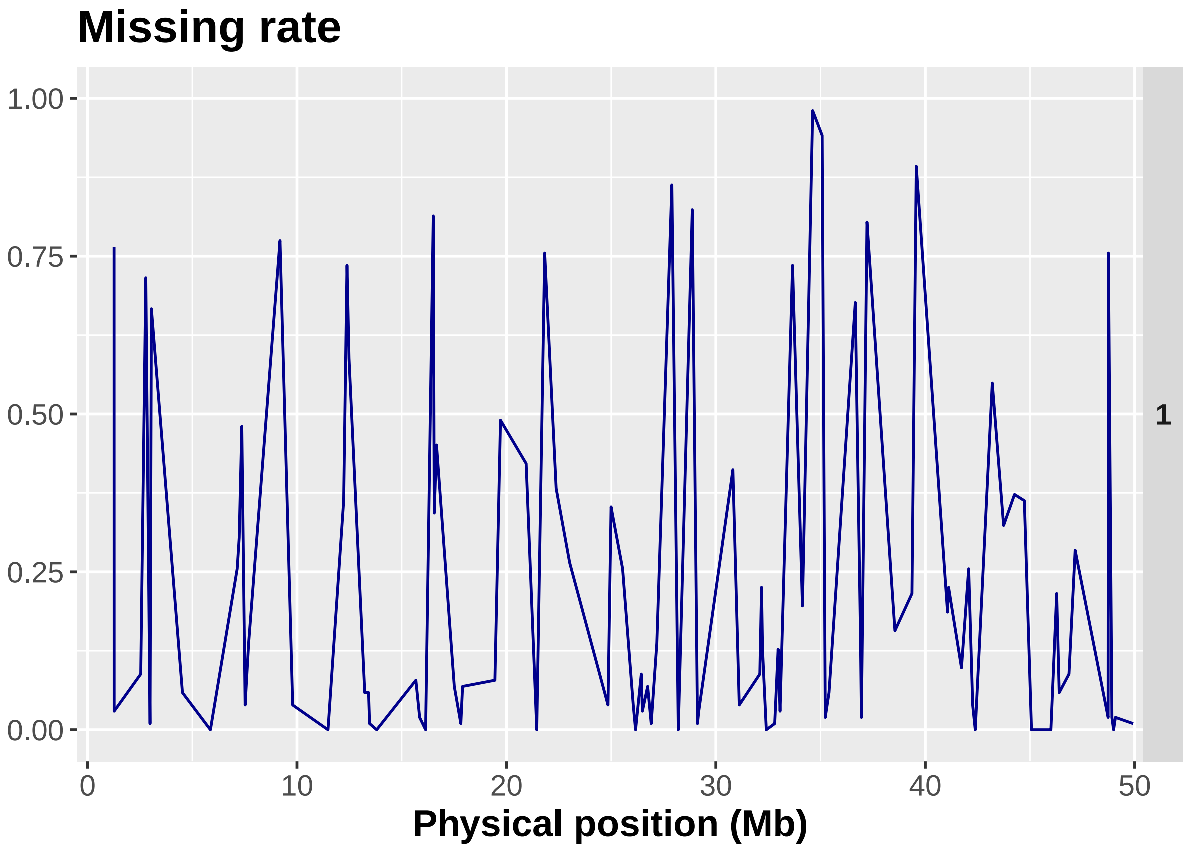


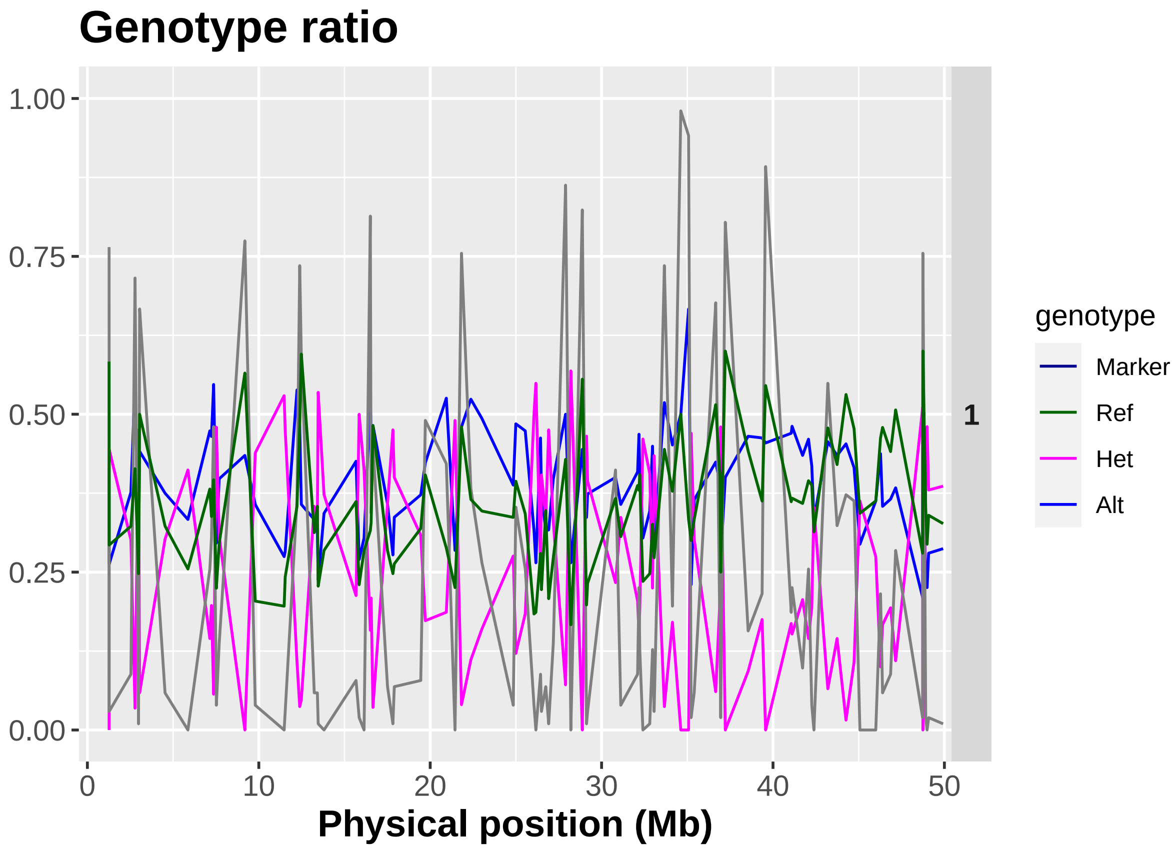
Figure 13: Quantile of alternative read depth per marker and per sample.

`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.

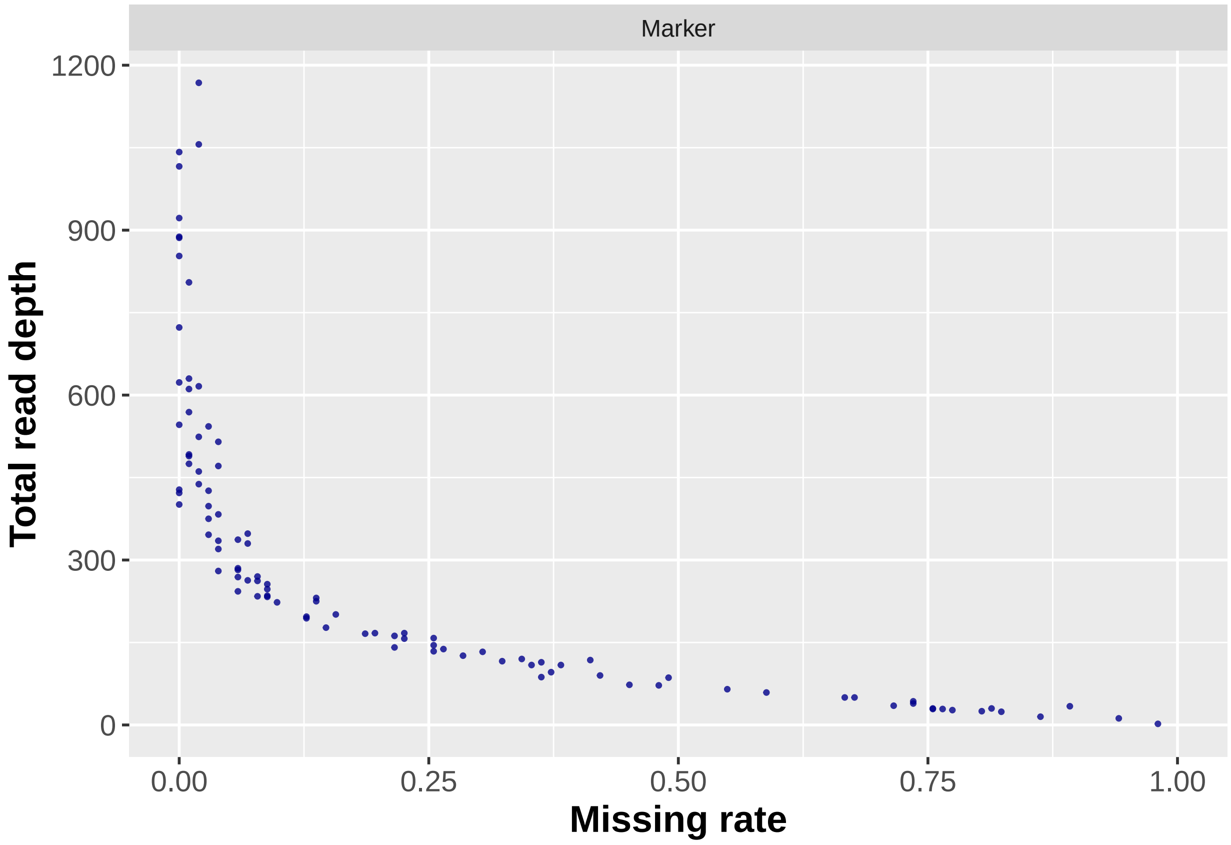
```
plotGBSR(gdata, stats = "missing", coord = c(6, 2)) # coord controls the number of rows and columns of f
```



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



```
pairsGBSR(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] 14 29 30 12 25 17
```

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

```
## [1] 78  0 16 51 30  5
```

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

```
## [1]  0 44 28  1 48  2
```

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

```
## [1]  0  1 44 18 33 25
```

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

```
## [1] 10 26 35 16 28 15
```

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

```
## [1]  0 73 16 11 11 49
```

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

```
## [1] 78  3  9 73  1 68
```

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

```
## [1] 22 26 24 20 26 21
```

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

```
## [1]  28 102  88  25  98  36
```

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

```
## [1] 156   1  76 120  93  35
```

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

```
## [1]  20  96  98  33 104  32
```

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

```
## [1]   0 147  76  40  55 123
```

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

```
## [1] 156   6  18 146   2 136
```

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

```
## [1] 44 52 48 40 52 42
```

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

```
## [1]  18 173 118  13 306  27
```

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

```
## [1] 298   1 143 230 200  65
```

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

```
## [1]  11 173 129  22 305  23
```

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

```
## [1]   0 301 130  77 111 236
```

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

```
## [1]  29 346 247  35 611  50
```

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

```
## [1] 298 302 273 307 311 301
```

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

```
## [1]  4319.066  7929.774  6861.562  3262.526 13979.806  4753.598
```

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

```
## [1] 12820.513  3311.258  8730.159 10857.763 10207.727  7198.228
```

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

```
## [1]  3742.007  8214.149  6874.512  4256.045 13350.078  4514.687
```

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

```
## [1]       NaN 13468.767  7936.508  8648.770  8111.663 10595.313
```

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

```
## [1] 1676.8778 4339.4326 3704.8188  179.6336 8476.8460 2586.4207
```

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

```
## [1]  8303.732        NA  6184.475 10609.018  9463.745  5726.696
```

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

```
## [1] 1078.841 4738.968 3499.503 1929.963 7448.037 2542.482
```

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

```
## [1]        NA 10632.409  7268.873  7087.210  7419.960 10282.999
```

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

```
## [1]  3430.623  6779.661  6568.162  3289.474 12861.736  3546.099
```

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

```
## [1] 10067.114  3311.258  7326.007  6514.658  6430.868  3322.259
```

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

```
## [1]  3430.623  6861.245  6756.757  3322.259 12904.435  3472.222
```

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

```
## [1]       NA 9933.774 3663.004 6514.658 6430.868 6644.518
```

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

```
## [1] 0.4166667 0.4848485 0.4731183 0.4310345 0.4851485 0.4705882
```

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

```
## [1] 0.000000000 0.006756757 0.500000000 0.250000000 0.371621622 0.221518987
```

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

```
## [1] 20 96 88 25 98 32
```

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

```
## [1]  0  1 76 40 55 35
```

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

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

```
## [1] 0.5833333 0.2929293 0.3225806 0.4137931 0.2475248 0.5000000
```

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

```
## [1] 0.00000000 0.44444444 0.30107527 0.03448276 0.47524752 0.05882353
```

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

```
## [1] 0.4166667 0.2626263 0.3763441 0.5517241 0.2772277 0.4411765
```

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

```
## [1] 0.764705882 0.029411765 0.088235294 0.715686275 0.009803922 0.666666667
```

The proportion of each allele counts.

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

```
## [1] 0.5833333 0.5151515 0.4731183 0.4310345 0.4851485 0.5294118
```

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

```
## [1] 0.4166667 0.4848485 0.5268817 0.5689655 0.5148515 0.4705882
```

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

```
## [1] 0.764705882 0.029411765 0.088235294 0.715686275 0.009803922 0.666666667
```

The proportion of each allele read counts.

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

```
## [1] 0.6206897 0.5000000 0.4777328 0.3714286 0.5008183 0.5400000
```

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

```
## [1] 0.3793103 0.5000000 0.5222672 0.6285714 0.4991817 0.4600000
```

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/softDevel/GBScleanR/inst/extdata/sim_pop.gds (49.1K)
## +    [  ] *
## |--+ sample.id   { Str8 102 LZMA_ra(16.9%), 245B }
## |--+ snp.id   { Int32 100 LZMA_ra(48.5%), 201B }
## |--+ snp.rs.id   { Str8 100 LZMA_ra(77.4%), 233B }
## |--+ snp.position   { Int32 100 LZMA_ra(104.5%), 425B }
## |--+ snp.allele   { Str8 100 LZMA_ra(22.5%), 97B }
## |--+ genotype   { Bit2 102x100 LZMA_ra(95.0%), 2.4K } *
## |--+ annotation   [  ]
## |  |--+ info   [  ]
## |  \--+ format   [  ]
## |     |--+ AD   [  ] *
## |     |  |--+ data   { VL_Int 102x200 LZMA_ra(32.5%), 6.5K } *
## |     |  |--+ norm   { Float32 200x102 LZMA_ra(13.3%), 10.6K }
## |     |  |--+ filt.scan   { Bit1 100x102 LZMA_ra(86.1%), 1.1K }
## |     |  \--+ filt.data   { VL_Int 102x200 LZMA_ra(17.2%), 3.4K }
## |     \--+ DP   [  ] *
## |        \--+ data   { VL_Int 102x100 LZMA_ra(41.1%), 4.2K } *
## |--+ snp.chromosome.name   { Str8 100 LZMA_ra(43.0%), 93B }
## |--+ snp.chromosome   { Int8 100 LZMA_ra(82.0%), 89B }
## |--+ estimated.haplotype   { Bit6 0 LZMA_ra, 18B }
## |--+ corrected.genotype   { Bit2 0 LZMA_ra, 18B }
## |--+ parents.genotype   { Bit2 0 LZMA_ra, 18B }
## \--+ filt.genotype   { Bit2 102x100 LZMA_ra(53.6%), 1.3K }
## An object of class 'SnpAnnotationDataFrame'
##   snps: 1 2 ... 100 (100 total)
##   varLabels: snpID chromosome ... qtileReadAlt0.5 (23 total)
##   varMetadata: labelDescription
## An object of class 'ScanAnnotationDataFrame'
##   scans: 1 2 ... 102 (102 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  TRUE  TRUE FALSE  TRUE FALSE
```

```
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] 52
```

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

```
## snps
##  100
```

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.

```
closeGDS(gdata)
```

Session information

```
sessionInfo()
```

```
## R version 4.1.1 (2021-08-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.3 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=C
##  [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_0.99.0    GWASTools_1.38.0    Biobase_2.52.0
## [4] BiocGenerics_0.38.0
##
## loaded via a namespace (and not attached):
##   [1] nlme_3.1-152           bitops_1.0-7           matrixStats_0.61.0
##   [4] fs_1.5.0               usethis_2.0.1          devtools_2.4.2
##   [7] bit64_4.0.5            rprojroot_2.0.2        GenomeInfoDb_1.28.4
##  [10] tools_4.1.1            backports_1.2.1        utf8_1.2.2
##  [13] R6_2.5.1               DBI_1.1.1              mgcv_1.8-37
##  [16] colorspace_2.0-2       DNAcopy_1.66.0         withr_2.4.2
##  [19] tidyselect_1.1.1       prettyunits_1.1.1      processx_3.5.2
##  [22] bit_4.0.4              compiler_4.1.1         cli_3.0.1
##  [25] quantreg_5.86          expm_0.999-6           mice_3.13.0
##  [28] SparseM_1.81           xml2_1.3.2             desc_1.4.0
##  [31] sandwich_3.0-1         labeling_0.4.2         scales_1.1.1
##  [34] lmtest_0.9-38          quantsmooth_1.58.0     callr_3.7.0
##  [37] digest_0.6.28          stringr_1.4.0          GWASExactHW_1.01
##  [40] rmarkdown_2.11         XVector_0.32.0         htmltools_0.5.2
##  [43] pkgconfig_2.0.3        sessioninfo_1.1.1      fastmap_1.1.0
##  [46] rlang_0.4.11           rstudioapi_0.13        RSQLite_2.2.8
##  [49] farver_2.1.0           generics_0.1.0         zoo_1.8-9
##  [52] dplyr_1.0.7            RCurl_1.98-1.5         magrittr_2.0.1
##  [55] GenomeInfoDbData_1.2.6 Matrix_1.3-4           Rcpp_1.0.7
##  [58] munsell_0.5.0          S4Vectors_0.30.1       fansi_0.5.0
##  [61] lifecycle_1.0.1        yaml_2.2.1             stringi_1.7.4
##  [64] zlibbioc_1.38.0        pkgbuild_1.2.0         grid_4.1.1
##  [67] formula.tools_1.7.1    blob_1.2.2             crayon_1.4.1
##  [70] lattice_0.20-44        Biostrings_2.60.2      splines_4.1.1
##  [73] knitr_1.34             ps_1.6.0               pillar_1.6.3
##  [76] GenomicRanges_1.44.0   logistf_1.24           gdsfmt_1.28.1
##  [79] stats4_4.1.1           pkgload_1.2.2          glue_1.4.2
##  [82] evaluate_0.14          RcppParallel_5.1.4     data.table_1.14.2
##  [85] remotes_2.4.0          operator.tools_1.6.3   vctrs_0.3.8
##  [88] testthat_3.0.4         MatrixModels_0.5-0     gtable_0.3.0
##  [91] purrr_0.3.4            tidyr_1.1.4            SeqArray_1.32.0
##  [94] cachem_1.0.6           ggplot2_3.3.5          xfun_0.26
##  [97] broom_0.7.9            roxygen2_7.1.2         survival_3.2-13
## [100] tibble_3.1.4           conquer_1.0.2          memoise_2.0.0
## [103] IRanges_2.26.0         ellipsis_0.3.2
```