Introduction to SeqVarTools

Stephanie M. Gogarten

November 1, 2026

Contents

1 Introduction 1

2 Converting a Variant Call Format (VCF) file 1

3 Applying methods to subsets of data 6

4 Examples 6

4.1 Transition/transversion ratio 6

4.2 Heterozygosity 10

4.3 Principal Component Analysis 11

4.4 Hardy-Weinberg Equilibrium 12

4.5 Mendelian errors 14

4.6 Association tests 15

1 Introduction

SeqArray provides an alternative to the Variant Call Format (VCF) for storage of variants called from sequencing data, enabling efficient storage, fast access to subsets of the data, and rapid computation.

SeqVarTools provides an interface to the *SeqArray* storage format with tools for many common tasks in variant analysis.

It is highly recommended to read the vignette in *SeqArray* in addition to this document to understand the data structure and full array of features.

2 Converting a Variant Call Format (VCF) file

To work with *SeqVarTools*, we must first convert a VCF file into the *SeqArray* GDS format. All information in the VCF file is preserved in the resulting GDS file.

```
> library(SeqVarTools)
> vcffile <- seqExampleFileName("vcf")
> gdsfile <- "tmp.gds"
> seqVCF2GDS(vcffile, gdsfile, verbose=FALSE)
> gds <- seqOpen(gdsfile)
> gds

Object of class "SeqVarGDSClass"
File: /tmp/RtmpC21OU7/Rbuild109e6ea4db28/SeqVarTools/vignettes/tmp.gds (163.0K)
+    [  ] *
|--+ description   [  ] *
|--+ sample.id   { Str8 90 LZMA_ra(34.7%), 257B } *
|--+ variant.id   { Int32 1348 LZMA_ra(16.7%), 905B } *
|--+ chromosome   { Str8 1348 LZMA_ra(4.39%), 157B } *
|--+ position   { Int32 1348 LZMA_ra(64.4%), 3.4K } *
|--+ allele   { Str8 1348 LZMA_ra(16.6%), 901B } *
|--+ genotype   [  ] *
|  |--+ data   { Bit2 2x90x1348 LZMA_ra(26.3%), 15.6K } *
|  |--+ extra.index   { Int32 3x0 LZMA_ra, 18B } *
|  \--+ extra   { Int16 0 LZMA_ra, 18B }
|--+ phase   [  ]
|  |--+ data   { Bit1 90x1348 LZMA_ra(0.86%), 137B } *
|  |--+ extra.index   { Int32 3x0 LZMA_ra, 18B } *
|  \--+ extra   { Bit1 0 LZMA_ra, 18B }
|--+ annotation   [  ]
|  |--+ id   { Str8 1348 LZMA_ra(38.3%), 5.5K } *
|  |--+ qual   { Float32 1348 LZMA_ra(2.11%), 121B } *
|  |--+ filter   { Int32,factor 1348 LZMA_ra(2.11%), 121B } *
|  |--+ info   [  ]
|  |  |--+ AA   { Str8 1328 LZMA_ra(22.1%), 593B } *
|  |  |--+ AC   { Int32 1348 LZMA_ra(24.1%), 1.3K } *
|  |  |--+ AN   { Int32 1348 LZMA_ra(19.6%), 1.0K } *
|  |  |--+ DP   { Int32 1348 LZMA_ra(47.7%), 2.5K } *
|  |  |--+ HM2   { Bit1 1348 LZMA_ra(145.6%), 253B } *
|  |  |--+ HM3   { Bit1 1348 LZMA_ra(145.6%), 253B } *
|  |  |--+ OR   { Str8 1348 LZMA_ra(19.6%), 341B } *
|  |  |--+ GP   { Str8 1348 LZMA_ra(24.3%), 3.8K } *
|  |  \--+ BN   { Int32 1348 LZMA_ra(20.7%), 1.1K } *
|  \--+ format   [  ]
|     \--+ DP   [  ] *
|        \--+ data   { VL_Int 90x1348 LZMA_ra(70.8%), 115.2K } *
\--+ sample.annotation   [  ]
```

We can look at some basic information in this file, such as the reference and alternate alleles.

```
> head(refChar(gds))
[1] "T" "G" "G" "T" "G" "C"
> head(altChar(gds))
[1] "C" "A" "A" "C" "C" "T"
```

How many alleles are there for each variant?

```
> table(nAlleles(gds))

   2    3
1346    2
```

Two variants have 3 alleles (1 REF and 2 ALT). We can extract the second alterate allele for these variants by using the argument `n=2` to `altChar`.

```
> multi.allelic <- which(nAlleles(gds) > 2)
> altChar(gds)[multi.allelic]
[1] "T,CT" "T,AT"
> altChar(gds, n=1)[multi.allelic]
[1] "T" "T"
> altChar(gds, n=2)[multi.allelic]
[1] "CT" "AT"
```

These two sites have three alleles, two are each single nucleotides and the third is a dinucleotide, representing an indel.

```
> table(isSNV(gds))

FALSE  TRUE
    2  1346
> isSNV(gds)[multi.allelic]
[1] FALSE FALSE
```

Chromosome and position can be accessed as vectors or as a *GRanges* object.

```
> head(seqGetData(gds, "chromosome"))
[1] "1" "1" "1" "1" "1" "1"
> head(seqGetData(gds, "position"))
[1] 1105366 1105411 1110294 3537996 3538692 3541597
> granges(gds)
```

```
GRanges object with 1348 ranges and 0 metadata columns:
       seqnames    ranges strand
          <Rle> <IRanges>  <Rle>
     1        1   1105366      *
     2        1   1105411      *
     3        1   1110294      *
     4        1   3537996      *
     5        1   3538692      *
   ...      ...       ...    ...
  1344       22  43690908      *
  1345       22  43690970      *
  1346       22  43691009      *
  1347       22  43691073      *
  1348       22  48958933      *
  -------
  seqinfo: 22 sequences from an unspecified genome; no seqlengths
```

We can also find the sample and variant IDs.

```
> head(seqGetData(gds, "sample.id"))
[1] "NA06984" "NA06985" "NA06986" "NA06989" "NA06994" "NA07000"
> head(seqGetData(gds, "variant.id"))
[1] 1 2 3 4 5 6
```

The variant IDs are sequential integers created by `seqVCF2GDS`. We may wish to rename them to something more useful. Note the "annotation/" prefix required to retrive the "id" variable. We need to confirm that the new IDs are unique (which is not always the case for the "annotation/id" field).

```
> rsID <- seqGetData(gds, "annotation/id")
> head(rsID)
[1] "rs111751804" "rs114390380" "rs1320571"   "rs2760321"   "rs2760320"
[6] "rs116230480"
> length(unique(rsID)) == length(rsID)
[1] TRUE
```

Renaming the variant IDs requires modifying the GDS file, so we have to close it first.

```
> seqClose(gds)
> setVariantID(gdsfile, rsID)
> gds <- seqOpen(gdsfile)
> head(seqGetData(gds, "variant.id"))
```

```
[1] "rs111751804" "rs114390380" "rs1320571"   "rs2760321"   "rs2760320"
[6] "rs116230480"
```

Note that using character strings for variant.id instead of integers may decrease performance for large datasets.

`getGenotype` transforms the genotypes from the internal storage format to VCF-like character strings.

```
> geno <- getGenotype(gds)
> dim(geno)
```

```
[1]   90 1348
```

```
> geno[1:10, 1:5]
```

```
         variant
sample    rs111751804 rs114390380 rs1320571 rs2760321 rs2760320
  NA06984 NA          NA          "0/0"     "1/0"     "0/0"
  NA06985 NA          NA          "0/0"     "1/1"     "0/0"
  NA06986 "0/0"       "0/0"       "0/0"     "1/1"     "0/0"
  NA06989 NA          NA          "0/0"     NA        "0/0"
  NA06994 NA          NA          "0/0"     NA        "0/0"
  NA07000 "0/0"       "0/0"       "0/0"     "1/1"     "1/0"
  NA07037 "0/0"       "0/0"       "0/0"     "1/1"     "0/0"
  NA07048 "0/0"       "0/0"       "0/0"     "1/1"     "0/0"
  NA07051 "0/0"       "1/0"       "0/0"     "1/1"     "0/0"
  NA07346 "0/0"       "0/0"       "0/0"     "1/1"     "0/0"
```

`getGenotypeAlleles` returns the nucleotides instead of integers.

```
> geno <- getGenotypeAlleles(gds)
> geno[1:10, 1:5]
```

```
         variant
sample    rs111751804 rs114390380 rs1320571 rs2760321 rs2760320
  NA06984 NA          NA          "G/G"     "C/T"     "G/G"
  NA06985 NA          NA          "G/G"     "C/C"     "G/G"
  NA06986 "T/T"       "G/G"       "G/G"     "C/C"     "G/G"
  NA06989 NA          NA          "G/G"     NA        "G/G"
  NA06994 NA          NA          "G/G"     NA        "G/G"
  NA07000 "T/T"       "G/G"       "G/G"     "C/C"     "C/G"
  NA07037 "T/T"       "G/G"       "G/G"     "C/C"     "G/G"
  NA07048 "T/T"       "G/G"       "G/G"     "C/C"     "G/G"
  NA07051 "T/T"       "A/G"       "G/G"     "C/C"     "G/G"
  NA07346 "T/T"       "G/G"       "G/G"     "C/C"     "G/G"
```

3 Applying methods to subsets of data

If a dataset is large, we may want to work with subsets of the data at one time. We can use `applyMethod` to select a subset of variants and/or samples. `applyMethod` is essentially a wrapper for `seqSetFilter` that enables us to apply a method or function to a data subset in one line. If it is desired to use the same filter multiple times, it may be more efficient to set the filter once instead of using `applyMethod`.

```
> samp.id <- seqGetData(gds, "sample.id")[1:10]
> var.id <- seqGetData(gds, "variant.id")[1:5]
> applyMethod(gds, getGenotype, variant=var.id, sample=samp.id)

# of selected samples: 10
# of selected variants: 5
         variant
sample    rs111751804 rs114390380 rs1320571 rs2760321 rs2760320
  NA06984 NA          NA          "0/0"     "1/0"     "0/0"
  NA06985 NA          NA          "0/0"     "1/1"     "0/0"
  NA06986 "0/0"       "0/0"       "0/0"     "1/1"     "0/0"
  NA06989 NA          NA          "0/0"     NA        "0/0"
  NA06994 NA          NA          "0/0"     NA        "0/0"
  NA07000 "0/0"       "0/0"       "0/0"     "1/1"     "1/0"
  NA07037 "0/0"       "0/0"       "0/0"     "1/1"     "0/0"
  NA07048 "0/0"       "0/0"       "0/0"     "1/1"     "0/0"
  NA07051 "0/0"       "1/0"       "0/0"     "1/1"     "0/0"
  NA07346 "0/0"       "0/0"       "0/0"     "1/1"     "0/0"
```

As an alternative to specifying variant ids, we can use a GRanges object to select a range on chromosome 22.

```
> library(GenomicRanges)
> gr <- GRanges(seqnames="22", IRanges(start=1, end=250000000))
> geno <- applyMethod(gds, getGenotype, variant=gr)

# of selected variants: 23

> dim(geno)

[1] 90 23
```

4 Examples

4.1 Transition/transversion ratio

The transition/transversion ratio (TiTv) is frequently used as a quality metric. We can calculate TiTv over the entire dataset or by sample.

```
> titv(gds)

[1] 3.562712

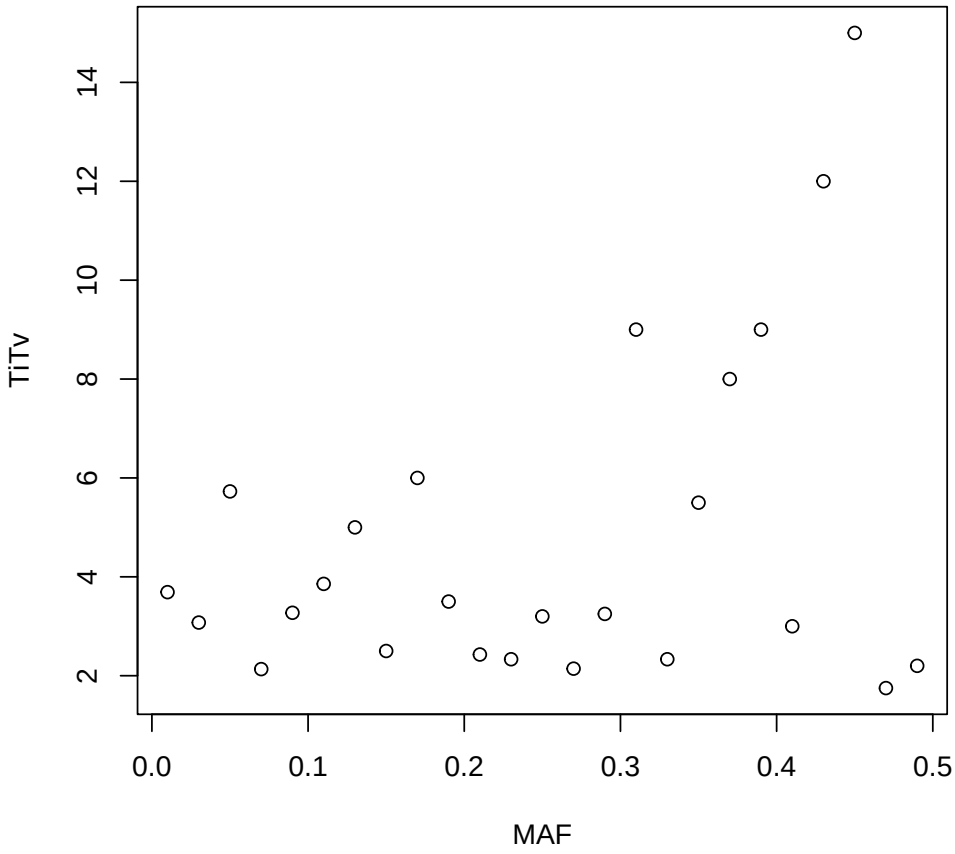
> head(titv(gds, by.sample=TRUE))

[1] 4.352941 3.791667 3.439394 3.568966 3.750000 3.646154
```

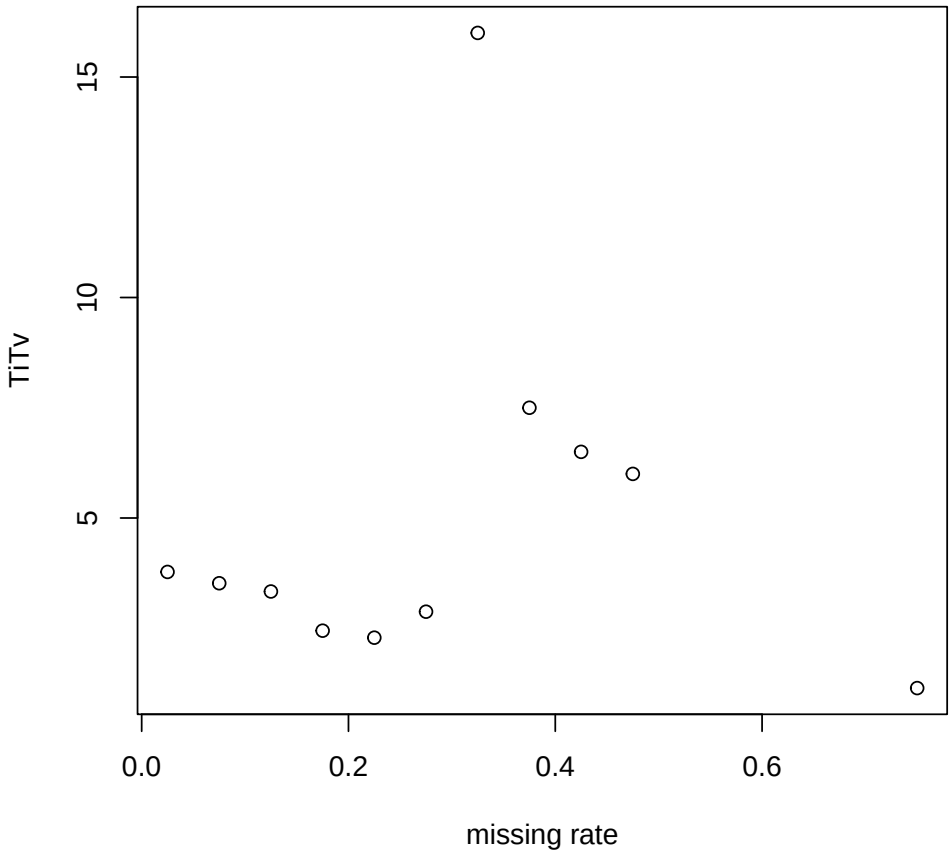
Alternatively, we can plot TiTv binned by various metrics (allele frequency, missing rate, depth) to assess variant quality. We need the ids of the variants that fall in each bin.

```
> binVar <- function(var, names, breaks) {
+   names(var) <- names
+   var <- sort(var)
+   mids <- breaks[1:length(breaks)-1] +
+     (breaks[2:length(breaks)] - breaks[1:length(breaks)-1])/2
+   bins <- cut(var, breaks, labels=mids, right=FALSE)
+   split(names(var), bins)
+ }
```

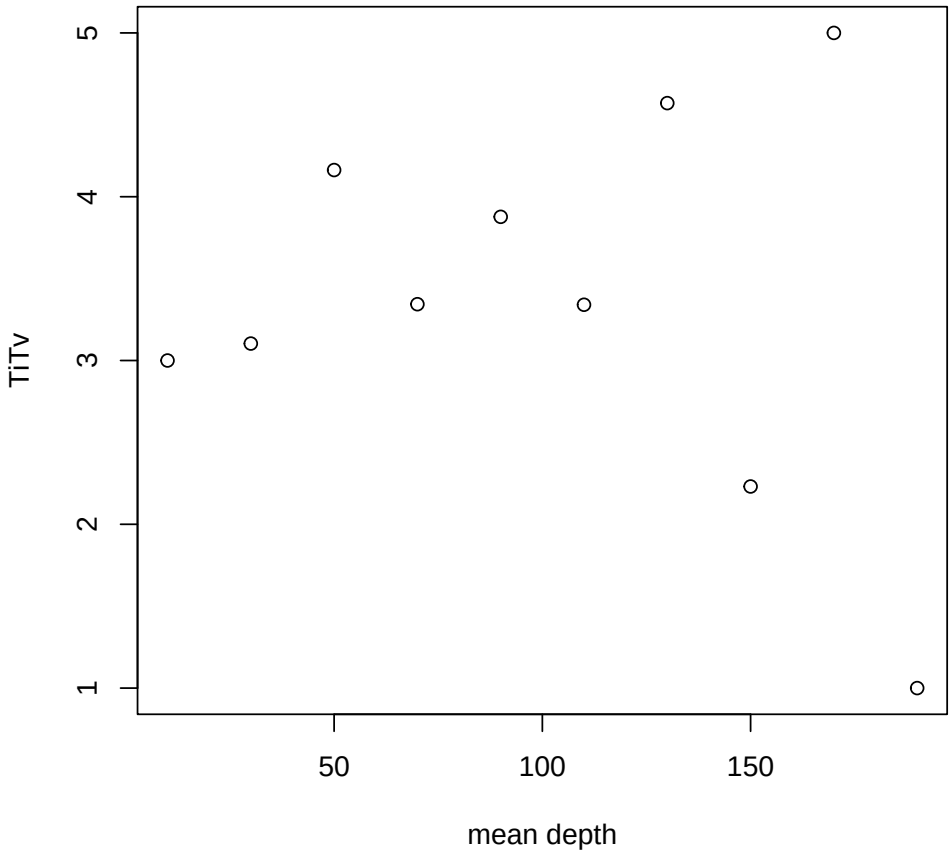
```
> variant.id <- seqGetData(gds, "variant.id")
> afreq <- alleleFrequency(gds)
> maf <- pmin(afreq, 1-afreq)
> maf.bins <- binVar(maf, variant.id, seq(0,0.5,0.02))
> nbins <- length(maf.bins)
> titv.maf <- rep(NA, nbins)
> for (i in 1:nbins) {
+   capture.output(titv.maf[i] <- applyMethod(gds, titv, variant=maf.bins[[i]]))
+ }
> plot(as.numeric(names(maf.bins)), titv.maf, xlab="MAF", ylab="TiTv")
```




```
> miss <- missingGenotypeRate(gds)
> miss.bins <- binVar(miss, variant.id, c(seq(0,0.5,0.05),1))
> nbins <- length(miss.bins)
> titv.miss <- rep(NA, nbins)
> for (i in 1:nbins) {
+   capture.output(titv.miss[i] <- applyMethod(gds, titv, variant=miss.bins[[i]]))
+ }
> plot(as.numeric(names(miss.bins)), titv.miss, xlab="missing rate", ylab="TiTv")
```




```
> depth <- seqApply(gds, "annotation/format/DP", mean, margin="by.variant",
+                   as.is="double", na.rm=TRUE)
> depth.bins <- binVar(depth, variant.id, seq(0,200,20))
> nbins <- length(depth.bins)
> titv.depth <- rep(NA, nbins)
> for (i in 1:nbins) {
+   capture.output(titv.depth[i] <- applyMethod(gds, titv, variant=depth.bins[[i]]))
+ }
> plot(as.numeric(names(depth.bins)), titv.depth, xlab="mean depth", ylab="TiTv")
```



4.2 Heterozygosity

We will calculate the ratio of heterozygotes to non-reference homozygotes by sample. First, we filter the data to exclude any variants with missing rate < 0.1 or heterozygosity$> 0.6\%$.

```
> miss.var <- missingGenotypeRate(gds, margin="by.variant")
> het.var <- heterozygosity(gds, margin="by.variant")
> filt <- seqGetData(gds, "variant.id")[miss.var <= 0.1 & het.var <= 0.6]
```

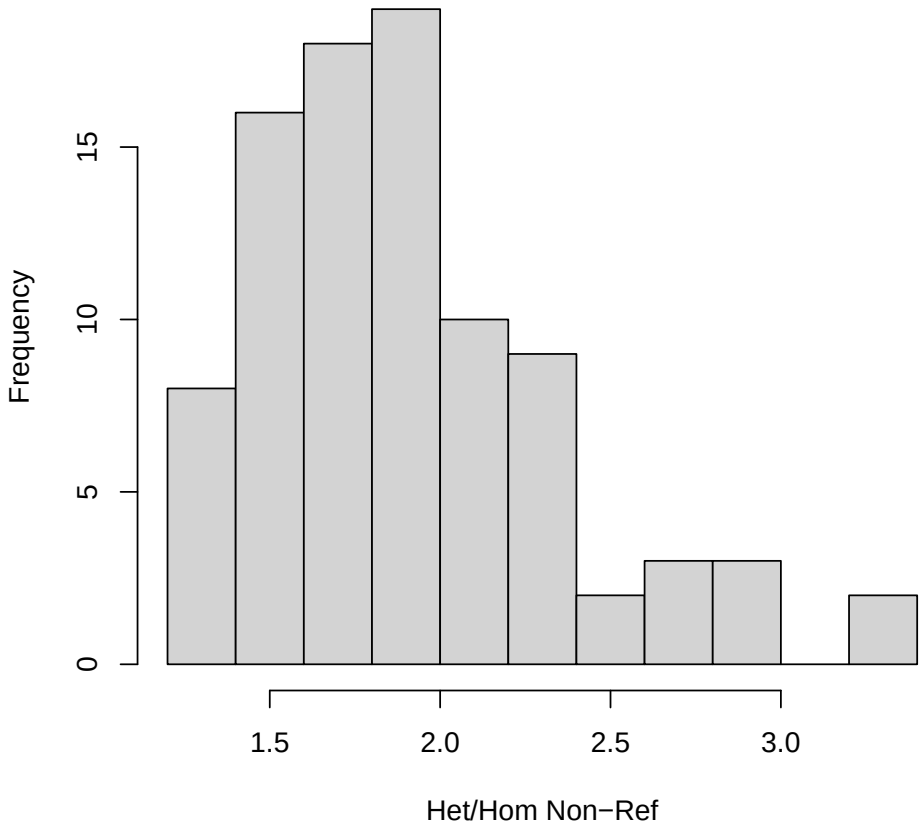
We calculate the heterozygosity and homozyogity by sample, using only the variants selected above.

```
> seqSetFilter(gds, variant.id=filt)

# of selected variants: 1,078
```

```
> hethom <- hethom(gds)
> hist(hethom, main="", xlab="Het/Hom Non-Ref")
> seqSetFilter(gds)

# of selected samples: 90
# of selected variants: 1,348
```



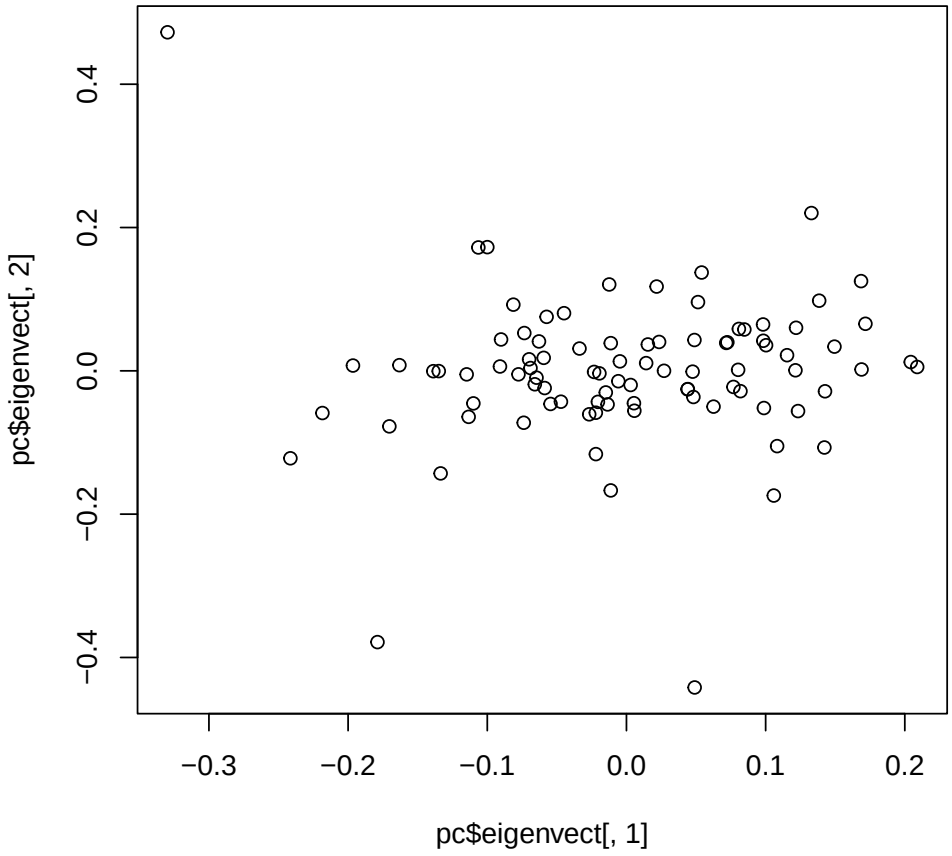
4.3 Principal Component Analysis

We can do Principal Component Analysis (PCA) to separate subjects by ancestry. All the samples in the example file are CEU, so we expect to see only one cluster.

```
> pc <- pca(gds)
> names(pc)

[1] "eigenval"  "eigenvect"
```

```
> plot(pc$eigenvect[,1], pc$eigenvect[,2])
```



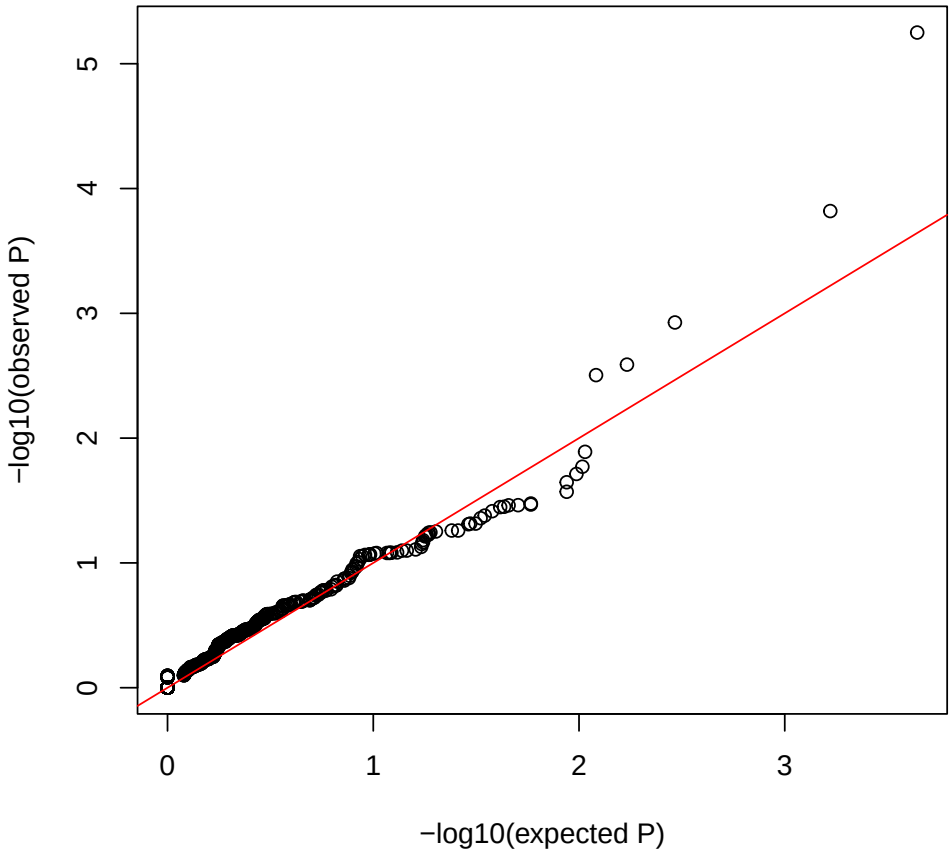
See also the packages *SNPRelate* and *GENESIS* for determining relatedness and population structure.

4.4 Hardy-Weinberg Equilibrium

We can test for deviations from Hardy-Weinberg Equilibrium (HWE), which can reveal variants of low quality. A test with permuted genotypes gives expected values under the null hypothesis of HWE.

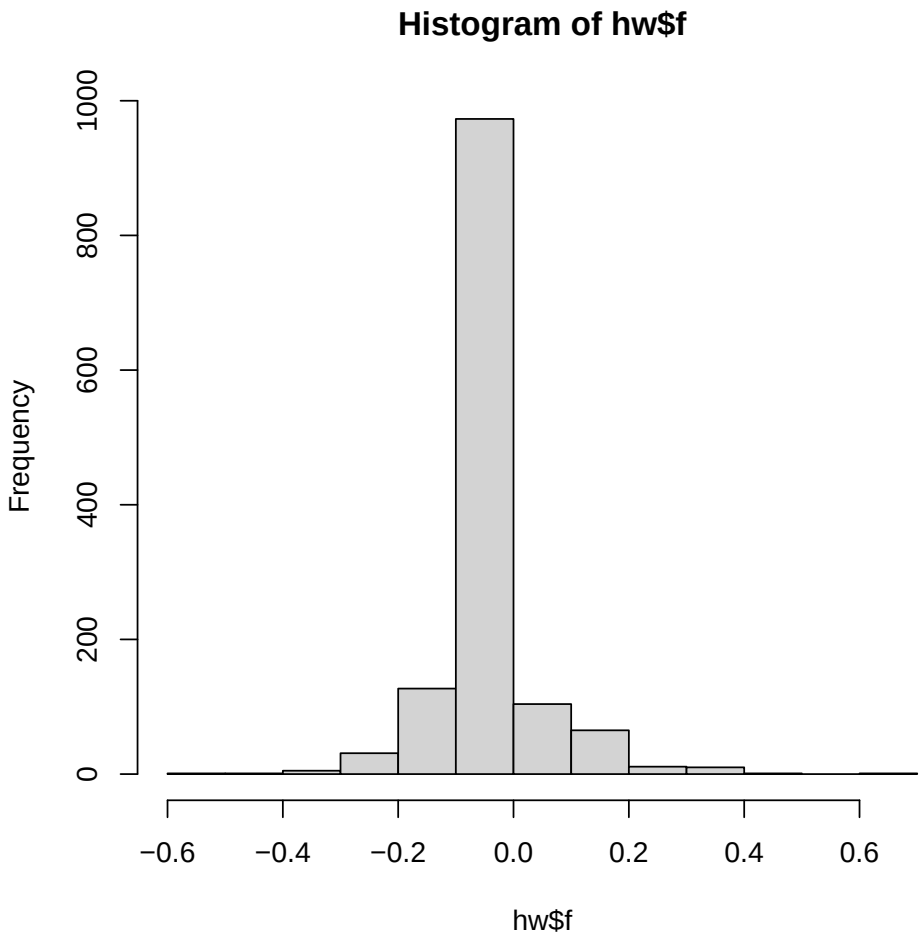
```
> hw <- hwe(gds)
> pval <- -log10(sort(hw$p))
> hw.perm <- hwe(gds, permute=TRUE)
> x <- -log10(sort(hw.perm$p))
> plot(x, pval, xlab="-log10(expected P)", ylab="-log10(observed P)")
```

```
> abline(0,1,col="red")
```



The inbreeding coefficient can also be used as a quality metric. For variants, this is 1 - observed heterozygosity / expected heterozygosity.

```
> hist(hw$f)
```



We can also calculate the inbreeding coefficient by sample.

```
> ic <- inbreedCoeff(gds, margin="by.sample")
> range(ic)

[1] -0.10035009  0.04180697
```

4.5 Mendelian errors

Checking for Mendelian errors is another way to assess variant quality. The example data contains a trio (child, mother, and father).

```
> data(pedigree)
> pedigree[pedigree$family == 1463,]

   family individ  father  mother sex sample.id
86   1463 NA12878 NA12891 NA12892   F   NA12878
87   1463 NA12889       0       0   M   NA12889
```

```
88   1463 NA12890       0       0   F   NA12890
89   1463 NA12891       0       0   M   NA12891
90   1463 NA12892       0       0   F   NA12892
```

```
> err <- mendelErr(gds, pedigree, verbose=FALSE)
> table(err$by.variant)
```

```
   0
1348
```

```
> err$by.trio
```

```
NA12878
      0
```

The example data do not have any Mendelian errors.

4.6 Association tests

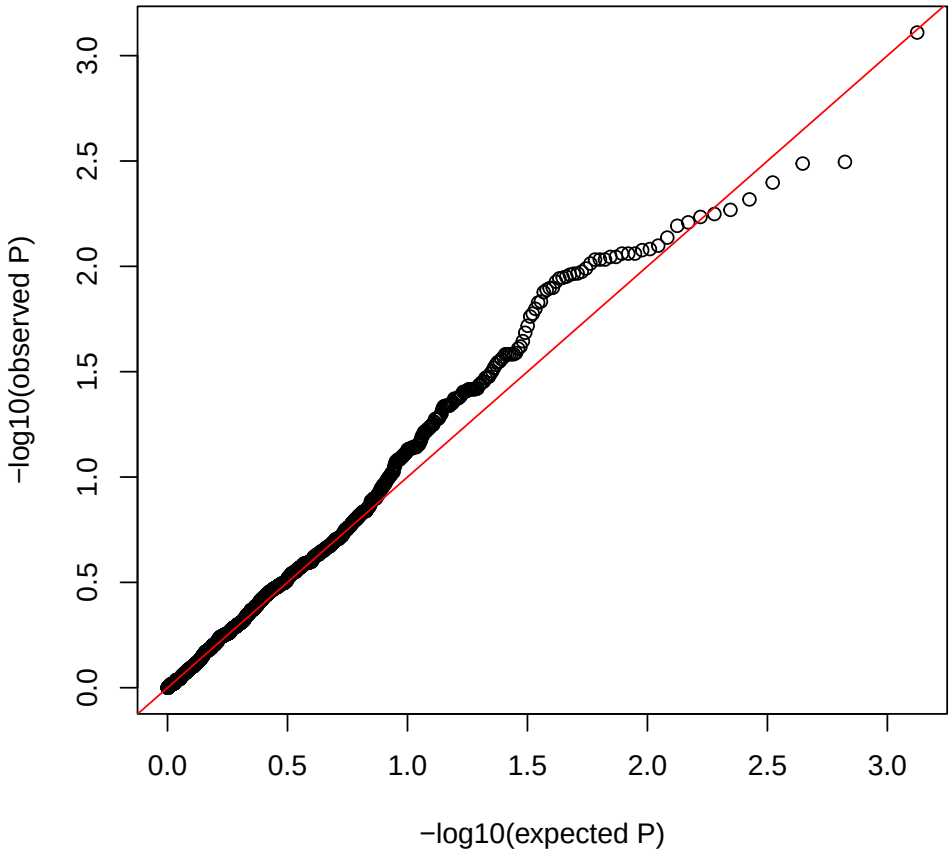
We can run single-variant association tests for continuous or binary traits. We use a *SeqVarData* object to combine the genotypes with sample annotation. We use the pedigree data from the previous section and add simulated phenotypes.

```
> library(Biobase)
> sample.id <- seqGetData(gds, "sample.id")
> pedigree <- pedigree[match(sample.id, pedigree$sample.id),]
> n <- length(sample.id)
> pedigree$phenotype <- rnorm(n, mean=10)
> pedigree$case.status <- rbinom(n, 1, 0.3)
> sample.data <- AnnotatedDataFrame(pedigree)
> seqData <- SeqVarData(gds, sample.data)
> ## continuous phenotype
> assoc <- regression(seqData, outcome="phenotype", covar="sex",
+                     model.type="linear")
> head(assoc)
```

```
   variant.id  n      freq         Est        SE  Wald.Stat   Wald.Pval
1 rs111751804 57 0.9649123  0.08781801 0.4843281 0.03287668 0.856116978
2 rs114390380 53 0.9905660 -1.31830421 0.9379235 1.97558849 0.159855923
3   rs1320571 77 0.9610390 -0.44410363 0.3978172 1.24623950 0.264271930
4   rs2760321 73 0.1232877 -0.64302148 0.2180960 8.69271675 0.003194841
5   rs2760320 89 0.9269663 -0.10837566 0.2630690 0.16971652 0.680363864
6 rs116230480 89 0.9943820 -0.60175041 0.8825409 0.46490386 0.495341022
```

```
> pval <- -log10(sort(assoc$Wald.Pval))
> n <- length(pval)
> x <- -log10((1:n)/n)
> plot(x, pval, xlab="-log10(expected P)", ylab="-log10(observed P)")
```

```
> abline(0, 1, col = "red")
```



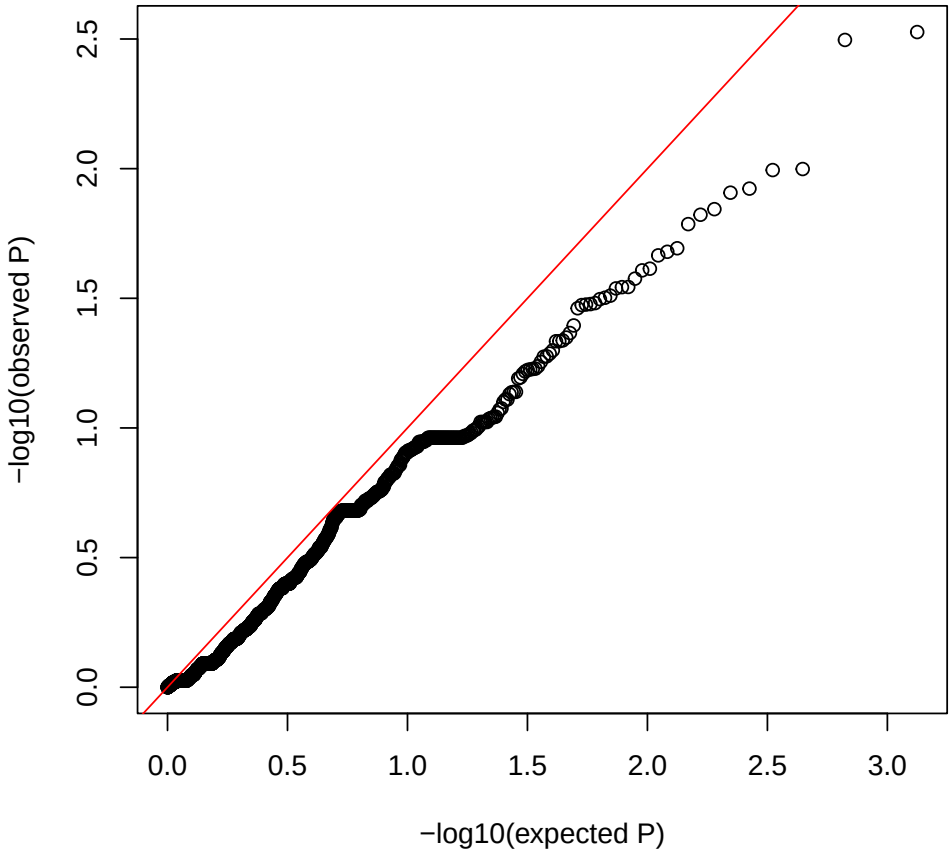
For binary phenotypes, there are two options, "logistic" and "firth". "logistic" uses `glm` and performs a Wald test. "firth" uses `logistf`. We recommend using the Firth test for rare variants [1]

```
> assoc <- regression(seqData, outcome="case.status", covar="sex",
+                     model.type="firth")
> head(assoc)
```

```
   variant.id n0 n1     freq0     freq1         Est        SE    PPL.Stat
1 rs111751804 42 15 0.9642857 0.9666667 -0.05436555 1.0360654 0.002715517
2 rs114390380 40 13 0.9875000 1.0000000 -0.08515781 1.6854721 0.002520740
3   rs1320571 59 18 0.9576271 0.9722222 -0.03571808 1.0084175 0.001248850
4   rs2760321 54 19 0.1296296 0.1052632 -0.04502697 0.5840856 0.005878726
5   rs2760320 66 23 0.9242424 0.9347826  0.15551611 0.6708210 0.054244341
6 rs116230480 65 24 0.9923077 1.0000000 -0.11954515 1.6703514 0.005028305
```

```
   PPL.Pval
1 0.9584405
2 0.9599575
3 0.9718094
4 0.9388838
5 0.8158358
6 0.9434690
```

```
> pval <- -log10(sort(assoc$PPL.Pval))
> n <- length(pval)
> x <- -log10((1:n)/n)
> plot(x, pval, xlab="-log10(expected P)", ylab="-log10(observed P)")
> abline(0, 1, col = "red")
```



For aggregate tests, see the package *GENESIS*.

```
> seqClose(gds)
```

References

[1] X. Wang. Firth logistic regression for rare variant association tests. *Front Genet*, 5:187, 2014. doi:10.3389/fgene.2014.00187.