GWAS Data Cleaning

GENEVA Coordinating Center
Department of Biostatistics
University of Washington

May 24, 2018

Contents

1 Overview

This vignette takes a user through the data cleaning steps developed and used for genome wide association data as part of the Gene Environment Association studies (GENEVA) project. This project (http://www.genevastudy.org) is a collection of whole-genome studies supported by the NIH-wide Gene-Environment Initiative. The methods used in this vignette have been published in Laurie et al. (2010).[1]

For replication purposes the data used here are taken from the HapMap project. These data were kindly provided by the Center for Inherited Disease Research (CIDR) at Johns Hopkins University and the Broad Institute of MIT and Harvard University (Broad). The data are in the same format as these centers use in providing data to investigators: the content and format of these data are a little different from those for processed data available at the HapMap project site. The data supplied here should not be used for any purpose other than this tutorial.

---

[1]Laurie, Cathy C., et al. Quality Control and Quality Assurance in Genotypic Data for Genome-Wide Association Studies. *Genetic Epidemiology* **34**, 591-602 (August 2010).

2 Preparing Data

2.1 Data formats used in GWASTools

The *GWASTools* package provides containers for storing annotation data called `SnpAnnotation-DataFrame` and `ScanAnnotationDataFrame` (derived from the `AnnotatedDataFrame` class in the *Biobase* package). The name "scan" refers to a single genotyping instance. Some subjects in a study are usually genotyped multiple times for quality control purposes, so these subjects will have duplicate scans. Throughout this tutorial, "scan" or "sample" refers to a unique genotyping instance.

The AnnotationDataFrame classes provide a way to store metadata about an annotation variable in the same R object as the variable itself. When a new column is added to an Annotation-DataFrame, we also add a column to the metadata describing what that data means. The SNP and scan AnnotationDataFrame objects are stored in R data objects (.RData files) which can be directly loaded into R.

The raw and called genotype data can be stored in the Genomic Data Structure (GDS) format (http://corearray.sourceforge.net), or the Network Common Data Format (NetCDF) (http://www.unidata.ucar.edu). In the *GWASTools* package, access to the GDS files is provided by the `GdsGenotypeReader` and `GdsIntensityReader` classes. These classes are built on top of the *gdsfmt* package and provide access to a standard set of variables defined for GWAS data. NetCDF files can be accessed with the equivalent classes `NcdfGenotypeReader` and `NcdfIntensityReader`, which are built on top of the *ncdf* package. The union classes `GenotypeReader` and `IntensityReader` allow the *GWASTools* functions to use either storage format interchangeably.

Additionally, the GDS (or NetCDF) files and SNP and scan annotation can be linked through the `GenotypeData` and `IntensityData` classes, which have slots for a `GenotypeReader` (or `IntensityReader`) object, a `SnpAnnotationDataFrame` object, and a `ScanAnnotationDataFrame` object. When an object of one of these classes is created, it performs checks to ensure that the annotation matches the data stored in the data file and all required information is present. The majority of the functions in the *GWASTools* package take `GenotypeData` or `IntensityData` objects as arguments.

2.2 Creating the SNP Annotation Data Object

All of the functions in *GWASTools* require a minimum set of variables in the SNP annotation data object. The minimum required variables are

- `snpID`, a unique integer identifier for each SNP.

- `chromosome`, an integer mapping for each chromosome, with values 1-27, mapped in order from 1-22, 23=X, 24=XY (the pseudoautosomal region), 25=Y, 26=M (the mitochondrial probes), and 27=U (probes with unknown positions). (It is possible to change the default integer mapping, e.g., to match the codes used by PLINK, but this should be done with caution. See the manual pages for more details.)

- `position`, the base position of each SNP on the chromosome.

We create the integer chromosome mapping for a few reasons. The chromosome is stored as an integer in the data files, so in order to link the SNP annotation with the data file, we use the integer values in the annotation as well. For convenience when using *GWASTools* functions, the chromosome variable is most times assumed to be an integer value. Thus, for the sex chromosomes,

we can simply use the `chromosome` values. For presentation of results, it is important to have the mapping of the integer values back to the standard designations for the chromosome names, thus the **getChromosome()** functions in the *GWASTools* objects have a `char=TRUE` option to return the characters 1-22, X, XY, Y, M, U. The position variable should hold all numeric values of the physical position of a probe. *The SNP annotation file is assumed to list the probes in order of chromosome and position within chromosome.*

```
> library(GWASTools)
> library(GWASdata)
> # Load the SNP annotation (simple data frame)
> data(illumina_snp_annot)
> # Create a SnpAnnotationDataFrame
> snpAnnot <- SnpAnnotationDataFrame(illumina_snp_annot)
> # names of columns
> varLabels(snpAnnot)

 [1] "snpID"        "chromosome"   "position"     "rsID"
 [5] "alleleA"      "alleleB"      "BeadSetID"    "IntensityOnly"
 [9] "tAA"          "tAB"          "tBB"          "rAA"
[13] "rAB"          "rBB"

> # data
> head(pData(snpAnnot))

   snpID chromosome position      rsID alleleA alleleB  BeadSetID IntensityOnly
1 999447         21 13733610 rs3132407       A       G 1185447327             1
2 999465         21 13852569 rs2775671       T       C 1169708488             0
3 999493         21 14038583 rs2775018       T       C 1192445330             0
4 999512         21 14136579 rs3115511       T       C 1149617207             0
5 999561         21 14396024 rs2822404       T       C 1149961944             0
6 999567         21 14404476 rs1556276       A       G 1149617207             0
          tAA       tAB       tBB       rAA       rAB       rBB
1 0.013743570 0.3290431 0.9184624 1.5622030 1.5927530 1.6142590
2 0.063259460 0.5440393 0.9796721 0.4431986 0.4431986 0.4431986
3 0.001315146 0.2623954 0.5362323 2.6349810 2.3149340 2.0012760
4 0.011004820 0.5692499 0.9846884 0.8781826 0.9453412 0.8209958
5 0.040206810 0.5691788 0.9902423 1.0941700 1.1270790 0.9898759
6 0.030895730 0.6842008 0.9837771 0.5954081 0.7681253 0.7900150

> # Add metadata to describe the columns
> meta <- varMetadata(snpAnnot)
> meta[c("snpID", "chromosome", "position", "rsID", "alleleA", "alleleB",
+   "BeadSetID", "IntensityOnly", "tAA", "tAB", "tBB", "rAA", "rAB", "rBB"),
+   "labelDescription"] <- c("unique integer ID for SNPs",
+   paste("integer code for chromosome: 1:22=autosomes,",
+    "23=X, 24=pseudoautosomal, 25=Y, 26=Mitochondrial, 27=Unknown"),
+   "base pair position on chromosome (build 36)",
```

```
+    "RS identifier",
+    "alelleA", "alleleB",
+    "BeadSet ID from Illumina",
+    "1=no genotypes were attempted for this assay",
+    "mean theta for AA cluster",
+    "mean theta for AB cluster",
+    "mean theta for BB cluster",
+    "mean R for AA cluster",
+    "mean R for AB cluster",
+    "mean R for BB cluster")
> varMetadata(snpAnnot) <- meta
```

Variables in the SNP annotation data frame can be accessed either with the data frame operators $ and [[or with "get" methods.

```
> snpID <- snpAnnot$snpID
> snpID <- getSnpID(snpAnnot)
> chrom <- snpAnnot[["chromosome"]]
> chrom <- getChromosome(snpAnnot)
> table(chrom)

chrom
  21    22    23    24    25    26
1000  1000  1000   100   100   100

> chrom <- getChromosome(snpAnnot, char=TRUE)
> table(chrom)

chrom
  21    22     M     X    XY     Y
1000  1000   100  1000   100   100

> position <- getPosition(snpAnnot)
> rsID <- getVariable(snpAnnot, "rsID")
```

The following methods are equivalent and can all be used on `SnpAnnotationDataFrame` objects:

AnnotatedDataFrame method	GWASTools method
pData	getAnnotation
varMetadata	getMetadata
varLabels	getVariableNames

However, only the `AnnotatedDataFrame` methods have corresponding "set" methods. New variables can be added with $ or [[. [also behaves as expected for standard data frames.

```
> tmp <- snpAnnot[,c("snpID", "chromosome", "position")]
> snp <- getAnnotation(tmp)
```

```
> snp$flag <- sample(c(TRUE, FALSE), nrow(snp), replace=TRUE)
> pData(tmp) <- snp
> meta <- getMetadata(tmp)
> meta["flag", "labelDescription"] <- "flag"
> varMetadata(tmp) <- meta
> getVariableNames(tmp)

[1] "snpID"      "chromosome" "position"   "flag"

> varLabels(tmp)[4] <- "FLAG"
> rm(tmp)
```

2.3   Creating the Scan Annotation Data Object

The scan annotation file holds attributes for each genotyping scan that are relevant to genotypic data cleaning. These data include processing variables such as tissue type, DNA extraction method, and genotype processing batch. They also include individual characteristics such as sex, race, ethnicity, and case status. Since a single subject may have been genotyped multiple times as a quality control measure, it is important to distinguish between the scanID (unique genotyping instance) and subjectID (person providing a DNA sample). The miniumum required variables for the scan annotation data object are

- scanID, a unique identifier for each scan.

- sex, coded as "M"/"F". (Note that a ScanAnnotationDataFrame object may be valid without a sex variable, but it is required for many *GWASTools* functions.)

```
> # Load the scan annotation (simple data frame)
> data(illumina_scan_annot)
> # Create a ScanAnnotationDataFrame
> scanAnnot <- ScanAnnotationDataFrame(illumina_scan_annot)
> # names of columns
> varLabels(scanAnnot)

 [1] "scanID"    "subjectID" "family"    "father"    "mother"    "CoriellID"
 [7] "race"      "sex"       "status"    "genoRunID" "plate"     "batch"
[13] "file"

> # data
> head(pData(scanAnnot))

  scanID subjectID family    father    mother CoriellID race sex status
1    280 200191449   1341         0         0   NA06985  CEU   F      1
2    281 200191449   1341         0         0   NA06985  CEU   F      1
3    282 200030290   1341 200099417 200191449   NA06991  CEU   F      0
4    283 200030290   1341 200099417 200191449   NA06991  CEU   F      0
5    284 200099417   1341         0         0   NA06993  CEU   M      1
```

```
6     285 200099417    1341             0        0    NA06993 CEU    M        1
                                genoRunID          plate batch
1 WG1000993-DNAG10-CIDR_06985@1007850397 WG0052814-AMP2     A
2 WG1000992-DNAF10-CIDR_06985@1007850586 WG0061258-AMP2     A
3 WG1000970-DNAB11-CIDR_06991@1007850444 WG0061536-AMP2     B
4 WG1000969-DNAA11-CIDR_06991@1007850587 WG0053489-AMP2     A
5 WG1000972-DNAE10-CIDR_06993@1007850591 WG0060475-AMP2     C
6 WG1000971-DNAD10-CIDR_06993@1007850421 WG0061540-AMP2     B
                  file
1 GENEVA_1M_HapMap_37.csv
2 GENEVA_1M_HapMap_58.csv
3  GENEVA_1M_HapMap_5.csv
4  GENEVA_1M_HapMap_3.csv
5 GENEVA_1M_HapMap_10.csv
6 GENEVA_1M_HapMap_71.csv

> # Add metadata to describe the columns
> meta <- varMetadata(scanAnnot)
> meta[c("scanID", "subjectID", "family", "father", "mother",
+   "CoriellID", "race", "sex", "status", "genoRunID", "plate",
+   "batch", "file"), "labelDescription"] <-
+    c("unique ID for scans",
+   "subject identifier (may have multiple scans)",
+   "family identifier",
+   "father identifier as subjectID",
+   "mother identifier as subjectID",
+   "Coriell subject identifier",
+   "HapMap population group",
+   "sex coded as M=male and F=female",
+   "simulated case/control status" ,
+   "genotyping instance identifier",
+   "plate containing samples processed together for genotyping chemistry",
+   "simulated genotyping batch",
+   "raw data file")
> varMetadata(scanAnnot) <- meta
```

As for `SnpAnnotationDataFrame`, variables in the scan annotation data frame can be accessed either with the data frame operators `$` and `[[` or with "get" methods.

```
> scanID <- scanAnnot$scanID
> scanID <- getScanID(scanAnnot)
> sex <- scanAnnot[["sex"]]
> sex <- getSex(scanAnnot)
> subjectID <- getVariable(scanAnnot, "subjectID")
```

The `AnnotatedDataFrame` methods and their *GWASTools* equivalents described in Section 2.2 apply to `ScanAnnotationDataFrame` as well.

2.4 Creating the Data Files

The data for genotype calls, allelic intensities and other variables such as B Allele Frequency are stored as GDS or NetCDF files. This format is used for the ease with which extremely large multidimensional arrays of data can be stored and accessed, as many GWAS datasets are too large to be stored in memory at one time. We will create three different GDS files to be used in subsequent cleaning and analysis steps.

All data files contain variables for `scanID`, `snpID`, `chromosome`, and `position`. The `scanID` ordering must match the `scanID` values as listed in the sample annotation object (Section 2.3). Since `snpID` is in chromosome and position order, these variables also provide a check on ordering and are often used to select subsets of SNPs for analysis. Analogous to the sample ordering, these values must match the `snpID` values listed in the SNP annotation object (Section 2.2). To prevent errors in ordering samples or SNPs, the functions in the *GWASTools* package take as arguments R objects which will return an error on creation if the sample and SNP annotation does not match the data file.

Genotype Files

The genotype files store genotypic data in 0, 1, 2 format indicating the number of "A" alleles in the genotype (i.e. AA=2, AB=1, BB=0 and missing=-1). The conversion from AB format and forward strand (or other) allele formats can be stored in the SNP annotation file.

The genotypic data are stored as a two-dimensional array, where rows are SNPs and columns are samples. To store the genotype data, the raw data files are opened and checked to ensure the sample identifier from the sample annotation file and the genotype data file match. If no discrepencies exist, the probes listed in the file are checked against the expected list of probes, then ordered and written to the data file. This process iterates over each file (sample). Diagnostics are stored as the process continues so that after the data are written one can ensure the function performed as expected.

Creating the Genotype file

We create a GDS file from a set of plain text files containing the genotypes, one file per sample. The data are written to the GDS file one sample at a time and, simultaneously, the corresponding sample identifier `scanID` is written to the sample ID variable. The `file` variable from the scan annotation holds the name of the raw data file for each sample/scan; these are the files we must read in to get genotype data for each sample.

The function `createDataFile` creates the common SNP variables as described above. In this case, we also want the `genotype` variable to be created, so the `variables` argument must be set to `"genotype"`. `col.nums` is an integer vector indicating which columns of the raw text file contain variables for input. A set of diagnostic values are written and stored in `diag.geno`, so we must look at those to ensure no errors occurred.

```
> # Define a path to the raw data files
> path <- system.file("extdata", "illumina_raw_data", package="GWASdata")
> geno.file <- "tmp.geno.gds"
> # first 3 samples only
> scan.annotation <- illumina_scan_annot[1:3, c("scanID", "genoRunID", "file")]
```

```
> names(scan.annotation)[2] <- "scanName"
> snp.annotation <- illumina_snp_annot[,c("snpID", "rsID", "chromosome", "position")]
> # indicate which column of SNP annotation is referenced in data files
> names(snp.annotation)[2] <-  "snpName"
> col.nums <- as.integer(c(1,2,12,13))
> names(col.nums) <- c("snp", "sample", "a1", "a2")
> diag.geno.file <- "diag.geno.RData"
> diag.geno <- createDataFile(path = path,
+   filename = geno.file,
+   file.type = "gds",
+   variables = "genotype",
+   snp.annotation = snp.annotation,
+   scan.annotation = scan.annotation,
+   sep.type = ",",
+   skip.num = 11,
+   col.total = 21,
+   col.nums = col.nums,
+   scan.name.in.file = 1,
+   diagnostics.filename = diag.geno.file,
+   verbose = FALSE)
> # Look at the values included in the "diag.geno" object which holds
> #   all output from the function call
> names(diag.geno)

[1] "read.file"    "row.num"       "samples"       "sample.match" "missg"
[6] "snp.chk"      "chk"

> # `read.file' is a vector indicating whether (1) or not (0) each file
> #   specified in the `files' argument was read successfully
> table(diag.geno$read.file)

1
3

> # `row.num' is a vector of the number of rows read from each file
> table(diag.geno$row.num)

3300
   3

> # `sample.match' is a vector indicating whether (1) or not (0)
> #   the sample name inside the raw text file matches that in the
> #   sample annotation data.frame
> table(diag.geno$sample.match)

1
3
```

```
> # `snp.chk' is a vector indicating whether (1) or not (0)
> #    the raw text file has the expected set of SNP names
> table(diag.geno$snp.chk)

1
3


> # `chk' is a vector indicating whether (1) or not (0) all previous
> #    checks were successful and the data were written to the data file
> table(diag.geno$chk)

1
3
```

Run the function `checkGenotypeFile` to check that the GDS file contains the same data as the raw data files.

```
> check.geno.file <- "check.geno.RData"
> check.geno <- checkGenotypeFile(path = path,
+     filename = geno.file,
+     file.type = "gds",
+     snp.annotation = snp.annotation,
+     scan.annotation = scan.annotation,
+     sep.type = ",",
+     skip.num = 11,
+     col.total = 21,
+     col.nums = col.nums,
+     scan.name.in.file = 1,
+     check.scan.index = 1:3,
+     n.scans.loaded = 3,
+     diagnostics.filename = check.geno.file,
+     verbose = FALSE)
> # Look at the values included in the "check.geno" object which holds
> #    all output from the function call
> names(check.geno)

[1] "read.file"     "row.num"       "sample.names" "sample.match" "missg"
[6] "snp.chk"       "chk"           "geno.chk"

> # 'geno.chk' is a vector indicating whether (1) or not (0) the genotypes
> #    match the text file
> table(check.geno$geno.chk)

1
3
```

Reading the Genotype file

The GdsGenotypeReader class provides a convenient interface for retrieving data from a genotype GDS file. Some of the same "get" methods that applied to SNP and scan annotation data objects can be used for GdsGenotypeReader objects.

```
> (gds <- GdsGenotypeReader(geno.file))

File: /tmp/Rtmp3T8PWN/Rbuild20974b120b55/GWASTools/vignettes/tmp.geno.gds (27.0K)
+    [  ]
|--+ sample.id   { Int32 3 LZMA_ra(683.3%), 89B }
|--+ snp.id    { Int32 3300 LZMA_ra(25.9%), 3.3K }
|--+ snp.chromosome    { UInt8 3300 LZMA_ra(3.45%), 121B }
|--+ snp.position    { Int32 3300 LZMA_ra(68.4%), 8.8K }
|--+ snp.rs.id   { Str8 3300 LZMA_ra(33.6%), 11.0K }
\--+ genotype    { Bit2 3300x3, 2.4K } *

> nscan(gds)

[1] 3

> nsnp(gds)

[1] 3300

> head(getScanID(gds))

[1] 280 281 282

> head(getSnpID(gds))

[1] 999447 999465 999493 999512 999561 999567

> head(getChromosome(gds))

[1] 21 21 21 21 21 21

> head(getPosition(gds))

[1] 13733610 13852569 14038583 14136579 14396024 14404476

> # genotypes for the first 3 samples and  the first 5 SNPs
> getGenotype(gds, snp=c(1,5), scan=c(1,3))

     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]    0    0    0
[3,]    0    0    0
[4,]    1    1    1
[5,]    0    0    0

> close(gds)
```

11

Intensity Files

The intensity files store quality scores and allelic intensity data for each SNP. The normalized X and Y intensities as well as the confidence scores are written to the file for all samples, for all SNPs. (To keep file sizes manageable, a separate file will store the B Allele Frequency and Log R Ratio data.)

In addition to the sample and SNP identifiers, chromosome, and position, the intensity and quality data are written to the intensity file in a two dimensional format, with SNPs corresponding to rows and samples corresponding to columns. To write the intensity data, the raw data files are opened and the intensities and quality score are read. Like with the genotype data, if all sample and probe identifiers match between the data files and the annotation files, the data are populated in the file and diagnostics are written.

Creating the Intensity file

We call `createDataFile` again, this time specifying quality, X, and Y in the `variables` argument. A set of diagnostic values are written and stored in `diag.qxy`.

```
> qxy.file <- "tmp.qxy.gds"
> col.nums <- as.integer(c(1,2,5,16,17))
> names(col.nums) <- c("snp", "sample", "quality", "X", "Y")
> diag.qxy.file <- "diag.qxy.RData"
> diag.qxy <- createDataFile(path = path,
+    filename = qxy.file,
+    file.type = "gds",
+    variables = c("quality","X","Y"),
+    snp.annotation = snp.annotation,
+    scan.annotation = scan.annotation,
+    sep.type = ",",
+    skip.num = 11,
+    col.total = 21,
+    col.nums = col.nums,
+    scan.name.in.file = 1,
+    diagnostics.filename = diag.qxy.file,
+    verbose = FALSE)
```

Run the function `checkIntensityFile` to check that the GDS file contains the same data as the raw data files.

```
> check.qxy.file <- "check.qxy.RData"
> check.qxy <- checkIntensityFile(path = path,
+    filename = qxy.file,
+    file.type = "gds",
+    snp.annotation = snp.annotation,
+    scan.annotation = scan.annotation,
+    sep.type = ",",
+    skip.num = 11,
```

```
+     col.total = 21,
+     col.nums = col.nums,
+     scan.name.in.file = 1,
+     check.scan.index = 1:3,
+     n.scans.loaded = 3,
+     diagnostics.filename = check.qxy.file,
+     verbose = FALSE)
```

Reading the Intensity file

The GdsIntensityReader class provides a convenient interface for retrieving data from an intensity
GDS file. Its methods are similar to GdsGenotypeReader.

```
> (gds <- GdsIntensityReader(qxy.file))

File: /tmp/Rtmp3T8PWN/Rbuild20974b120b55/GWASTools/vignettes/tmp.qxy.gds (66.9K)
+    [  ]
|--+ sample.id   { Int32 3 LZMA_ra(683.3%), 89B }
|--+ snp.id    { Int32 3300 LZMA_ra(25.9%), 3.3K }
|--+ snp.chromosome   { UInt8 3300 LZMA_ra(3.45%), 121B }
|--+ snp.position   { Int32 3300 LZMA_ra(68.4%), 8.8K }
|--+ snp.rs.id    { Str8 3300 LZMA_ra(33.6%), 11.0K }
|--+ quality    { Float32 3300x3 LZMA_ra(20.4%), 7.9K }
|--+ X    { Float32 3300x3 LZMA_ra(44.6%), 17.2K }
\--+ Y    { Float32 3300x3 LZMA_ra(43.4%), 16.8K }

> nscan(gds)

[1] 3

> nsnp(gds)

[1] 3300

> head(getScanID(gds))

[1] 280 281 282

> head(getSnpID(gds))

[1] 999447 999465 999493 999512 999561 999567

> head(getChromosome(gds))

[1] 21 21 21 21 21 21

> head(getPosition(gds))
```

```
[1] 13733610 13852569 14038583 14136579 14396024 14404476

> # quality score for the first 3 samples and  the first 5 SNPs
> getQuality(gds, snp=c(1,5), scan=c(1,3))

        [,1]   [,2]   [,3]
[1,] 0.0000 0.0000 0.0000
[2,] 0.7242 0.7242 0.7242
[3,] 0.2587 0.2587 0.2587
[4,] 0.9206 0.9206 0.9206
[5,] 0.9389 0.9389 0.9389

> # X intensity for the first 3 samples and  the first 5 SNPs
> getX(gds, snp=c(1,5), scan=c(1,3))

       [,1]  [,2]  [,3]
[1,] 1.668 1.581 1.462
[2,] 0.010 0.011 0.008
[3,] 0.941 0.965 0.796
[4,] 0.390 0.380 0.329
[5,] 0.010 0.010 0.008

> # Y intensity for the first 3 samples and  the first 5 SNPs
> getY(gds, snp=c(1,5), scan=c(1,3))

       [,1]  [,2]  [,3]
[1,] 0.027 0.034 0.034
[2,] 0.404 0.482 0.429
[3,] 1.088 1.149 0.934
[4,] 0.480 0.486 0.485
[5,] 0.996 0.919 0.984

> close(gds)
```

B Allele Frequency and Log R Ratio Files

The B Allele Frequency (BAF) and Log R Ratio (LRR) file stores these values for every sample by SNP. For Illumina data, these values are calculated by the BeadStudio software and may be provided by the genotyping center. For a thorough explanation and presentation of an application of these values, please refer to Peiffer, Daniel A., et al. (2006).[2]

For a given sample and SNP, R and θ are calculated using the X and Y intensities, where

$$R = X + Y \tag{1}$$

$$\theta = \frac{2\arctan(Y/X)}{\pi}$$

---

[2]Peiffer, Daniel A., et al. High-resolution genomic profiling of chromosomal aberrations using Infinium whole-genome genotyping. *Genome Research* **16**, 1136-1148 (September 2006).

θ corresponds to the polar coordinate angle and R is the sum of the normalized X and Y intensities (not, as one might assume, the magnitude of the polar coordinate vector).

The LRR is given below. The expected value of R is derived from a plot of θ versus R for a given SNP. It is the predicted value of R derived from a line connecting the centers of the two nearest genotype clusters.

$$\text{LRR} = \log\left(\frac{R_{\text{observed values}}}{R_{\text{expected values}}}\right) \tag{2}$$

Variation in the LRR across a single chromosome indicates possible duplication or deletion, and is an indication of overall sample quality.

The BAF is the frequency of the B allele in the population of cells from which the DNA is extracted. Each sample and SNP combination has a BAF value. Note the BAF values vary for a subject with each DNA extraction and tissue used. After all SNPs have been read and all samples have been clustered for a probe, the mean θ "cluster" value is calculated for each probe, for each of the three genotype clusters, resulting in θ_{AA}, θ_{AB} and θ_{BB} for every probe. Then the θ value for each sample, call it θ_n, is compared to θ_{AA}, θ_{AB} and θ_{BB}. The BAF is calculated

$$\text{BAF} = \begin{cases} 0 & \text{if } \theta_n < \theta_{AA} \\[2ex] \dfrac{(1/2)(\theta_n - \theta_{AA})}{\theta_{AB} - \theta_{AA}} & \text{if } \theta_{AA} \leq \theta_n < \theta_{AB} \\[2ex] \dfrac{1}{2} + \dfrac{(1/2)(\theta_n - \theta_{AB})}{\theta_{BB} - \theta_{AB}} & \text{if } \theta_{AB} \leq \theta_n < \theta_{BB} \\[2ex] 1 & \text{if } \theta_n \geq \theta_{BB} \end{cases}$$

A θ_n value of 0 or 1 corresponds to a homozygote genotype for sample n at that particular probe, and a θ_n value of 1/2 indicates a heterozygote genotype. Thus, $BAF \in [0, 1]$ for each probe. Across a chromosome, three bands are expected, one hovering around 0, one around 1 and one around 0.5, and any deviation from this is considered aberrant.

We use the BAF and LRR values to detect mixed samples or samples of low quality, as well as chromosomal duplications and deletions. Samples that have a significantly large (partial or full chromosome) aberration for a particular chromosome as detected from the BAF values are recommended to be filtered out, for the genotype data are not reliable in these situations. Because of these applications, the BAF and LRR values are a salient part of the data cleaning steps.

Creating the BAF and LRR file

We call `createDataFile` again, this time specifying BAlleleFreq and LogRRatio in the `variables` argument.

```
> bl.file <- "tmp.bl.gds"
> col.nums <- as.integer(c(1,2,20,21))
> names(col.nums) <- c("snp", "sample", "BAlleleFreq", "LogRRatio")
> diag.bl.file <- "diag.bl.RData"
> diag.bl <- createDataFile(path = path,
```

```
+    filename = bl.file,
+    file.type = "gds",
+    variables = c("BAlleleFreq","LogRRatio"),
+    snp.annotation = snp.annotation,
+    scan.annotation = scan.annotation,
+    sep.type = ",",
+    skip.num = 11,
+    col.total = 21,
+    col.nums = col.nums,
+    scan.name.in.file = 1,
+    diagnostics.filename = diag.bl.file,
+    verbose = FALSE)
```

Reading the BAF and LRR file

We also use the `GdsIntensityReader` class for BAF/LRR data.

```
> (gds <- GdsIntensityReader(bl.file))

File: /tmp/Rtmp3T8PWN/Rbuild20974b120b55/GWASTools/vignettes/tmp.bl.gds (63.6K)
+    [ ]
|--+ sample.id   { Int32 3 LZMA_ra(683.3%), 89B }
|--+ snp.id    { Int32 3300 LZMA_ra(25.9%), 3.3K }
|--+ snp.chromosome    { UInt8 3300 LZMA_ra(3.45%), 121B }
|--+ snp.position    { Int32 3300 LZMA_ra(68.4%), 8.8K }
|--+ snp.rs.id    { Str8 3300 LZMA_ra(33.6%), 11.0K }
|--+ BAlleleFreq    { Float32 3300x3 LZMA_ra(38.7%), 15.0K }
\--+ LogRRatio    { Float32 3300x3 LZMA_ra(61.7%), 23.9K }

> getBAlleleFreq(gds, snp=c(1,5), scan=c(1,3))

        [,1]    [,2]    [,3]
[1,]  0.0000  0.0000  0.0005
[2,]  1.0000  1.0000  1.0000
[3,]  0.9898  1.0000  0.9987
[4,]  0.4773  0.4873  0.5363
[5,]  1.0000  1.0000  1.0000

> getLogRRatio(gds, snp=c(1,5), scan=c(1,3))

         [,1]     [,2]     [,3]
[1,]   0.0840   0.0140  -0.0962
[2,]  -0.2727  -0.0180  -0.1944
[3,]   0.1124   0.1779  -0.1147
[4,]   0.0104   0.0020  -0.0858
[5,]   0.0409  -0.0734   0.0224

> close(gds)
```

2.5   Combining data files with SNP and Scan annotation

The `GenotypeData` and `IntensityData` objects combine SNP and scan annotation with GDS (or NetCDF) files, ensuring that scanID, snpID, chromosome, and position are consistent. The constructor for a `GenotypeData` object takes a `GdsGenotypeReader` object as its first argument. Either or both of the `scanAnnot` and `snpAnnot` slots may be empty (`NULL`), but if annotation objects are provided to the constructor, the relevant columns will be checked against the data file during object creation.

```
> genofile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
> gds <- GdsGenotypeReader(genofile)
> # only GDS file
> genoData <- GenotypeData(gds)
> # with scan annotation
> genoData <- GenotypeData(gds, scanAnnot=scanAnnot)
> # with scan and SNP annotation
> genoData <- GenotypeData(gds, snpAnnot=snpAnnot, scanAnnot=scanAnnot)
> genoData

An object of class GenotypeData
 | data:
File: /home/biocbuild/bbs-3.7-bioc/R/library/GWASdata/extdata/illumina_geno.gds (94.0K)
+    [  ]
|--+ sample.id   { Int32 77 ZIP(42.2%), 130B }
|--+ snp.id    { Int32 3300 ZIP(38.5%), 5.0K }
|--+ snp.chromosome    { UInt8 3300 ZIP(1.15%), 38B } *
|--+ snp.position    { Int32 3300 ZIP(91.0%), 11.7K }
|--+ snp.rs.id    { VStr8 3300 ZIP(38.4%), 12.5K }
|--+ snp.allele    { VStr8 3300 ZIP(9.49%), 1.2K }
\--+ genotype    { Bit2 3300x77, 62.0K } *
 | SNP Annotation:
An object of class 'SnpAnnotationDataFrame'
  snps: 1 2 ... 3300 (3300 total)
  varLabels: snpID chromosome ... rBB (14 total)
  varMetadata: labelDescription
 | Scan Annotation:
An object of class 'ScanAnnotationDataFrame'
  scans: 1 2 ... 77 (77 total)
  varLabels: scanID subjectID ... file (13 total)
  varMetadata: labelDescription
```

GenotypeData objects have methods in common with `GdsGenotypeReader`, `SnpAnnotationDataFrame`, and `ScanAnnotationDataFrame`, along with methods to access variables in the annotation slots.

```
> nsnp(genoData)
```

```
[1] 3300

> nscan(genoData)

[1] 77

> # scan annotation
> range(getScanID(genoData))

[1] 280 356

> hasSex(genoData)

[1] TRUE

> table(getSex(genoData))

 F  M
33 44

> hasScanVariable(genoData, "subjectID")

[1] TRUE

> head(getScanVariable(genoData, "subjectID"))

[1] 200191449 200191449 200030290 200030290 200099417 200099417

> getScanVariableNames(genoData)

 [1] "scanID"    "subjectID" "family"    "father"    "mother"    "CoriellID"
 [7] "race"      "sex"       "status"    "genoRunID" "plate"     "batch"
[13] "file"

> # snp annotation
> range(getSnpID(genoData))

[1]  999447 1072820

> table(getChromosome(genoData, char=TRUE))

  21   22    M    X   XY    Y
1000 1000  100 1000  100  100

> head(getPosition(genoData))

[1] 13733610 13852569 14038583 14136579 14396024 14404476

> hasSnpVariable(genoData, "rsID")
```

18

```
[1] TRUE

> head(getSnpVariable(genoData, "rsID"))

[1] "rs3132407" "rs2775671" "rs2775018" "rs3115511" "rs2822404" "rs1556276"

> getSnpVariableNames(genoData)

 [1] "snpID"          "chromosome"     "position"       "rsID"
 [5] "alleleA"        "alleleB"        "BeadSetID"      "IntensityOnly"
 [9] "tAA"            "tAB"            "tBB"            "rAA"
[13] "rAB"            "rBB"

> # genotypes
> getGenotype(genoData, snp=c(1,5), scan=c(1,5))

     [,1] [,2] [,3] [,4] [,5]
[1,]   NA   NA   NA   NA   NA
[2,]    0    0    0    0    0
[3,]    0    0    0    0    1
[4,]    1    1    1    1    0
[5,]    0    0    0    0    0

> close(genoData)
```

IntensityData objects behave in the same way as GenotypeData objects, but take a GdsIntensityReader object as the first argument.

```
> # quality score, X and X intensity
> qxyfile <- system.file("extdata", "illumina_qxy.gds", package="GWASdata")
> gds <- GdsIntensityReader(qxyfile)
> qxyData <- IntensityData(gds, snpAnnot=snpAnnot, scanAnnot=scanAnnot)
> qxyData

An object of class IntensityData
 | data:
File: /home/biocbuild/bbs-3.7-bioc/R/library/GWASdata/extdata/illumina_qxy.gds (1.0M)
+    [  ]
|--+ sample.id   { Int32 77 ZIP(42.2%), 130B }
|--+ snp.id    { Int32 3300 ZIP(38.5%), 5.0K }
|--+ snp.chromosome    { UInt8 3300 ZIP(1.15%), 38B } *
|--+ snp.position    { Int32 3300 ZIP(91.0%), 11.7K }
|--+ snp.rs.id    { VStr8 3300 ZIP(38.4%), 12.5K }
|--+ quality    { Float32 3300x77 ZIP(4.93%), 48.9K }
|--+ X    { Float32 3300x77 ZIP(50.3%), 498.8K }
\--+ Y    { Float32 3300x77 ZIP(49.4%), 490.5K }
 | SNP Annotation:
```

```
An object of class 'SnpAnnotationDataFrame'
  snps: 1 2 ... 3300 (3300 total)
  varLabels: snpID chromosome ... rBB (14 total)
  varMetadata: labelDescription
 | Scan Annotation:
An object of class 'ScanAnnotationDataFrame'
  scans: 1 2 ... 77 (77 total)
  varLabels: scanID subjectID ... file (13 total)
  varMetadata: labelDescription

> getQuality(qxyData, snp=c(1,5), scan=c(1,5))

        [,1]   [,2]   [,3]   [,4]   [,5]
[1,] 0.0000 0.0000 0.0000 0.0000 0.0000
[2,] 0.7242 0.7242 0.7242 0.7242 0.7242
[3,] 0.2587 0.2587 0.2587 0.2587 0.2587
[4,] 0.9206 0.9206 0.9206 0.9206 0.9206
[5,] 0.9389 0.9389 0.9389 0.9389 0.9389

> getX(qxyData, snp=c(1,5), scan=c(1,5))

       [,1]  [,2]  [,3]  [,4]  [,5]
[1,] 1.668 1.581 1.462 1.456 1.512
[2,] 0.010 0.011 0.008 0.008 0.000
[3,] 0.941 0.965 0.796 0.942 1.518
[4,] 0.390 0.380 0.329 0.411 0.023
[5,] 0.010 0.010 0.008 0.016 0.013

> getY(qxyData, snp=c(1,5), scan=c(1,5))

       [,1]  [,2]  [,3]  [,4]  [,5]
[1,] 0.027 0.034 0.034 0.024 0.027
[2,] 0.404 0.482 0.429 0.357 0.437
[3,] 1.088 1.149 0.934 1.045 0.696
[4,] 0.480 0.486 0.485 0.462 0.895
[5,] 0.996 0.919 0.984 0.978 1.006

> close(qxyData)
> # BAF/LRR
> blfile <- system.file("extdata", "illumina_bl.gds", package="GWASdata")
> gds <- GdsIntensityReader(blfile)
> blData <- IntensityData(gds, snpAnnot=snpAnnot, scanAnnot=scanAnnot)
> blData

An object of class IntensityData
 | data:
File: /home/biocbuild/bbs-3.7-bioc/R/library/GWASdata/extdata/illumina_bl.gds (1.1M)
```

```
+    [ ]
|--+ sample.id   { Int32 77 ZIP(42.2%), 130B }
|--+ snp.id   { Int32 3300 ZIP(38.5%), 5.0K }
|--+ snp.chromosome   { UInt8 3300 ZIP(1.15%), 38B } *
|--+ snp.position   { Int32 3300 ZIP(91.0%), 11.7K }
|--+ snp.rs.id   { VStr8 3300 ZIP(38.4%), 12.5K }
|--+ BAlleleFreq   { Float32 3300x77 ZIP(40.9%), 405.8K }
\--+ LogRRatio   { Float32 3300x77 ZIP(68.2%), 676.6K }
 | SNP Annotation:
An object of class 'SnpAnnotationDataFrame'
  snps: 1 2 ... 3300 (3300 total)
  varLabels: snpID chromosome ... rBB (14 total)
  varMetadata: labelDescription
 | Scan Annotation:
An object of class 'ScanAnnotationDataFrame'
  scans: 1 2 ... 77 (77 total)
  varLabels: scanID subjectID ... file (13 total)
  varMetadata: labelDescription

> getBAlleleFreq(blData, snp=c(1,5), scan=c(1,5))

       [,1]   [,2]   [,3]   [,4]   [,5]
[1,] 0.0000 0.0000 0.0005 0.0000 0.0000
[2,] 1.0000 1.0000 1.0000 1.0000 1.0000
[3,] 0.9898 1.0000 0.9987 0.9664 0.4893
[4,] 0.4773 0.4873 0.5363 0.4531 0.9978
[5,] 1.0000 1.0000 1.0000 1.0000 1.0000

> getLogRRatio(blData, snp=c(1,5), scan=c(1,5))

        [,1]    [,2]    [,3]    [,4]    [,5]
[1,]  0.0840  0.0140 -0.0962 -0.1122 -0.0550
[2,] -0.2727 -0.0180 -0.1944 -0.4495 -0.2003
[3,]  0.1124  0.1779 -0.1147  0.0729  0.0628
[4,]  0.0104  0.0020 -0.0858  0.0223  0.1904
[5,]  0.0409 -0.0734  0.0224  0.0226  0.0591

> close(blData)
```

3 Batch Quality Checks

The overall goal of this step is to check the quality of the sample batches. Substantial quality control is done by the genotyping centers prior to releasing the genotype data; however it is our experience that despite the stringent quality controls it is still possible for batches with lower than desired quality to pass the pre-release data quality checks. If a lower quality batch is detected then it may be necessary to re-run the genotyping for that batch. We check the batch quality by comparing the missing call rates between batches and looking for significant allele frequency differences between batches.

3.1 Calculate Missing Call Rate for Samples and SNPs

The first step is to calculate the missing call rates for each SNP and for each sample. A high missing call rate for a sample is often indicative of a poorly performing sample. It has been seen that samples from DNA that has undergone whole-genome amplification (WGA) have a relatively higher missing call rate. Similarly a high missing call rate for a SNP is indicative of a problem SNP. Experience from the GENEVA studies has shown that there seem to be a subset of SNPs from which genotype calls are more difficult to make than others. We calculate the missing call rates in a two step process: first the missing call rates over all samples and SNPs are calculated, then the missing call rates are calculated again, filtering out SNPs and samples that have an initial missing call rate greater than 0.05. The initial SNP missing call rate over all samples is saved in the SNP annotation data file as `missing.n1`. The analogous idea is applied to the samples: `missing.e1` is saved in the sample annotation file and corresponds to the missing call rate per sample over all SNPs, excluding those SNPs with all calls missing. The `missing.n2` is calculated as the call rate per SNP over all samples whose `missing.e1` is less than 0.05. Again, similarly for the samples, `missing.e2` is calculated for each sample over all SNPs with `missing.n2` values less than 0.05. It is important to remember that the Y chromosome values should be calculated for males only, since we expect females to have no genotype values for the Y chromosome.

Calculate `missing.n1`

This step calculates and examines `missing.n1`, the missing call rate per SNP over all samples by calling the function `missingGenotypeBySnpSex`. This function takes a `GenotypeData` object as an argument, and requires that the scan annotation of this object contains a "sex" column. There is also an option to send a vector of SNPs to exclude from the calculation, which is what we will use later to find `missing.n2`. For now, we will use all SNPs for each sample, being sure to calculate by sex. The function returns a list, with one element that holds the missing counts for each SNP, one element that holds the sex counts, and one element that holds the fraction of missing calls.

```
> # open the GDS file and create a GenotypeData object
> gdsfile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
> gds <- GdsGenotypeReader(gdsfile)
> # sex is required for this function, so we need the scan annotation
> genoData <-  GenotypeData(gds, scanAnnot=scanAnnot)
> # Calculate the number of missing calls for each snp over all samples
> #     for each sex separately
> miss <- missingGenotypeBySnpSex(genoData)
```

```
> # Examine the results
> names(miss)

[1] "missing.counts"   "scans.per.sex"    "missing.fraction"

> head(miss$missing.counts)

        M  F
999447 44 33
999465  0  1
999493  3  2
999512  0  0
999561  0  0
999567  0  0

> miss$scans.per.sex

 M  F
44 33

> head(miss$missing.fraction)

    999447     999465     999493     999512     999561     999567
1.00000000 0.01298701 0.06493506 0.00000000 0.00000000 0.00000000
```

The Y chromosome should be missing for all females, but an occasional probe on the Y chromosome is called in a female. `missingGenotypeBySnpSex` excludes females when calculating the missing rate for Y chromosome SNPs. Note this may need to be changed later if there are some sex mis-annotations because the Y chromosome SNP missing call rates may change. We add the missing call rates to the SNP annotation.

```
> # Make sure ordering matches snp annotation
> allequal(snpAnnot$snpID, as.numeric(names(miss$missing.fraction)))

[1] TRUE

> snpAnnot$missing.n1 <- miss$missing.fraction
> varMetadata(snpAnnot)["missing.n1", "labelDescription"] <- paste(
+   "fraction of genotype calls missing over all samples",
+   "except that females are excluded for Y chr SNPs")
> summary(snpAnnot$missing.n1)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00000 0.00000 0.00000 0.04899 0.00000 1.00000
```
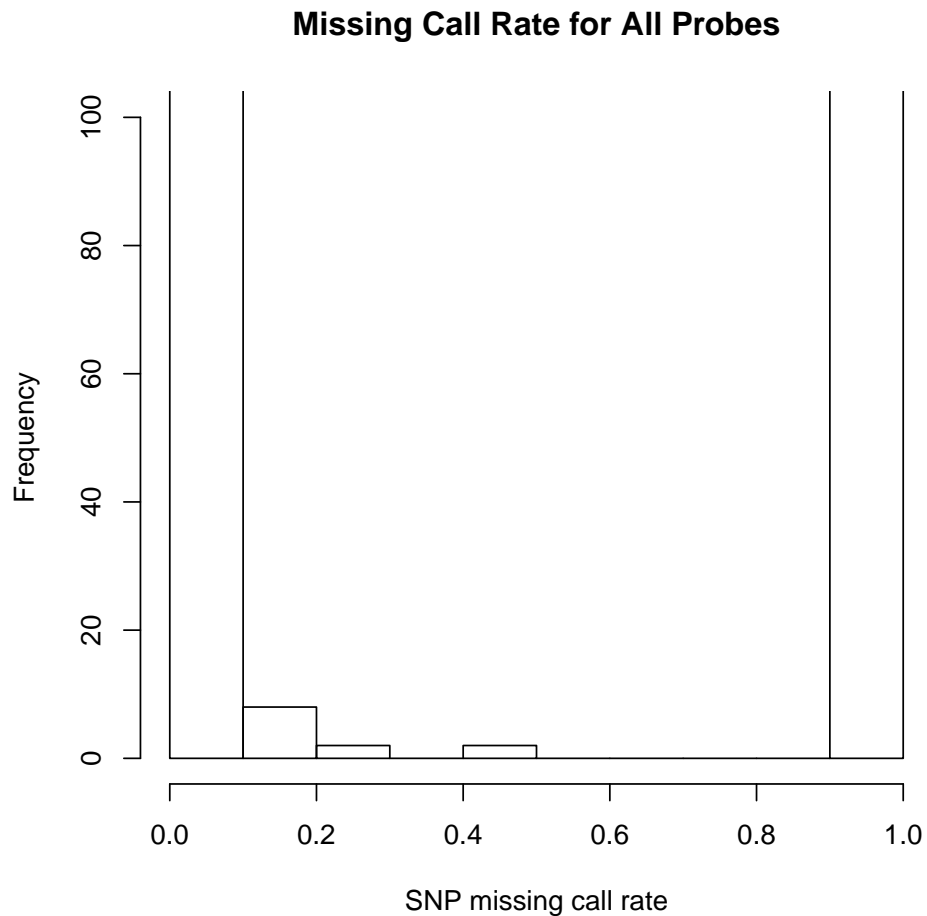
We plot the missing call rates so we can easily identify any outliers. We also find the number of SNPs with 100% missing, and the fraction of SNPs with missing call rate less than 0.05 for each chromosome type.

```
> hist(snpAnnot$missing.n1, ylim=c(0,100),
+      xlab="SNP missing call rate",
+      main="Missing Call Rate for All Probes")
```

Missing Call Rate for All Probes


```
> # Find the number of SNPs with every call missing
> length(snpAnnot$missing.n1[snpAnnot$missing.n1 == 1])

[1] 151

> # Fraction of autosomal SNPs with missing call rate < 0.05
> x <- snpAnnot$missing.n1[snpAnnot$chromosome < 23]
> length(x[x < 0.05]) / length(x)

[1] 0.9805

> # Fraction of X chromosome SNPs with missing call rate < 0.05
> x <- snpAnnot$missing.n1[snpAnnot$chromosome == 23]
> length(x[x < 0.05]) / length(x)
```

```
[1] 0.95

> # Fraction of Y chromosome SNPs with missing call rate < 0.05
> x <- snpAnnot$missing.n1[snpAnnot$chromosome == 25]
> length(x[x < 0.05]) / length(x)

[1] 0.38
```

Calculate `missing.e1`

This step calculates `missing.e1`, which is the missing call rate per sample over all SNPs, by chromosome. We read in the new SNP annotation file which holds the `missing.n1` variable. For those SNPs with a `missing.n1` value less than one, we call the `missingGenotypeByScanChrom` function that returns a list with one element holding the missing counts per sample by chromosome, one element holding the number of SNPs per chromosome, and one element holding the fraction of missing calls over all chromosomes.

```
> # Want to exclude all SNP probes with 100% missing call rate
> # Check on how many SNPs to exclude
> sum(snpAnnot$missing.n1 == 1)

[1] 151

> # Create a variable that contains the IDs of these SNPs to exclude
> snpexcl <- snpAnnot$snpID[snpAnnot$missing.n1 == 1]
> length(snpexcl)

[1] 151

> # Calculate the missing call rate per sample
> miss <- missingGenotypeByScanChrom(genoData, snp.exclude=snpexcl)
> names(miss)

[1] "missing.counts"   "snps.per.chr"      "missing.fraction"

> head(miss$missing.counts)

    21 22 X XY  Y  M
280  2  3 0  1 42  0
281  1  3 0  1 42  8
282  6  5 3  0 42  0
283  3  4 3  0 42 25
284  1  2 1  0  1  0
285  0  0 1  0  1  9

> head(miss$snps.per.chr)

 21  22   X  XY   Y   M
999 997 950  61  42 100
```

```
> # Check to make sure that the correct number of SNPs were excluded
> sum(miss$snps.per.chr)
```

```
[1] 3149
```

```
> nrow(snpAnnot) - sum(miss$snps.per.chr)
```

```
snps
 151
```

`missingGenotypeByScanChrom` calculates the missing call rate for each sample over all SNPs. For females, the missing call rate does not include the probes on the Y chromosome. The values for `missing.e1` are added to the sample annotation file.

```
> head(miss$missing.fraction)
```

```
        280         281         282         283         284         285
0.001931123 0.004184100 0.004505954 0.011264886 0.001587806 0.003493172
```

```
> # Check the ordering matches the sample annotation file
> allequal(names(miss$missing.fraction), scanAnnot$scanID)
```

```
[1] TRUE
```

```
> # Add the missing call rates vector to the sample annotation file
> scanAnnot$missing.e1 <- miss$missing.fraction
> varMetadata(scanAnnot)["missing.e1", "labelDescription"] <- paste(
+    "fraction of genotype calls missing over all snps with missing.n1<1",
+    "except that Y chr SNPs are excluded for females")
> summary(scanAnnot$missing.e1)
```

```
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.0000000 0.0009656 0.0022229 0.0032324 0.0041283 0.0247827
```

We will create a histogram of the overall missing call rate per sample in order to identify any samples with a relatively larger missing call rate. It is known that genotype data taken from DNA that has been through whole-genome amplification (WGA) has an overall higher missing call rate; this is something that we would see at this step if any samples are of WGA origin. We also look at the summary of the missing call rate for females and males separately to ensure there are no large sex differences. Finally, we calculate the number of samples with a missing call rate greater than 0.05. In this case, there are no such samples but in other data this may not be the case. If any samples have a high missing rate, we recommend further investigation of what may be causing the missing calls; the samples with a missing call rate greater than 0.05 should be filtered out due to low sample quality.

```
> hist(scanAnnot$missing.e1,
+      xlab="Fraction of missing calls over all probes",
+      main="Histogram of Sample Missing Call Rate for all Samples")
```

Histogram of Sample Missing Call Rate for all Samples
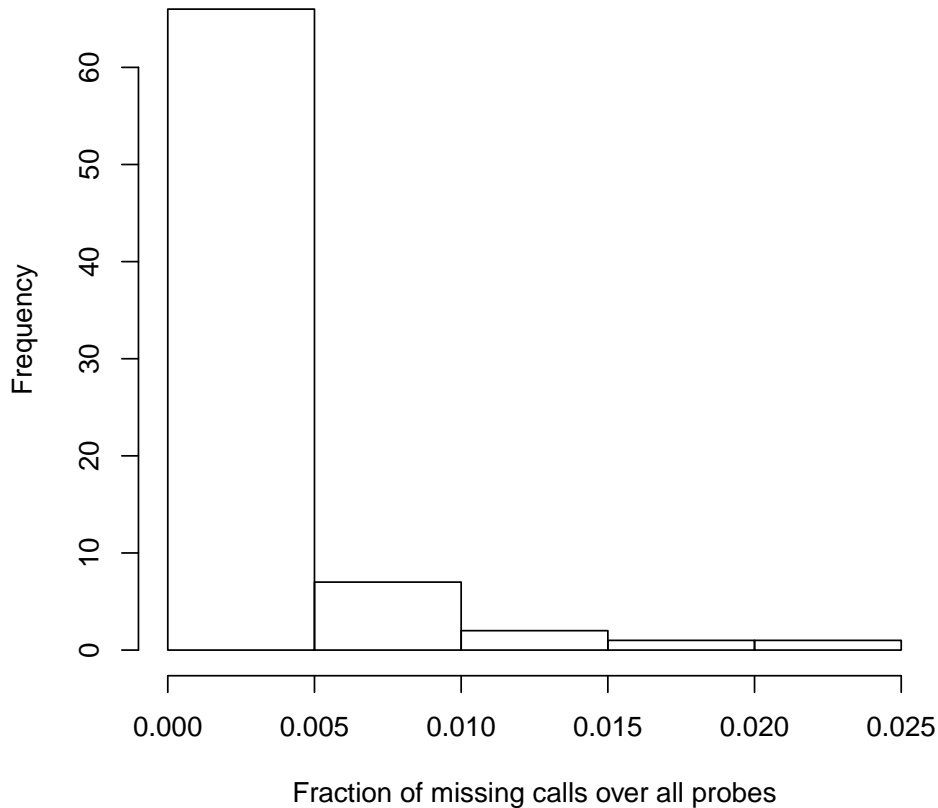

```
> # Look at missing.e1 for males
> summary(scanAnnot$missing.e1[scanAnnot$sex == "M"])

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.000000 0.001270 0.002540 0.003082 0.003890 0.013655

> # Look at missing.e1 for females
> summary(scanAnnot$missing.e1[scanAnnot$sex == "F"])

     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.0003219 0.0009656 0.0016093 0.0034331 0.0041841 0.0247827

> # Number of samples with missing call rate > 5%
> sum(scanAnnot$missing.e1 > 0.05)

[1] 0
```

For some analyses we require the missing call rate for autosomes and the X chromosome to be separated. We calculate these values here and add them to the sample annotation file. Also,

we will create a logical `duplicated` variable. We can identify the duplicated scans in the sample annotation file by identifying the subject ids that occur more than once. Among samples with the same subject id, the one with the lowest `missing.e1` value will have the variable `duplicated` set to FALSE.

```
> auto <- colnames(miss$missing.counts) %in% 1:22
> missa <- rowSums(miss$missing.counts[,auto]) / sum(miss$snps.per.chr[auto])
> summary(missa)

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.000000 0.001002 0.001503 0.002726 0.003507 0.027555

> missx <- miss$missing.counts[,"X"] / miss$snps.per.chr["X"]
> summary(missx)

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.000000 0.000000 0.001053 0.001394 0.001053 0.022105

> # check they match sample annotation file
> allequal(names(missa), scanAnnot$scanID)

[1] TRUE

> allequal(names(missx), scanAnnot$scanID)

[1] TRUE

> # Add these separate sample missing call rates to the sample
> # annotation
> scanAnnot$miss.e1.auto <- missa
> scanAnnot$miss.e1.xchr <- missx
> # Order scanAnnot by missing.e1 so duplicate subjectIDs
> # with a higher missing rate are marked as duplicates
> scanAnnot <- scanAnnot[order(scanAnnot$subjectID, scanAnnot$missing.e1),]
> scanAnnot$duplicated <- duplicated(scanAnnot$subjectID)
> table(scanAnnot$duplicated, useNA="ifany")

FALSE   TRUE
   43     34

> # Put scanAnnot back in scanID order; this is very important!!
> scanAnnot <- scanAnnot[order(scanAnnot$scanID),]
> allequal(scanAnnot$scanID, sort(scanAnnot$scanID))

[1] TRUE

> varMetadata(scanAnnot)["duplicated", "labelDescription"] <-
+   "TRUE for duplicate scan with higher missing.e1"
```

Calculate `missing.n2`

This step calculates `missing.n2`, which is the missing call rate per SNP with `missing.e1` less than 0.05 over all samples. In some cases, there will be samples with missing call rate greater than 0.05. However, because of the high quality of the HapMap data, there are no such samples in this case. We will continue with the steps as if there are samples we must exclude from the `missing.n2` calculation. We call the `missingGenotypeBySnpSex` function just as we did to calculate for `missing.n1`, but this time we include the list of sample numbers to exclude from the calculation (although here that list is empty).

```
> # Find the samples with missing.e1 > .05 and make a vector of
> # scanID to exclude from the calculation
> scan.exclude <- scanAnnot$scanID[scanAnnot$missing.e1 > 0.05]
> # Call missingGenotypeBySnpSex and save the output
> miss <- missingGenotypeBySnpSex(genoData, scan.exclude=scan.exclude)
> snpAnnot$missing.n2 <- miss$missing.fraction
> varMetadata(snpAnnot)["missing.n2", "labelDescription"] <- paste(
+    "fraction of genotype calls missing over all samples with missing.e1<0.05",
+    "except that females are excluded for Y chr SNPs")
> summary(snpAnnot$missing.n2)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00000 0.00000 0.00000 0.04899 0.00000 1.00000
```

Calculate `missing.e2`

This step calculates `missing.e2`, which is the missing call rate per sample over all SNPs with `missing.n2` less than 0.05.

```
> # Create a vector of the SNPs to exclude.
> snpexcl <- snpAnnot$snpID[snpAnnot$missing.n2 >= 0.05]
> length(snpexcl)

[1] 206

> miss <- missingGenotypeByScanChrom(genoData, snp.exclude=snpexcl)
> # Add the missing call rates vector to the sample annotation file
> scanAnnot$missing.e2 <- miss$missing.fraction
> varMetadata(scanAnnot)["missing.e2", "labelDescription"] <- paste(
+    "fraction of genotype calls missing over all snps with missing.n2<0.05",
+    "except that Y chr SNPs are excluded for females")
> summary(scanAnnot$missing.e2)

     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.0000000 0.0003272 0.0006544 0.0018289 0.0019392 0.0242147
```
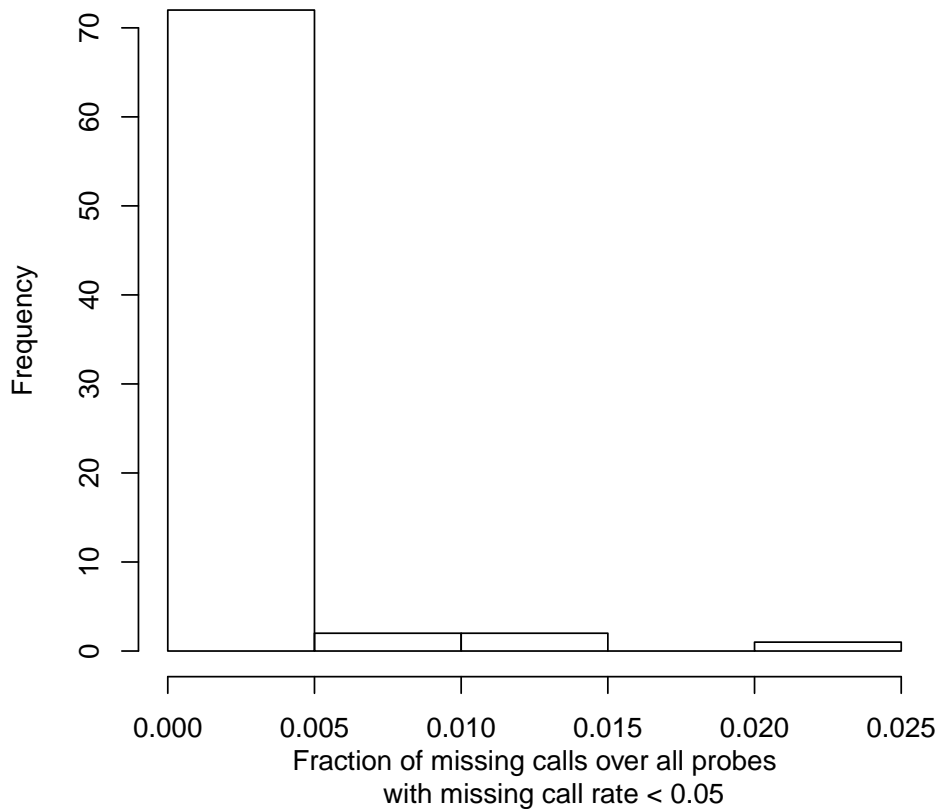
We will create a histogram of the overall missing call rate per sample in order to identify any samples with a relatively larger missing call rate.

```
> hist(scanAnnot$missing.e2, xlab="Fraction of missing calls over all probes
+        with missing call rate < 0.05",
+        main="Histogram of Sample Missing Call Rate for all Samples")
```

Histogram of Sample Missing Call Rate for all Samples

3.2 Calculate Missing Call Rates by Batch

Next, the missing call rate by batch is calculated to check that there are no batches with compara-
tively lower call rates. Usually a "batch" is a plate containing samples that were processed together
through the genotyping chemistry. In this case all samples were run on different plates (as controls
for another dataset), so we use the simulated variable "batch." We calculate the mean missing call
rate for all samples in each of the batches.

```
> varLabels(scanAnnot)

 [1] "scanID"       "subjectID"    "family"       "father"       "mother"
 [6] "CoriellID"    "race"         "sex"          "status"       "genoRunID"
[11] "plate"        "batch"        "file"         "missing.e1"   "miss.e1.auto"
[16] "miss.e1.xchr" "duplicated"   "missing.e2"
```
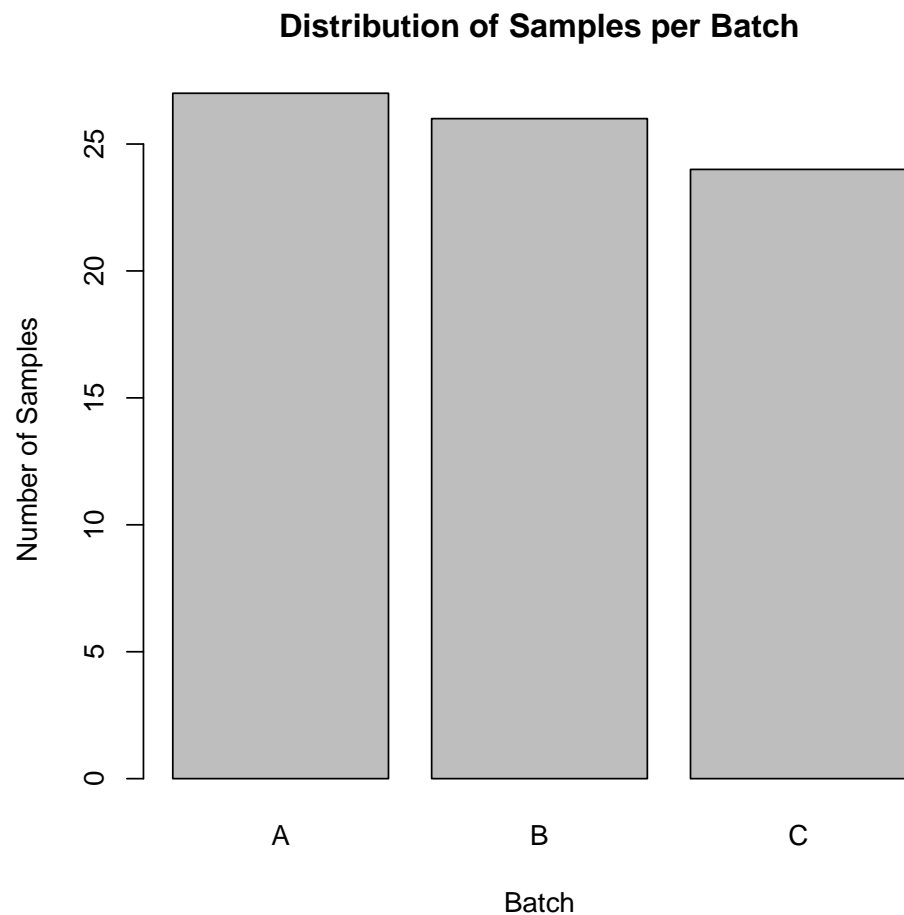
```
> # Check how many batches exist and how many samples are in each batch
> length(unique(scanAnnot$batch))

[1] 3

> table(scanAnnot$batch, useNA="ifany")

 A  B  C
27 26 24

> # Plot the distribution of the number of samples per batch.
> barplot(table(scanAnnot$batch),
+         ylab="Number of Samples", xlab="Batch",
+         main="Distribution of Samples per Batch")
```


Distribution of Samples per Batch

```
> # Examine the mean missing call rate per batch for all SNPs
> batches <- unique(scanAnnot$batch)
> bmiss <- rep(NA,length(batches)); names(bmiss) <- batches
```

```
> bn <- rep(NA,length(batches)); names(bn) <- batches
> for(i in 1:length(batches)) {
+   x <- scanAnnot$missing.e1[is.element(scanAnnot$batch, batches[i])]
+   bmiss[i] <- mean(x)
+   bn[i] <- length(x)
+ }
```

To find the slope of the regression line from the mean missing call rate per batch regressed on the number of samples per batch, we will take the results from ANOVA. Then we can plot the mean missing call rate against the number of samples in the batch with the regression line. For studies with more batches, this test can identify any batch outliers with regard to missing call rate for samples in a given batch. We can do the same analysis using the mean missing call rate for autosomal SNPs, or SNPs on the X chromosome in the exact same way, substituting `missing.e1` with either `miss.e1.auto` or `miss.e1.xchr`. Because the results are nearly identical, we will not show them here.

```
> y <- lm(bmiss ~ bn)
> anova(y)

Analysis of Variance Table

Response: bmiss
          Df     Sum Sq    Mean Sq F value Pr(>F)
bn         1 8.7186e-07 8.7186e-07   1.114 0.4828
Residuals  1 7.8266e-07 7.8266e-07

> plot(bn, bmiss,
+  xlab="Number of samples per batch", ylab="Mean missing call rate",
+  main="Mean Missing Call Rate vs\nSamples per Batch")
> abline(y$coefficients)
```
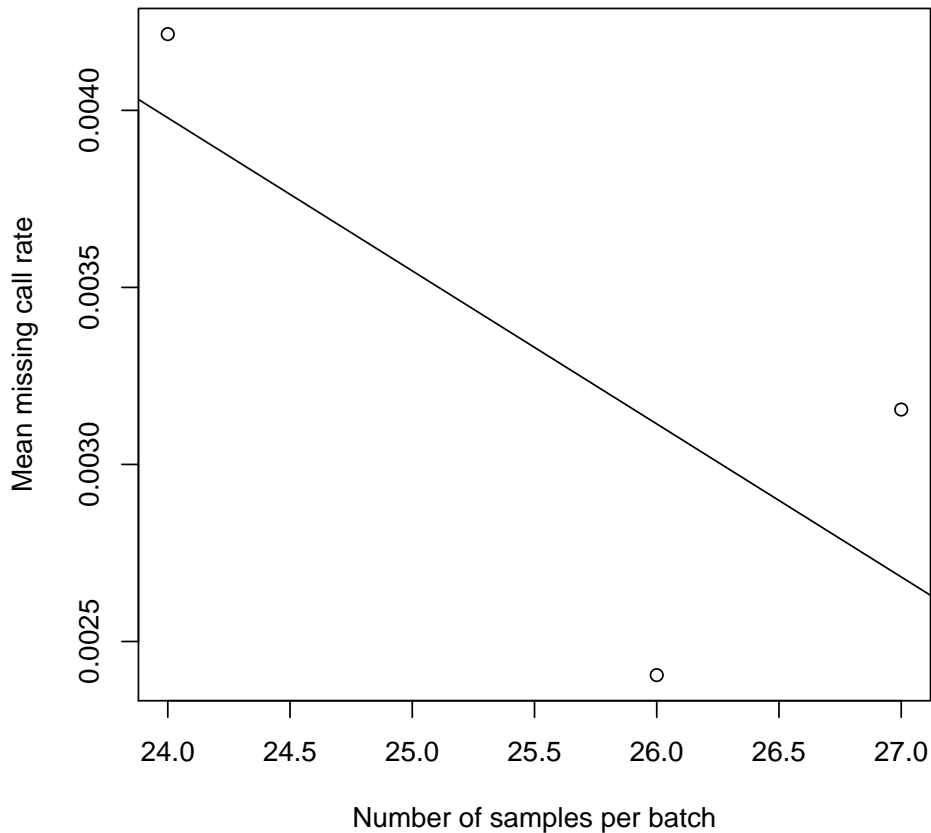
**Mean Missing Call Rate vs
Samples per Batch**

3.3   Chi-Square Test of Allelic Frequency Differences in Batches

In this step, the chi-square test for differences in allelic frequency is performed between each batch individually and a pool of all the other batches in the study. We then look at the mean χ^2 statistic over all SNPs for each batch as a function of the ethnic composition of samples in a batch. We use the `batch` variable in the scan annotation to identify the samples in each batch, so we must include the scan annotation in the `GenotypeData` object. Then we call the function `batchChisqTest` which calculates the χ^2 values from 2×2 tables for each SNP, comparing each batch with the other batches. This function returns the genomic inflation factors for each batch, as well as matrix of χ^2 values for each SNP.

```
> res <- batchChisqTest(genoData, batchVar="batch", return.by.snp=TRUE)
> close(genoData)
> # chi-square values for each SNP
> dim(res$chisq)

[1] 2000     3
```

```
> # genomic inflation factor
> res$lambda

        A         B         C
0.6742432 0.1769204 0.3857391

> # average chi-square test statistic for each of the batches
> res$mean.chisq

        A         B         C
0.8733652 0.3452383 0.5587996
```

Next we test for association between batches and population groups, using a χ^2 contingency test. Then we look at the relationship between the ethnic composition of each batch and the previously calculated χ^2 test of allelic frequency between each batch and a pool of the other batches. The point is to look for batches that differ from others of similar ethnic composition, which might indicate a batch effect due to genotyping artifact. In this experiment, there are only a few batches and wide variations in race among batches, so it is difficult to interpret the results. In larger GWAS experiments, we generally observe a U-shaped curve of allelic frequency test statistic as a function of ethnic composition.

```
> x <- table(scanAnnot$race, useNA="ifany")
> x

CEU YRI
 49  28

> x[1] / sum(x)

      CEU
0.6363636

> x[2] / sum(x)

      YRI
0.3636364

> x <- table(scanAnnot$race, scanAnnot$batch)
> x

       A  B  C
  CEU 15 18 16
  YRI 12  8  8

> # Run an approximate chi-square test to see if there are ethnic effects
> chisq <- chisq.test(x)
> chisq$p.value
```

```
[1] 0.5464046

> # Calculate the fraction of samples in each batch that are CEU
> batches <- unique(scanAnnot$batch)
> eth <- rep(NA,length(batches)); names(eth) <- sort(batches)
> for(i in 1:length(batches)){
+    x <- scanAnnot$race[is.element(scanAnnot$batch, batches[i])]
+    xl <- length(x[x == "CEU"])
+    eth[i] <- xl / length(x)
+ }
> allequal(names(eth), names(res$mean.chisq))

[1] TRUE

> # Plot the average Chi-Square test statistic against the
> #      fraction of samples that are CEU
> plot(eth, res$mean.chisq, xlab="Fraction of CEU Samples per Batch",
+   ylab="Average Chi-square Test Statistic",
+   main="Fraction of CEU Samples per Batch
+   vs Average Chi-square Test Statistic")
> abline(v=mean(eth), lty=2, col="red")
```
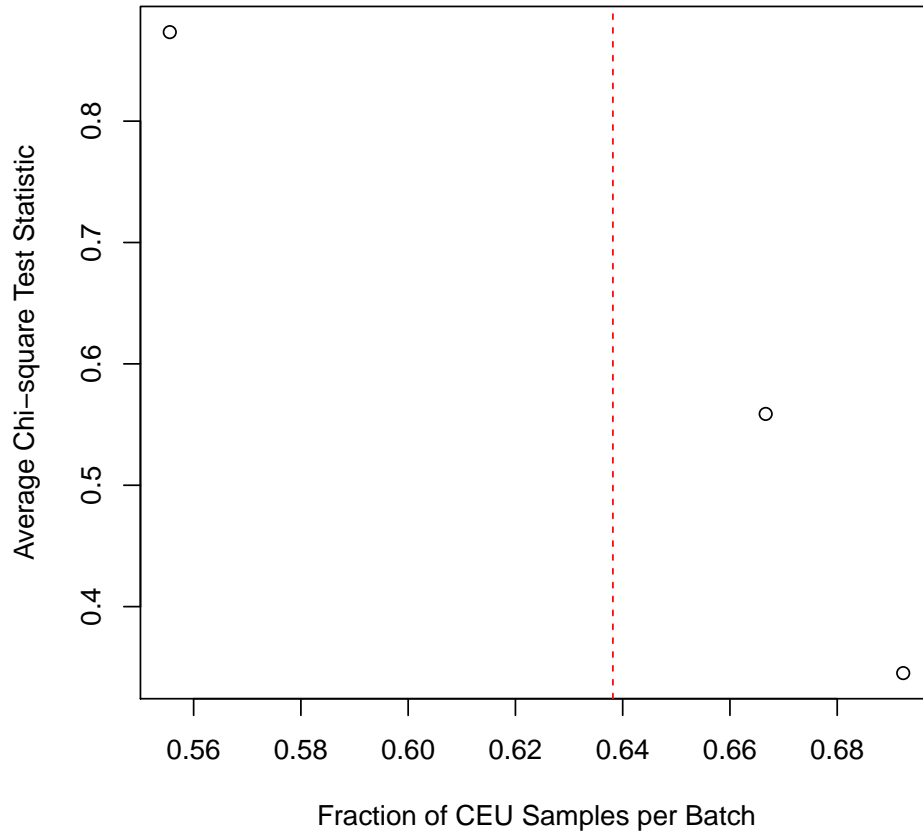
**Fraction of CEU Samples per Batch
vs Average Chi–square Test Statistic**

The χ^2 test is not suitable when the 2×2 tables for each SNP have very small values. For arrays in which many SNPs have very low minor allele frequency, Fisher's exact test is more appropriate. The function `batchFisherTest` can be used in a very similar way to `batchChisqTest`, but the run time is significantly longer, as it iterates over each SNP.

4 Sample Quality Checks

In this step we examine sample quality using three methods. We check for outliers in genotype quality score; we check for anomalous sample-chromosome pairs using BAF variance analysis; lastly, we check sample missingness and heterozygosities.

4.1 Sample genotype quality scores

Genotype calling algorithms report quality scores and classify genotypes with insufficient confidence as missing. This code calculates the mean and median genotype quality score for each sample.

Calculate quality scores by sample. The `qualityScoreByScan` function requires both an `IntensityData` object, to read the quality scores, and a `GenotypeData` object, to determine which scans have missing genotypes and should be omitted from the calculation.

```
> qxyfile <- system.file("extdata", "illumina_qxy.gds", package="GWASdata")
> qualGDS <- GdsIntensityReader(qxyfile)
> qualData <- IntensityData(qualGDS, scanAnnot=scanAnnot)
> genofile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
> genoGDS <- GdsGenotypeReader(genofile)
> genoData <- GenotypeData(genoGDS, scanAnnot=scanAnnot)
> qual.results <- qualityScoreByScan(qualData, genoData)
> close(qualData)
```

We plot the distribution of median quality scores; it is unsurprising that these are all good, given that some quality checking happens at the genotyping centers. Clear outliers in this plot would be cause for concern that the sample(s) in question were of significantly lower quality than the other samples.

```
> hist(qual.results[,"median.quality"], main="Median Genotype Quality Scores
+   of Samples", xlab="Median Quality")
```

Median Genotype Quality Scores
of Samples
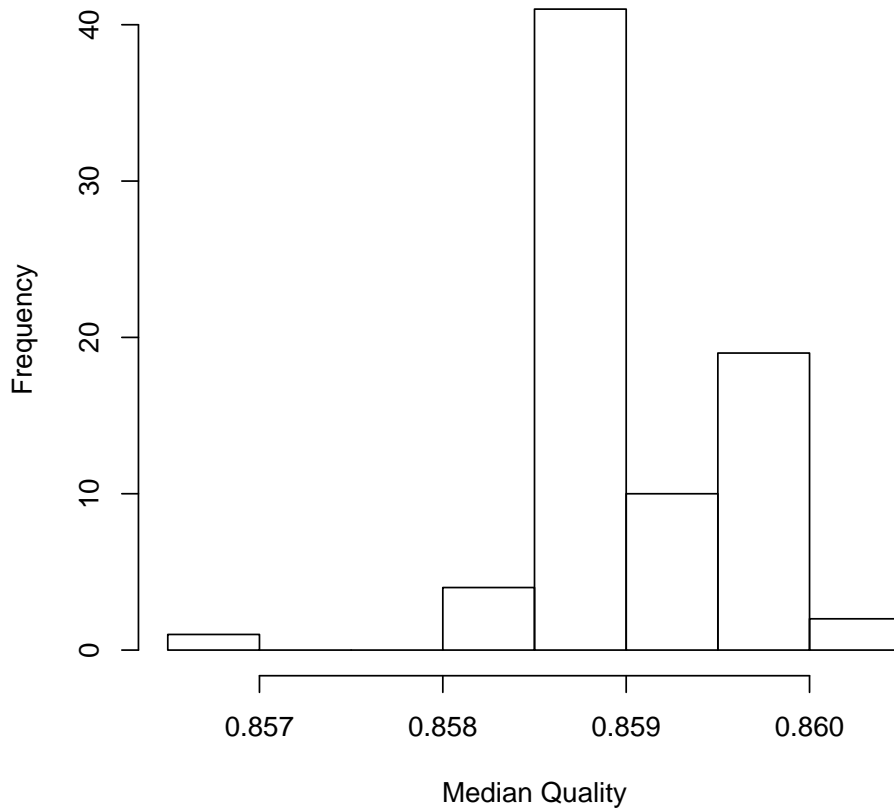
4.2   B Allele Frequency variance analysis

BAF is a standardized version of the polar coordinate angle (Section 2.4). It calculates the frequency of the B allele within a single sample. Under normal circumstances, the true frequency is 0, $\frac{1}{2}$, or 1. In cases of allelic imbalance the true frequencies may vary. For example, in a population of trisomic cells, the true frequencies would be 0, $\frac{1}{3}$, $\frac{2}{3}$, or 1. Here we calculate the variance of BAF (for SNPs called as heterozygotes) within a sliding window along each chromosome for each sample. Each chromosome is divided into 12 sections with equal numbers of SNPs and the variance is calculated in a window of two adjacent sections (one-sixth of the chromosome), which slides along the chromosome in increments of one section. Regions (windows) with very high BAF variance can indicate chromosomal anomalies.

Calculate the sliding window BAF standard deviation

This process identifies chromosome-sample pairs that have windows with very high BAF standard deviation, with "very high" defined as more than 4 standard deviations from the window's mean BAF standard deviation over all samples. The output is a matrix listing all sample-chromosome

pairs with high BAF standard deviations, the number of windows with high SDs in each pair, and the sample's sex. We examine plots of BAF by position for each identified chromosome-sample pair (though only a subset of plots are shown here).

First, run the `meanBAFbyScanChromWindow` function. This requires both an `IntensityData` object with BAF and a `GenotypeData` object. Its output is a list of matrices, with one matrix for each chromosome containing the standard deviation of BAF at each window in each scan.

```
> blfile <- system.file("extdata", "illumina_bl.gds", package="GWASdata")
> blGDS <- GdsIntensityReader(blfile)
> blData <- IntensityData(blGDS, scanAnnot=scanAnnot)
> nbins <- rep(12, 3)
> slidingBAF12 <- sdByScanChromWindow(blData, genoData, nbins=nbins)
> names(slidingBAF12)

[1] "21" "22" "X"

> dim(slidingBAF12[["21"]])

[1] 77 11
```

The function `meanBAFSDbyChromWindow` calculates the mean and standard deviation of the BAF standard deviations in each window in each chromosome over all samples. For the X chromosome, males and females are calculated separately, and we save the results split by sex.

```
> sds.chr <- meanSdByChromWindow(slidingBAF12, scanAnnot$sex)
> sds.chr[["21"]]

           [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
Mean 0.08262367 0.04971894 0.04991385 0.04578610 0.04156682 0.04139052
SD   0.02082800 0.01652451 0.01710329 0.01507464 0.01547533 0.01498545
           [,7]       [,8]       [,9]      [,10]      [,11]
Mean 0.04120665 0.03912475 0.03953937 0.04189777 0.04518619
SD   0.01663141 0.01243865 0.01064164 0.01197254 0.01572490

> sds.chr[["X"]]

                   [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
Female Mean 0.14716196 0.09812081 0.13814218 0.16212499 0.12026203 0.16164527
Male Mean   0.45376698 0.40033938 0.42632934 0.49092806 0.51290672 0.46628564
Female SD   0.02180043 0.02262856 0.02079684 0.01511986 0.01887621 0.03871470
Male SD     0.07911264 0.21107322 0.14798379 0.04035185 0.12124391 0.03363656
                   [,7]       [,8]       [,9]      [,10]      [,11]
Female Mean 0.20995912 0.20889014 0.19920944 0.16731386 0.16652783
Male Mean   0.46238267 0.46621029 0.47629489 0.46236851 0.39731254
Female SD   0.03532426 0.02814304 0.02518857 0.01615855 0.01887127
Male SD     0.02354006 0.03209559 0.03023546 0.06024623 0.08535684
```

Next, identify windows within sample-chromosome pairs that have very high BAF standard deviations compared to the same window in other samples.

```
> res12bin4sd <- findBAFvariance(sds.chr, slidingBAF12, scanAnnot$sex,
+                                sd.threshold=4)
> head(res12bin4sd)

     scanID chromosome bin sex
[1,] "322"  "21"       "1" "M"
[2,] "324"  "21"       "3" "F"
[3,] "350"  "21"       "2" "F"
[4,] "296"  "22"       "2" "M"
[5,] "297"  "22"       "2" "M"
[6,] "324"  "22"       "2" "F"

> table(res12bin4sd[, "chromosome"])

21 22
 3  3
```
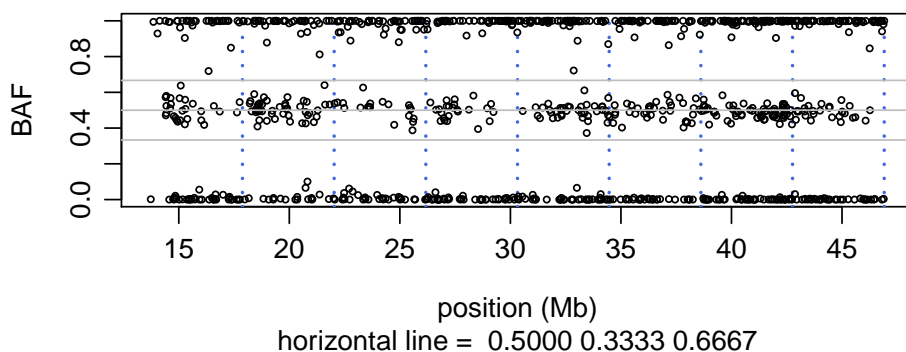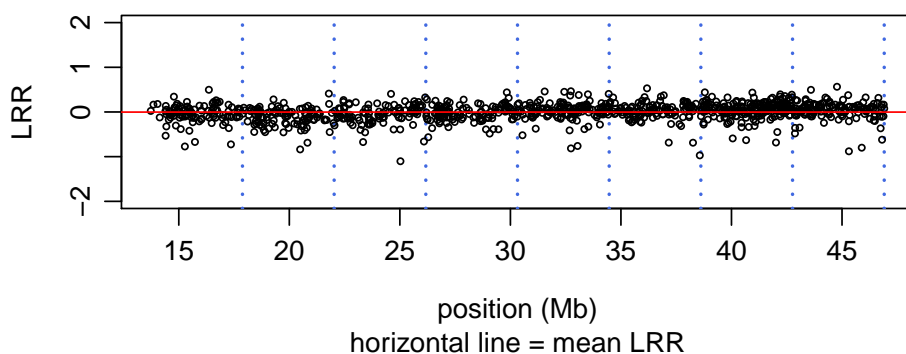
Call `chromIntensityPlot` to plot the BAF of all SNPs on the indicated chromosome-sample pairs against position. This yields many plots that must be individually examined to distinguish noisy data from chromosomal abnormalities.

```
> scanID <- as.integer(res12bin4sd[, "scanID"])
> chrom <- as.integer(res12bin4sd[, "chromosome"])
> chrom[res12bin4sd[, "chromosome"] == "X"] <- 23
> bincode <- paste("Bin", res12bin4sd[, "bin"], sep = " ")
> chromIntensityPlot(blData, scanID, chrom, info=bincode, ideogram=FALSE)
> close(blData)
```

Scan 322 – M – Chromosome 21 – Bin 1

position (Mb)
horizontal line = mean LRR

position (Mb)
horizontal line = 0.5000 0.3333 0.6667

At this stage, we have generated plots of those chromosomes (over all chromosomes and samples) that have unusually high BAF standard deviation. The next step in the process is to examine each of these plots to look for evidence of sample contamination or other quality issues.

4.3 Missingness and heterozygosity within samples

This step calculates the percent of missing and heterozygous genotypes in each chromosome of each sample. We create boxplots of missingness by individual chromosome, as well as autosomal and X chromosome heterozygosity in each population. This allows for identification of samples that may have relatively high heterozygosity for all chromosomes, indicating a possible mixed sample. Further, we are able to identify any outliers with regard to missingness. Plotting by chromosome enables visualization of chromosomal artifacts on a particular subset of SNPs that lie on a chromosome.
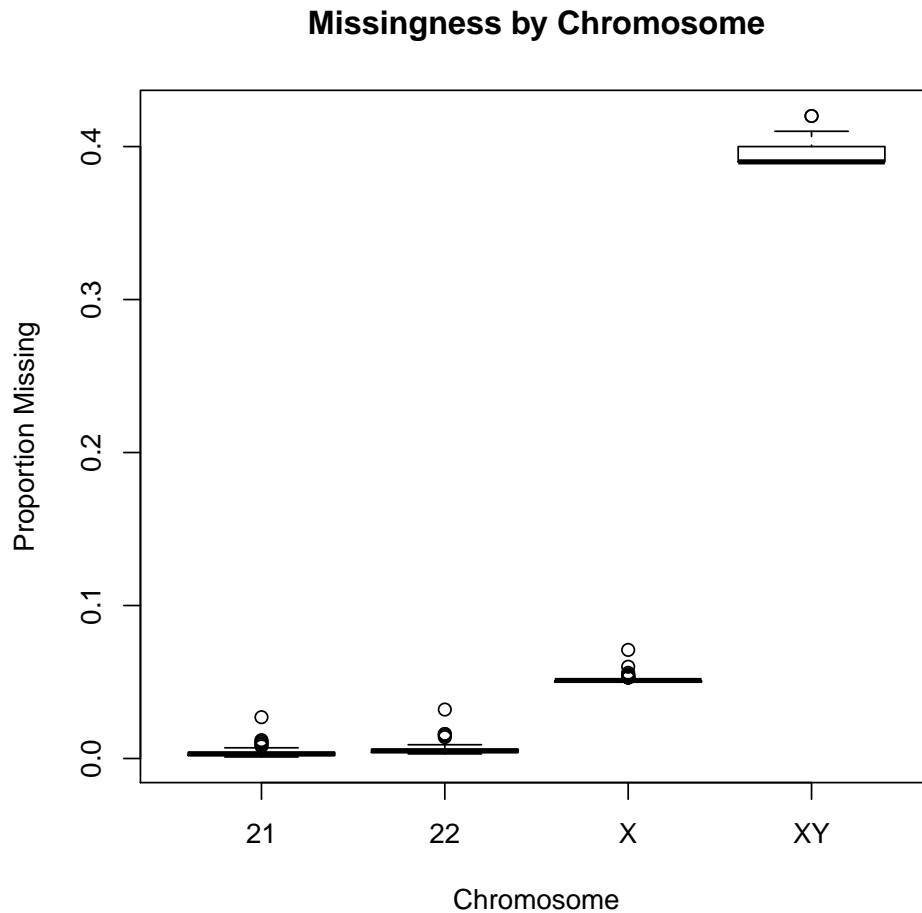
We will call the function `missingGenotypeByScanChrom` to calculate the missing call rate. Since the function returns missing counts per chromosome as well as snps per chromosome, we divide to find the missing call rate per chromosome. We then make a boxplot of missingness in the autosomes, the X chromosome, and the pseudoautosomal region, and a boxplot of X chromosome missingness for each sex.

```
> miss <- missingGenotypeByScanChrom(genoData)
> miss.rate <- t(apply(miss$missing.counts, 1, function(x) {
+   x / miss$snps.per.chr}))
> miss.rate <- as.data.frame(miss.rate)

> cols <- names(miss.rate) %in% c(1:22, "X", "XY")
> boxplot(miss.rate[,cols], main="Missingness by Chromosome",
+   ylab="Proportion Missing", xlab="Chromosome")
```
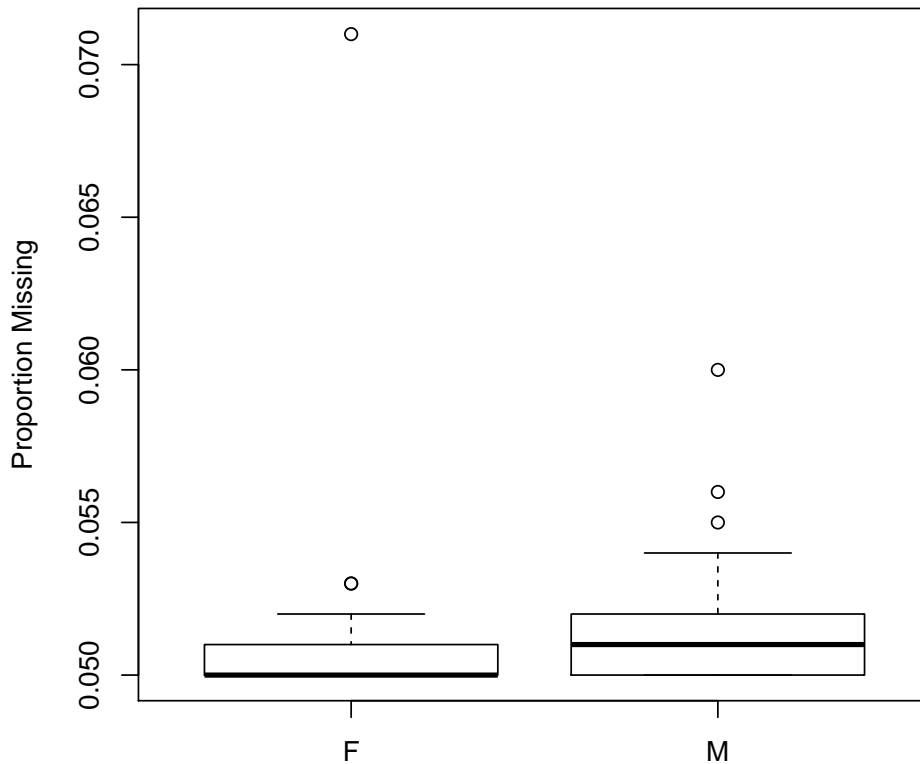


```
> boxplot(miss.rate$X ~ scanAnnot$sex,
+   main="X Chromosome Missingness by Sex",
+   ylab="Proportion Missing")
```

X Chromosome Missingness by Sex

We will call the function `hetByScanChrom` to calculate the heterozygosity. We store the heterozygosity calculations in the sample annotation.

```
> # Calculate heterozygosity by scan by chromosome
> het.results <- hetByScanChrom(genoData)
> close(genoData)
> # Ensure heterozygosity results are ordered correctly
> allequal(scanAnnot$scanID, rownames(het.results))

[1] TRUE

> # Write autosomal and X chr heterozygosity to sample annot
> scanAnnot$het.A <- het.results[,"A"]
> scanAnnot$het.X <- het.results[,"X"]
> varMetadata(scanAnnot)["het.A", "labelDescription"] <-
+    "fraction of heterozygotes for autosomal SNPs"
> varMetadata(scanAnnot)["het.X", "labelDescription"] <-
+    "fraction of heterozygotes for X chromosome SNPs"
```

There are two plots for heterozygosity. First is a boxplot of heterozygosity over the autosomes, subsetted by population. We recommend examining BAF plots for high heterozygosity outliers, to look for evidence of sample contamination (more than 3 bands on all chromosomes). Examination of low heterozygosity samples may also identify chromosomal anomalies with wide splits in the intermediate BAF band. Second is a boxplot of female heterozygosity on the X chromosome, subsetted by population.

```
> boxplot(scanAnnot$het.A ~ scanAnnot$race,
+    main="Autosomal Heterozygosity")
```

Autosomal Heterozygosity
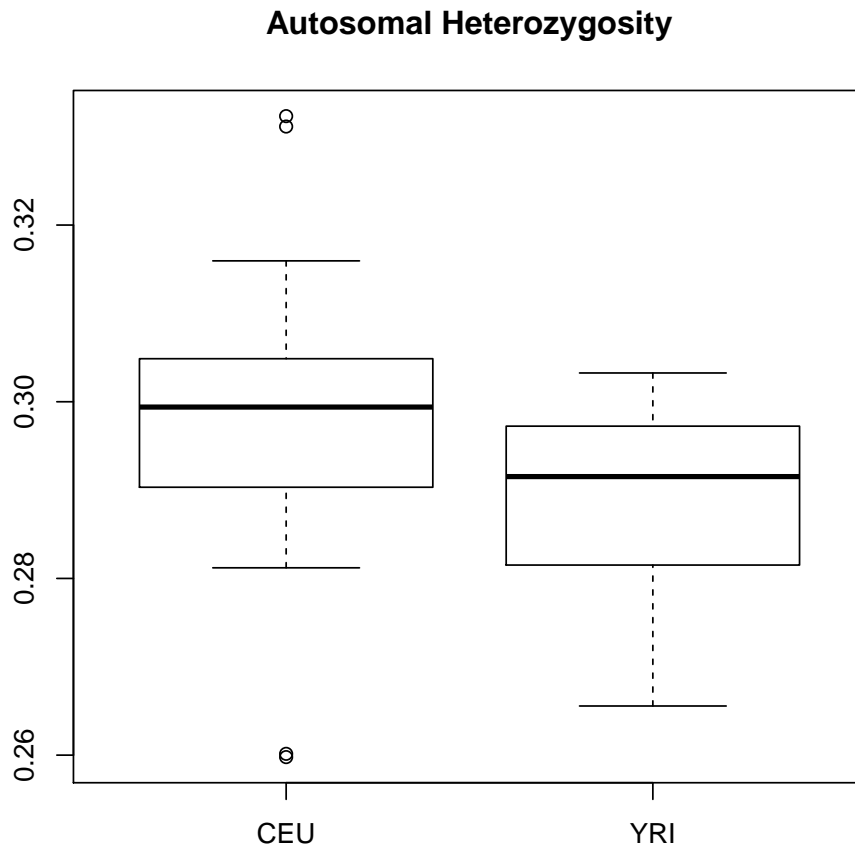

```
> female <- scanAnnot$sex == "F"
> boxplot(scanAnnot$het.X[female] ~ scanAnnot$race[female],
+    main="X Chromosome Heterozygosity in Females")
```

X Chromosome Heterozygosity in Females
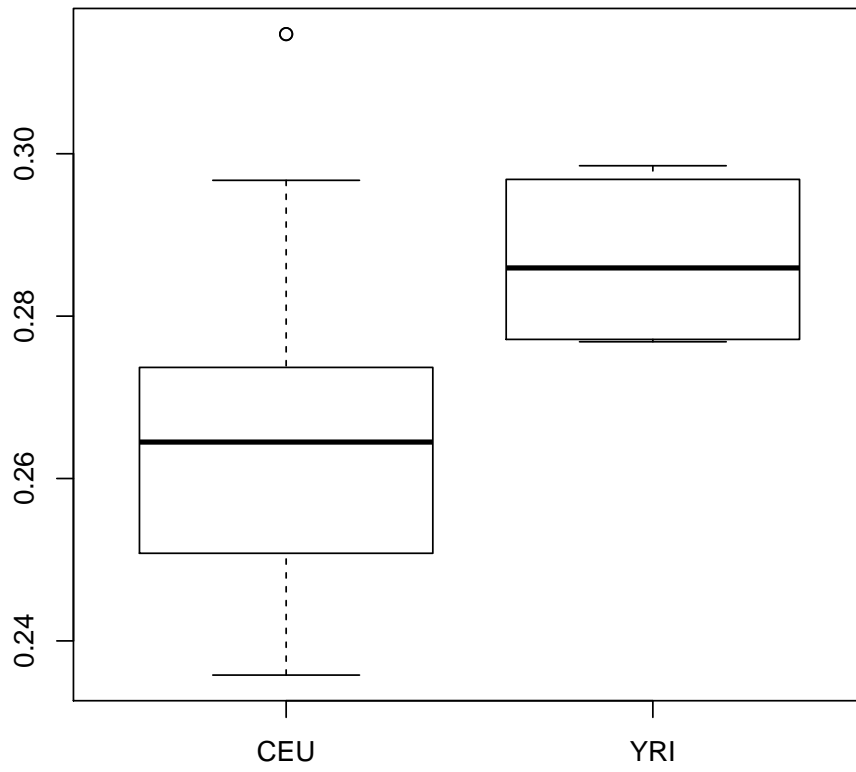
5 Sample Identity Checks

This step performs a series of identity checks on the samples. First, samples are analyzed to determine if there exist any discrepancies between the annotated sex and genetic sex in the sample. Next, the relatedness among samples is investigated through IBD estimation. Finally, the samples are checked for potential population substructure, which if unidentified can threaten the validity of subsequent analyses.

5.1 Mis-annotated Sex Check

This section looks for discrepancies between the annotated sex and genetic sex. Sex is usually inferred from X chromosome heterozygosity, but our experience is that this variable can give ambiguous results when used alone (for example, in XXY males or due to genotyping artifacts). Plots of the mean allelic intensities of SNPs on the X and Y chromosomes can identify mis-annotated sex as well as sex chromosome aneuploidies. It is important to have accurate sex annotation not only for completeness but also for analyses which treat male and female samples separately. Any found sex mis-annotations are presented to the investigators in order to resolve discrepancies. If a genetic and recorded sex do not match, a collective decision must be made regarding the inclusion of those genetic data. In some cases a recording error explains the discrepancy, but more often the discrepancy is unexplained. These cases are assumed to be a sample mis-identification and these samples are excluded from subsequent analyses.

In order to compare the mean X and Y chromosome intensities for all samples, we must calculate the mean intensity for each sample by chromosome. The function `meanIntensityByScanChrom` calculates for each sample the mean and standard deviation of the sum of the two allelic intensities for each probe on a given chromosome. A matrix with one row per sample and one column per chromosome with entries $[i, j]$ corresponding to either the mean or standard deviation of all probe intensities for the ith sample and the jth chromosome is returned from the function. Note that "X" and "Y" in the list names refer to the X and Y intensity values and not to the chromosomes.

```
> qxyfile <- system.file("extdata", "illumina_qxy.gds", package="GWASdata")
> intenGDS <- GdsIntensityReader(qxyfile)
> inten.by.chrom <- meanIntensityByScanChrom(intenGDS)
> close(intenGDS)
> names(inten.by.chrom)

[1] "mean.intensity" "sd.intensity"   "mean.X"         "sd.X"
[5] "mean.Y"         "sd.Y"
```

Now we will use the calculated mean intensities by sample to identify any sex mis-annotation or sex chromosome aneuploidies. For the plots, we will create a color coding corresponding to the annotated sex, with blue for males and red for females. We also use the SNP annotation to find the probe counts for the X and Y chromosomes; we use these in the plot axis labels.

```
> mninten <- inten.by.chrom[[1]]   # mean intensities
> dim(mninten)

[1] 77  6
```

```
> # Check to be sure sample ordering is consistent
> allequal(scanAnnot$scanID, rownames(mninten))

[1] TRUE

> # Assign each sex a color
> xcol <- rep(NA, nrow(scanAnnot))
> xcol[scanAnnot$sex == "M"] <- "blue"
> xcol[scanAnnot$sex == "F"] <- "red"
> nx <- sum(snpAnnot$chromosome == 23)
> ny <- sum(snpAnnot$chromosome == 25)
```

For two of the plots we will create next, we use the autosome and X chromosome heterozygosity values calculated in an earlier step and stored in the sample annotation. Four plots will now be created: mean X chromosome intensity versus mean Y chromosome intensity, mean X chromosome intensity versus X chromosome heterozygosity, mean X chromosome heterozygosity versus mean Y chromosome intensity and mean autosomal heterozygosity versus mean X chromosome heterozygosity. The fourth plot applies to annotated females only, since males are expected to have zero heterozygosity on the X chromosome.

```
> #All intensities
> x1 <-mninten[,"X"]; y1 <- mninten[,"Y"]
> main1 <- "Mean X vs \nMean Y Chromosome Intensity"
> #Het on X vs X intensity
> x2 <- mninten[,"X"]; y2 <- scanAnnot$het.X
> main2 <- "Mean X Chromosome Intensity vs
+   Mean X Chromosome Heterozygosity"
> # Het on X vs Y intensity
> y3 <- mninten[,"Y"]; x3 <- scanAnnot$het.X
> main3 <- "Mean X Chromosome Heterozygosity vs
+   Mean Y Chromosome Intensity"
> # X vs A het
> x4 <- scanAnnot$het.A[scanAnnot$sex == "F"]
> y4 <- scanAnnot$het.X[scanAnnot$sex == "F"]
> main4 <- "Mean Autosomal Heterozygosity vs
+   Mean X Chromosome Heterozygosity"
> cols <- c("blue","red")
> mf <- c("male", "female")
> xintenlab <- paste("X intensity (n=", nx, ")", sep="")
> yintenlab <- paste("Y intensity (n=", ny, ")", sep="")
> pdf("DataCleaning-sex.pdf")
> par(mfrow=c(2,2))
> plot(x1, y1, xlab=xintenlab, ylab=yintenlab,
+   main=main1, col=xcol, cex.main=0.8)
> legend("topright",mf,col=cols,pch=c(1,1))
> plot(x2, y2, col=xcol, xlab=xintenlab,
```

```
+    ylab="X heterozygosity", main=main2, cex.main=0.8)
> plot(x3, y3, col=xcol, ylab=yintenlab,
+    xlab="X heterozygosity", main=main3, cex.main=0.8)
> plot(x4,y4, col="red", xlab="Autosomal heterozygosity",
+    ylab="X heterozygosity", main=main4, cex.main=0.8)
> dev.off()
```
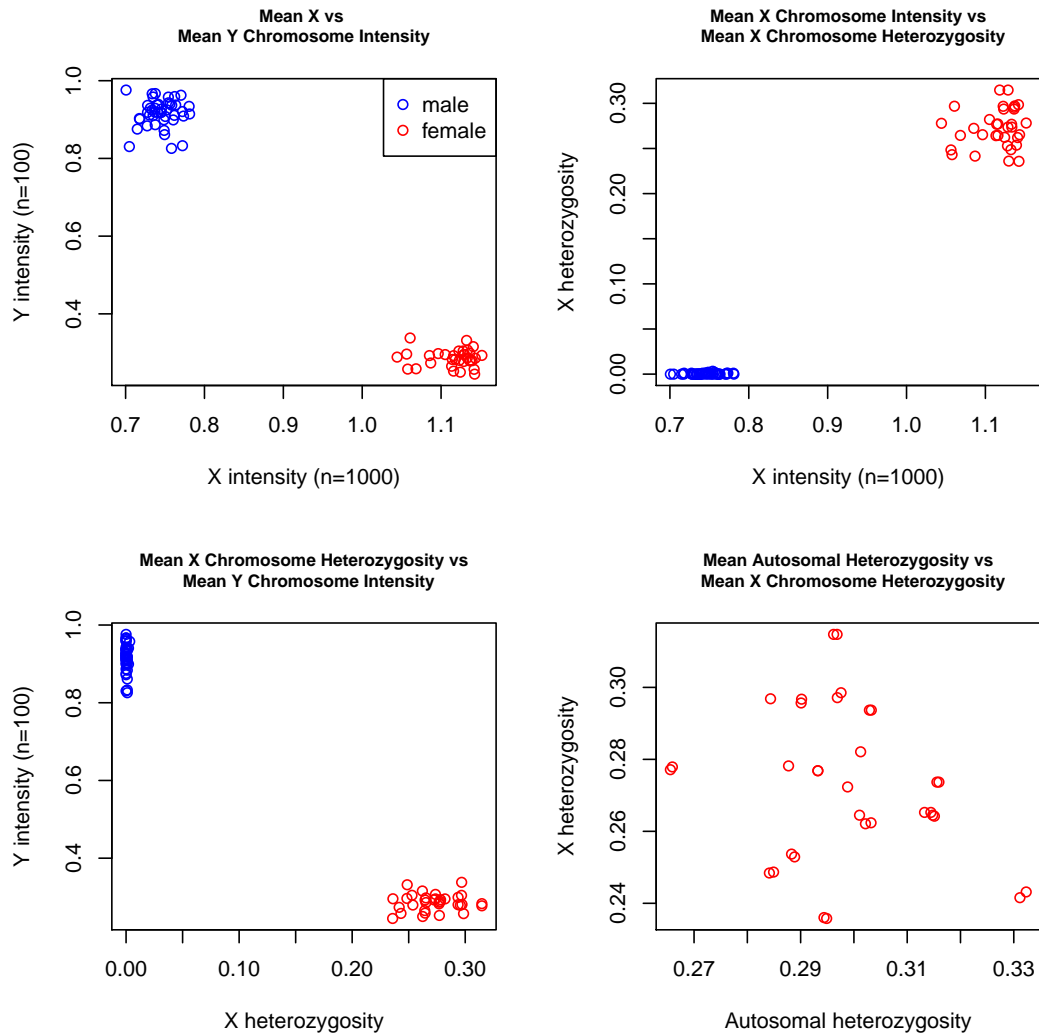

5.2 Relatedness and IBD Estimation

In most studies, there are discrepancies between pedigrees provided and relatedness inferred from the genotype data. To infer genetic relatedness, we estimate coefficients of identity by descent (IBD). It is important to identify and record unannotated relationships so that analyses assuming all subjects are unrelated can use a filtered subset of samples. From our experience, it is difficult to accurately estimate low levels of relatedness, but higher levels can be more reliably determined.

Users are encouraged to employ analyses which take into accounts the IBD estimates themselves rather than discrete relationship coefficients for any relationships.

The *SNPRelate* package includes three methods for calculating IBD: maximum likelihood estimation (MLE), which is accurate but computationally intensive, PLINK Method of Moments (MoM), which is faster but does not perform well with multiple ancestry groups analyzed together, and KING, which is robust to population structure[3]. This example will use the KING method.

```
> library(SNPRelate)
> gdsfile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
> gdsobj <- snpgdsOpen(gdsfile)
> ibdobj <- snpgdsIBDKING(gdsobj)

IBD analysis (KING method of moment) on genotypes:
Excluding 1,300 SNPs on non-autosomes
Excluding 39 SNPs (monomorphic: TRUE, MAF: NaN, missing rate: NaN)
Working space: 77 samples, 1,961 SNPs
    using 1 (CPU) core
No family is specified, and all individuals are treated as singletons.
Relationship inference in the presence of population stratification.
KING IBD:    the sum of all selected genotypes (0,1,2) = 134860
CPU capabilities: Double-Precision SSE2
Thu May 24 19:23:57 2018    (internal increment: 65536)

[..................................................]  0%, ETC: ---
[==================================================] 100%, completed in 0s
Thu May 24 19:23:57 2018    Done.

> snpgdsClose(gdsobj)
> names(ibdobj)

[1] "sample.id" "snp.id"    "afreq"     "IBS0"      "kinship"

> dim(ibdobj$kinship)

[1] 77 77

> ibdobj$kinship[1:5,1:5]

            [,1]       [,2]       [,3]      [,4]       [,5]
[1,] 0.50000000 0.50000000 0.2430314 0.2435009 0.01695652
[2,] 0.50000000 0.50000000 0.2425829 0.2430556 0.01611498
[3,] 0.24303136 0.24258290 0.5000000 0.4995645 0.26872822
[4,] 0.24350087 0.24305556 0.4995645 0.5000000 0.27000000
[5,] 0.01695652 0.01611498 0.2687282 0.2700000 0.50000000
```

---

[3]Manichaikul et al, Robust relationship inference in genome-wide association studies, Bioinformatics, 26(22):2867–2873, 2010

We find the expected relationships between samples based on the pedigree data that is stored in the sample annotation file. We will create a subset of the sample annotation that has one line per sample and columns that hold family, father and mother ids, where an entry of 0 indicates no familial data. Then the function `pedigreeCheck` is called, which determines if there are any duplicates, singleton families, mothers/fathers whose sex does not match, impossible relationships, subfamilies, or missing entries.

```
> ped <- pData(scanAnnot)[,c("family", "subjectID", "father", "mother", "sex")]
> dim(ped)

[1] 77  5

> names(ped) <- c("family", "individ", "father", "mother", "sex")
> ped[1:5, ]

  family   individ    father    mother sex
1   1341 200191449         0         0   F
2   1341 200191449         0         0   F
3   1341 200030290 200099417 200191449   F
4   1341 200030290 200099417 200191449   F
5   1341 200099417         0         0   M

> (chk <- pedigreeCheck(ped))

$duplicates
   family   individ copies match
1    1341 200191449      2  TRUE
2    1341 200030290      2  TRUE
3    1341 200099417      2  TRUE
4    1341 200015835      2  TRUE
5    1340 200099636      2  TRUE
6    1340 200076256      2  TRUE
7    1408 200074814      2  TRUE
8    1408 200094287      2  TRUE
9    1408 200019401      2  TRUE
10   1334 200016815      2  TRUE
11   1334 200118596      2  TRUE
12   1334 200019634      2  TRUE
13   1344 200071490      2  TRUE
14   1344 200116780      2  TRUE
15   1344 200005043      2  TRUE
16   1347 200121301      2  TRUE
17   1347 200009887      2  TRUE
18   1347 200044196      2  TRUE
19   1362 200024383      2  TRUE
20   1362 200187448      2  TRUE
21   1362 200169440      2  TRUE
```

```
22    1362 200018192      2   TRUE
23    1362 200118709      2   TRUE
24       4 200047857      2   TRUE
25       4 200073630      2   TRUE
26       5 200102386      2   TRUE
27       5 200079101      2   TRUE
28       5 200022488      2   TRUE
29       9 200066330      2   TRUE
30       9 200033736      2   TRUE
31      12 200066777      2   TRUE
32      12 200160551      2   TRUE
33      28 200003216      2   TRUE
34      28 200034659      2   TRUE


$parent.no.individ.entry
  row.num family no_individ_entry  parentID
1      13   1341             mother 200039107
2      14   1341             mother 200039107


$subfamilies.ident
  family subfamily    individ
1     58         1 200122151
2     58         2 200105428
```

The functions that determine expected relationships require no duplicates in the pedigree, so we remove them with `pedigreeDeleteDuplicates`.

```
> dups <- chk$duplicates
> uni.ped <- pedigreeDeleteDuplicates(ped, dups)
> (chk <- pedigreeCheck(uni.ped))

$parent.no.individ.entry
  row.num family no_individ_entry  parentID
1       8   1341             mother 200039107


$subfamilies.ident
  family subfamily    individ
7     58         1 200122151
8     58         2 200105428
1   1362         1 200003297
2   1362         1 200169440
3   1362         1 200187448
6   1362         2 200018192
4   1362         2 200024383
5   1362         2 200118709
```

There is one parent with no individual entry, so we add a row for that parent.

```
> ni <- chk$parent.no.individ.entry
> parent <- data.frame(family=ni$family, individ=ni$parentID,
+                      father=0, mother=0, sex="F",
+                      stringsAsFactors=FALSE)
> ped.complete <- rbind(uni.ped, parent)
> (chk <- pedigreeCheck(ped.complete))

$subfamilies.ident
   family subfamily   individ
13     58         1 200122151
14     58         2 200105428
2    1341         1 200030290
3    1341         1 200099417
1    1341         1 200191449
5    1341         2 200015835
6    1341         2 200039107
4    1341         2 200122600
7    1362         1 200003297
8    1362         1 200169440
9    1362         1 200187448
12   1362         2 200018192
10   1362         2 200024383
11   1362         2 200118709
```

There are multiple subfamilies identified, so we will need to assign new family IDs to the subfamilies. One subfamily has two unrelated people (likely founders), so we remove this family from the pedigree.

```
> ped.complete <- ped.complete[ped.complete$family != 58,]
> subf <- chk$subfamilies.ident
> table(subf$family)

  58 1341 1362
   2    6    6

> subf.ids <- subf$individ[subf$subfamily == 2]
> newfam <- ped.complete$individ %in% subf.ids
> ped.complete$family[newfam] <- paste0(ped.complete$family[newfam], "-2")
> table(ped.complete$family)

    12   1334   1340   1341 1341-2   1344   1347   1362 1362-2   1408     28
     3      3      3      3      3      3      3      3      3      3      3
            4      5      9
            3      3      3

> pedigreeCheck(ped.complete)
```

```
NULL
```

The revised pedigree passes all checks. Now from the verified sample list excluding duplicate samples, we can calculate the expected relationships among the samples by calling the function `pedigreePairwiseRelatedness`. The relationships looked for as annotated include: unrelated (U), parent/offspring (PO), full siblings (FS), second-degree relatives (half-siblings, avuncular and grandparent-grandchild), and third-degree relatives (first cousins). Families where mothers and fathers are related are also looked for among the family annotations.

```
> rels <- pedigreePairwiseRelatedness(ped.complete)
> length(rels$inbred.fam)

[1] 0

> relprs <- rels$relativeprs
> relprs[1:5,]

   Individ1  Individ2 relation kinship family
1 200191449 200030290       PO    0.25   1341
2 200191449 200099417        U    0.00   1341
3 200030290 200099417       PO    0.25   1341
4 200099636 200032162        U    0.00   1340
5 200099636 200076256       PO    0.25   1340

> table(relprs$relation)

PO  U
28 14
```

In order to plot the IBD coefficient estimates color coded by expected relationships, we retrieve a data.frame of sample pairs with $KC > 1/32$. The samples must be coded in terms of subject id and each pair of samples must be annotated with the expected relationship. We can also assign observed relationships based on the values of $k0$ and $k1$.

```
> samp <- pData(scanAnnot)[,c("scanID", "subjectID")]
> samp <- samp[match(ibdobj$sample.id, samp$scanID),]
> names(samp) <- c("scanID", "Individ")
> ibd <- snpgdsIBDSelection(ibdobj, kinship.cutoff=1/32)
> ibd <- merge(ibd, samp, by.x="ID1", by.y="scanID")
> ibd <- merge(ibd, samp, by.x="ID2", by.y="scanID", suffixes=c("1","2"))
> ibd$ii <- pasteSorted(ibd$Individ1, ibd$Individ2)
> relprs$ii <- pasteSorted(relprs$Individ1, relprs$Individ2)
> ibd <- merge(ibd, relprs[,c("ii","relation")], all.x=TRUE)
> names(ibd)[names(ibd) == "relation"] <- "exp.rel"
> ibd$exp.rel[ibd$Individ1 == ibd$Individ2] <- "Dup"
> ibd$exp.rel[is.na(ibd$exp.rel)] <- "U"
> table(ibd$exp.rel, useNA="ifany")
```

```
Dup  PO   U
 34  90 130
```

```
> # assign observed relationships
> ibd$obs.rel <- ibdAssignRelatednessKing(ibd$IBS0, ibd$kinship)
> table(ibd$obs.rel, useNA="ifany")
```

```
Deg3  Dup   PO    U
  66   34   90   64
```

Now the pedigree information is in the proper format for the IBD estimates to be plotted for each pair of samples, color coded by expected relationship.

```
> ## thresholds for assigning relationships using kinship coefficients
> ## in table 1 of Manichaikul (2010)
> cut.dup <- 1/(2^(3/2))
> cut.deg1 <- 1/(2^(5/2))
> cut.deg2 <- 1/(2^(7/2))
> cut.deg3 <- 1/(2^(9/2))
> cols <- c(Dup="magenta", PO="cyan", U="black")
> plot(ibd$IBS0, ibd$kinship, col=cols[ibd$exp.rel],
+      xlab="Fraction of IBS=0", ylab="Kinship coefficient")
> abline(h=c(cut.deg1, cut.deg2, cut.deg3, cut.dup), lty=2, col="gray")
> legend("topright", legend=names(cols), col=cols, pch=1)
```
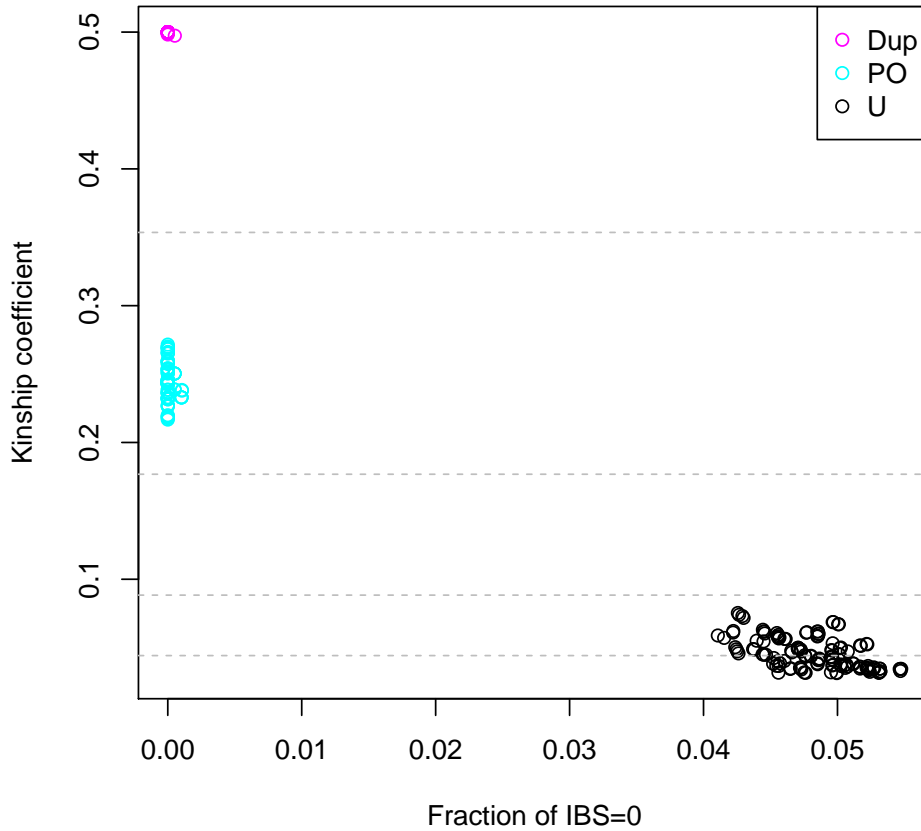
5.3 Population Structure

Principal Component Analysis on all ethnic groups

In this section, we perform principal component analysis (PCA) in order to detect any population substructure that may exist among samples in a study. After calculating the eigenvectors for the samples, we plot the values for each of the first 4 eigenvectors in a pairwise fashion for each individual. By color coding the plots by annotated race and/or ethnicity, we can identify any individuals whose recorded self-identified race/ethnicity differs from their inferred genetic ancestry. Further, we can use the PCA-identified continental ancestry when stratifying samples by population group. It may also be useful to include the values of some eigenvectors as covariates in association tests.

For PCA, we use linkage disequilibrium (LD) pruning (`snpgdsLDpruning`) to select a set of SNPs within which each pair has a low level of LD (e.g. $r^2 < 0.1$ in a sliding 10 Mb window), from a starting pool of autosomal SNPs with `missing.n1` < 0.05 and MAF < 0.05. We also remove SNPs in regions with known correlation (2q21 (LCT), HLA, 8p23, and 17q21.31). We must also ensure no duplicate samples are used for the principal component calculations.

```
> filt <- get(data(pcaSnpFilters.hg18))
> chrom <- getChromosome(snpAnnot)
> pos <- getPosition(snpAnnot)
> snpID <- getSnpID(snpAnnot)
> snp.filt <- rep(TRUE, length(snpID))
> for (f in 1:nrow(filt)) {
+    snp.filt[chrom == filt$chrom[f] & filt$start.base[f] < pos
+            & pos < filt$end.base[f]] <- FALSE
+ }
> snp.sel <- snpID[snp.filt]
> length(snp.sel)

[1] 3300

> sample.sel <- scanAnnot$scanID[!scanAnnot$duplicated]
> length(sample.sel)

[1] 43

> gdsobj <- snpgdsOpen(gdsfile)
> snpset <- snpgdsLDpruning(gdsobj, sample.id=sample.sel, snp.id=snp.sel,
+                           autosome.only=TRUE, maf=0.05, missing.rate=0.05,
+                           method="corr", slide.max.bp=10e6,
+                           ld.threshold=sqrt(0.1))

SNP pruning based on LD:
Excluding 1,300 SNPs (non-autosomes or non-selection)
Excluding 237 SNPs (monomorphic: TRUE, MAF: 0.05, missing rate: 0.05)
Working space: 43 samples, 1,763 SNPs
    using 1 (CPU) core
    sliding window: 10,000,000 basepairs, Inf SNPs
    |LD| threshold: 0.316228
    method: correlation
Chromosome 21: 8.70%, 87/1,000
Chromosome 22: 9.10%, 91/1,000
178 markers are selected in total.

> snp.pruned <- unlist(snpset, use.names=FALSE)
> length(snp.pruned)

[1] 178
```

The **snpgdsPCA** function is called with the SNP and sample subsets to calculate the first 32 eigenvectors.

```
> pca <- snpgdsPCA(gdsobj, sample.id=sample.sel, snp.id=snp.pruned)
```

```
Principal Component Analysis (PCA) on genotypes:
Excluding 3,122 SNPs (non-autosomes or non-selection)
Excluding 0 SNP (monomorphic: TRUE, MAF: NaN, missing rate: NaN)
Working space: 43 samples, 178 SNPs
    using 1 (CPU) core
PCA:    the sum of all selected genotypes (0,1,2) = 6290
CPU capabilities: Double-Precision SSE2
Thu May 24 19:23:57 2018    (internal increment: 75344)

[...............................................] 0%, ETC: ---
[=================================================] 100%, completed in 1s
Thu May 24 19:23:58 2018    Begin (eigenvalues and eigenvectors)
Thu May 24 19:23:58 2018    Done.

> names(pca)

[1] "sample.id" "snp.id"    "eigenval"  "eigenvect" "varprop"    "TraceXTX"
[7] "Bayesian"  "genmat"

> length(pca$eigenval)

[1] 43

> dim(pca$eigenvect)

[1] 43 32
```

We will make a pairs plot showing the first four eigenvectors. A simple calculation is made to find the fraction of variance among the samples as explained by each eigenvector.

```
> # Calculate the percentage of variance explained
> # by each principal component.
> pc.frac <- pca$eigenval/sum(pca$eigenval)
> lbls <- paste("EV", 1:4, "\n", format(pc.frac[1:4], digits=2), sep="")
> samp <- pData(scanAnnot)[match(pca$sample.id, scanAnnot$scanID),]
> cols <- rep(NA, nrow(samp))
> cols[samp$race == "CEU"] <- "blue"
> cols[samp$race == "YRI"] <- "red"

> pairs(pca$eigenvect[,1:4], col=cols, labels=lbls,
+   main = "CEU: blue, YRI: red")
```

CEU: blue, YRI: red
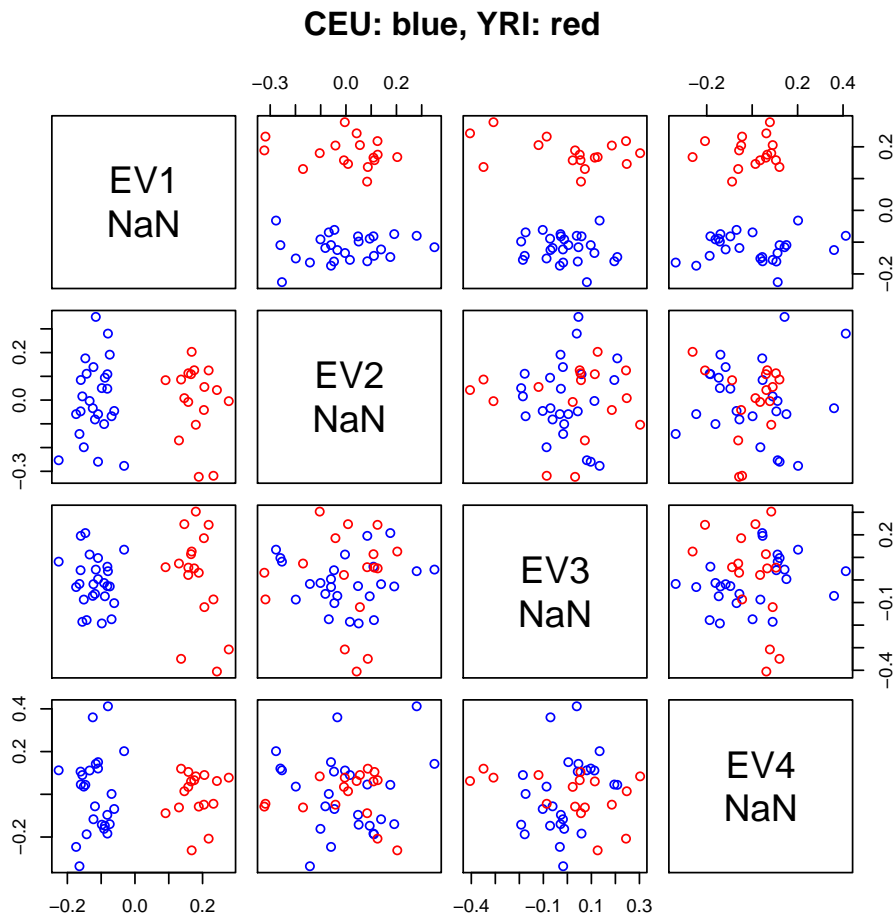
Parallel Coordinates Plot

A handy method of visualizing the effects of eigenvectors on clusters for a principal components analysis is the parallel coordinates plot. The genetic diversity in the YRI group is apparent in the later eigenvectors, while the remaining groups remain in clusters throughout.

```
> par.coord <- pca$eigenvect
> rangel <- apply(par.coord, 2, function(x) range(x)[1])
> rangeh <- apply(par.coord, 2, function(x) range(x)[2])
> std.coord <- par.coord
> for (i in 1:14)
+   std.coord[,i] <- (par.coord[,i] - rangel[i])/(rangeh[i]-rangel[i])
> plot(c(0,15), c(0,1), type = 'n', axes = FALSE, ylab = "", xlab = "",
+  main = "Parallel Coordinates Plot
+  CEU: blue, YRI: red")
> for (j in 1:13)
+   for (i in sample(1:nrow(std.coord)) )
```

```
+       lines(c(j,j+1), std.coord[i,c(j,j+1)], col=cols[i], lwd=0.25)
> axis(1, at = 1:14, labels = paste("PC",1:14, sep = "."))
```

Parallel Coordinates Plot
CEU: blue, YRI: red


PC.1   PC.3   PC.5   PC.7   PC.9   PC.11   PC.14
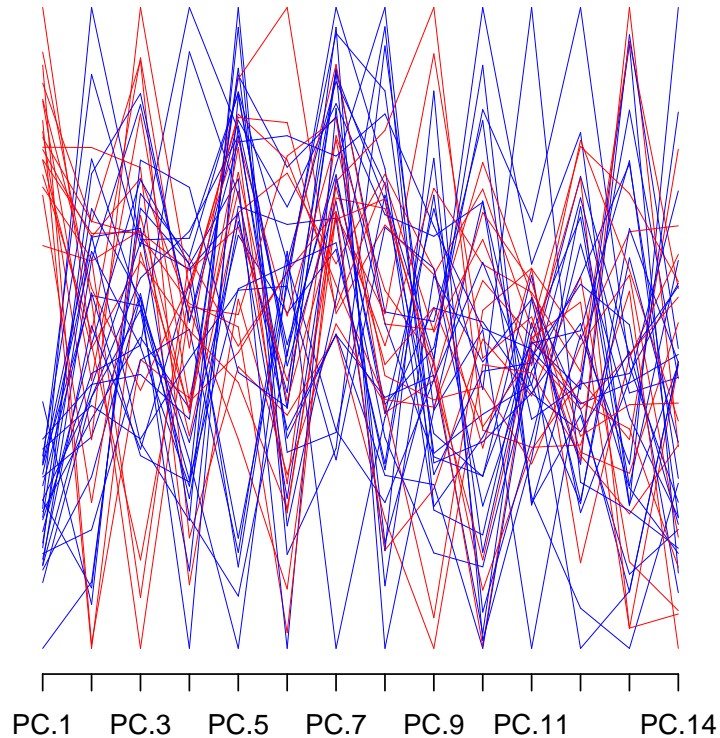
SNP-PC correlation

We confirm that there are no correlations between SNP regions and specific eigenvectors by examining plots of SNP correlation vs. position on the chromosome. Usually we check the first 8–12 eigenvectors, but here we plot only 1–4.

```
> corr <- snpgdsPCACorr(pca, gdsobj, eig.which=1:4)
> snpgdsClose(gdsobj)
> snp <- snpAnnot[match(corr$snp.id, snpID),]
> chrom <- getChromosome(snp, char=TRUE)
> pdf("DataCleaning-corr.pdf")
> par(mfrow=c(4,1))
> for (i in 1:4) {
+    snpCorrelationPlot(abs(corr$snpcorr[i,]), chrom,
```

59

```
+                              main=paste("Eigenvector",i), ylim=c(0,1))
+ }
> dev.off()
```

Eigenvector 1

Eigenvector 2

Eigenvector 3

Eigenvector 4
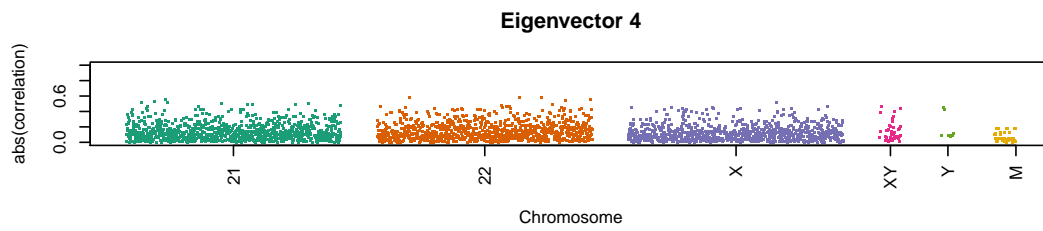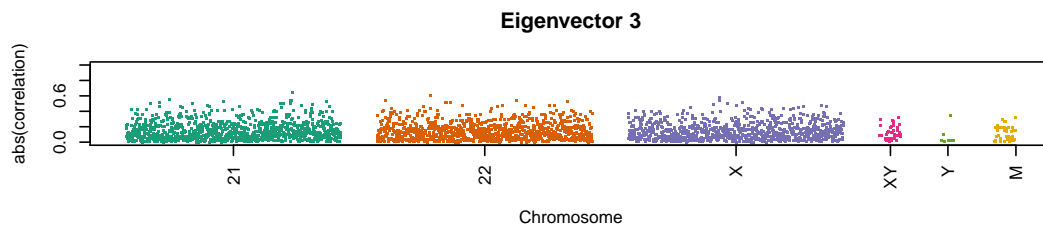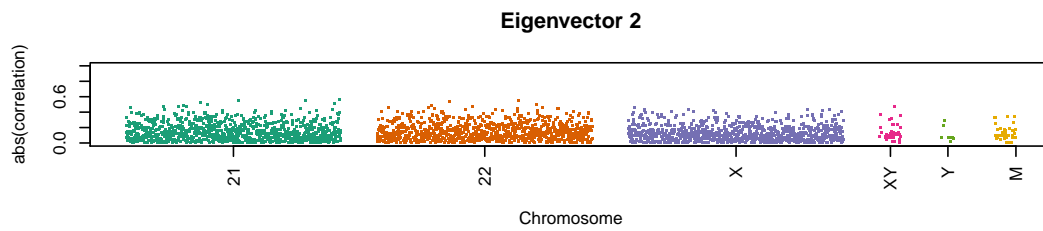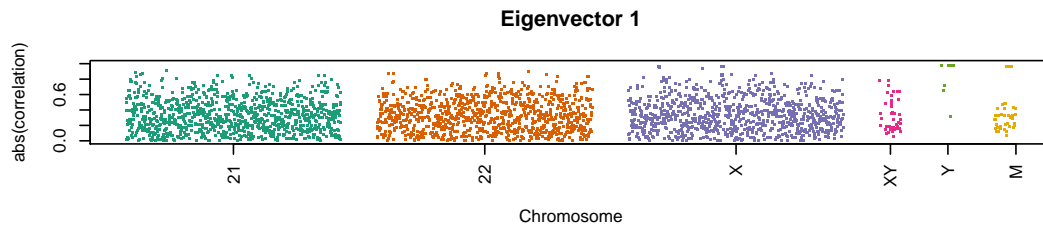
6 Case-Control Confounding

We recommend checking for case-control confounding as part of the data cleaning process for GWAS. This involves checking both the principal components and the missing call rate for a relationship with case status.

6.1 Principal Components Differences

This step examines differences in principal components according to case-control status.

Collate PCA information with sample number, case-control status, and population group.

```
> princomp <- as.data.frame(pca$eigenvect)
> samples.nodup <- pData(scanAnnot)[!scanAnnot$duplicated,]
> princomp$scanID <- as.factor(samples.nodup$scanID)
> princomp$case.ctrl.status <- as.factor(samples.nodup$status)
> princomp$race <- as.factor(samples.nodup$race)
```

The code below gives what percent of variation is accounted for by the principal component for the first 32 PCs.

```
> pc.percent <- 100 * pca$eigenval[1:32]/sum(pca$eigenval)
> pc.percent

 [1] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
[20] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN

> lbls <- paste("EV", 1:3, "\n", format(pc.percent[1:3], digits=2), "%", sep="")
> table(samples.nodup$status)

 0  1
21 21

> cols <- rep(NA, nrow(samples.nodup))
> cols[samples.nodup$status == 1] <- "green"
> cols[samples.nodup$status == 0] <- "magenta"
```

We plot the principal component pairs for the first three PCs by case-control status. We then make boxplots for the first few PCs to show differences between cases and controls, along with a two-factor ANOVA accounting for case-control status and population group. Since we are using randomized case-control status, we do not expect to see a significant difference in principal components between cases and controls, when considering population group.
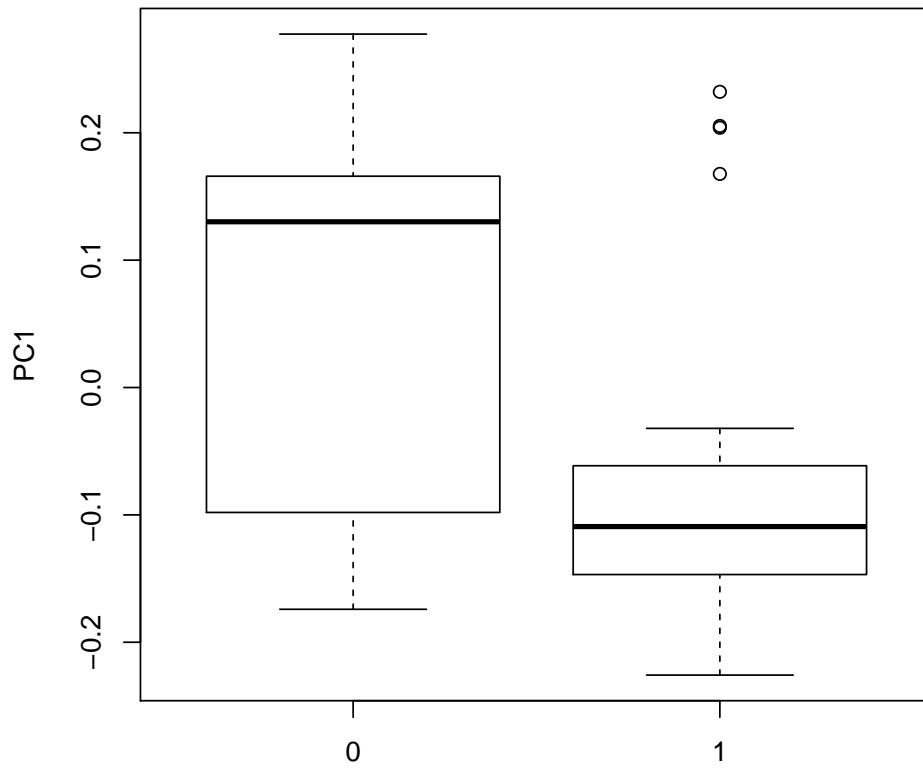
```
> pairs(pca$eigenvect[,1:3], col=cols, labels=lbls,
+   main = "First Three EVs by Case-Control Status")
```
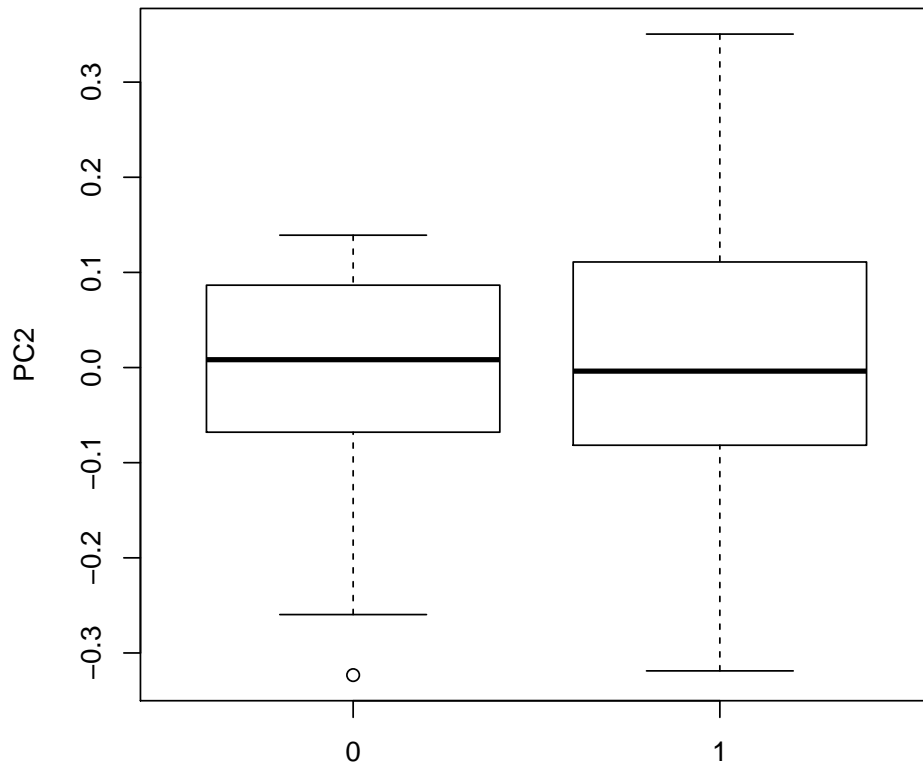
First Three EVs by Case−Control Status
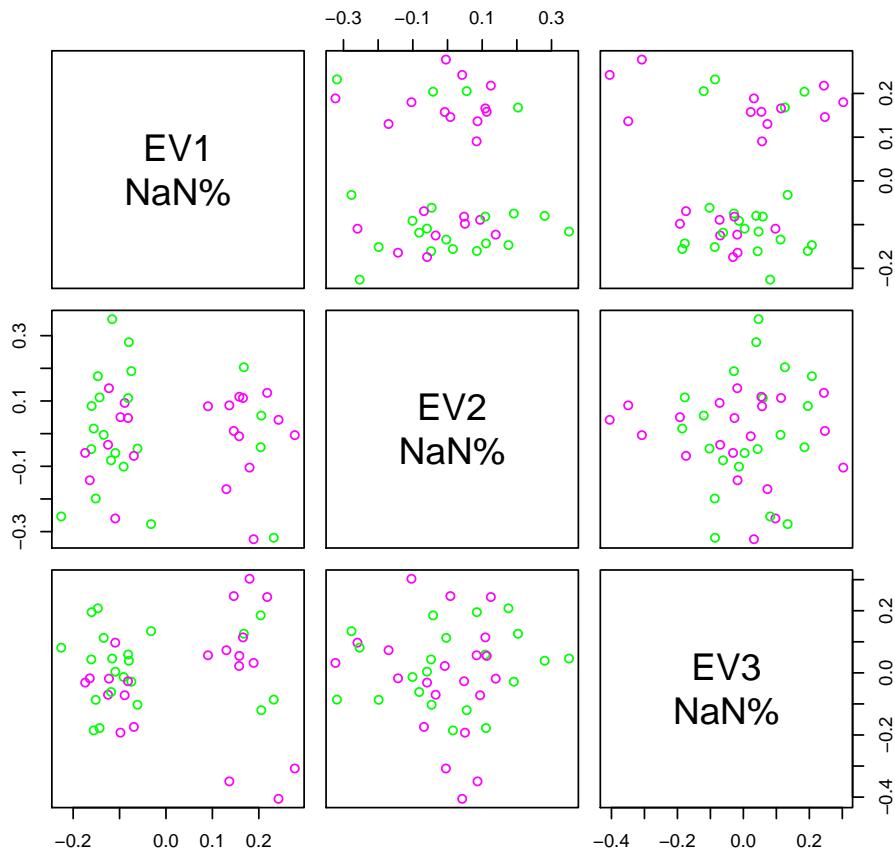

```
> boxplot(princomp[, 1] ~ princomp$case.ctrl.status,
+   ylab = "PC1", main = "PC1 vs. Case-control Status")
```
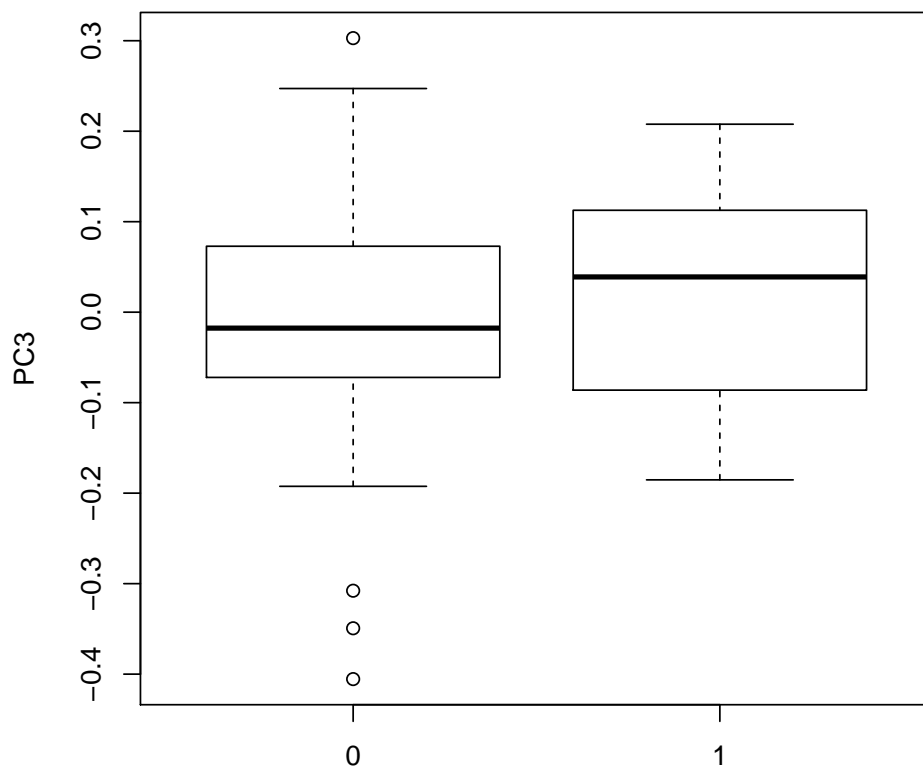
PC1 vs. Case–control Status

```
> boxplot(princomp[, 2] ~ princomp$case.ctrl.status,
+    ylab = "PC2", main = "PC2 vs. Case-control Status")
```

PC2 vs. Case−control Status


```
> boxplot(princomp[, 3] ~ princomp$case.ctrl.status,
+    ylab = "PC3", main = "PC3 vs. Case-control Status")
```

PC3 vs. Case–control Status


```
> aov.p1 <- aov(princomp[,1] ~ princomp$race *
+   princomp$case.ctrl.status, princomp)
> summary(aov.p1)

                                     Df Sum Sq Mean Sq F value Pr(>F)
princomp$race                         1 0.8889  0.8889 438.306 <2e-16 ***
princomp$case.ctrl.status             1 0.0003  0.0003   0.155  0.696
princomp$race:princomp$case.ctrl.status 1 0.0022  0.0022   1.091  0.303
Residuals                            38 0.0771  0.0020
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
1 observation deleted due to missingness

> aov.p2 <- aov(princomp[,2] ~ princomp$race *
+   princomp$case.ctrl.status, princomp)
> summary(aov.p2)

                                     Df Sum Sq  Mean Sq F value Pr(>F)
princomp$race                         1 0.0009 0.000923   0.036  0.850
```

```
princomp$case.ctrl.status                    1 0.0033 0.003339   0.131  0.720
princomp$race:princomp$case.ctrl.status  1 0.0078 0.007767   0.304  0.585
Residuals                                   38 0.9718 0.025574
1 observation deleted due to missingness

> aov.p3 <- aov(princomp[,3] ~ princomp$race *
+   princomp$case.ctrl.status, princomp)
> summary(aov.p3)

                                        Df Sum Sq  Mean Sq F value Pr(>F)
princomp$race                            1 0.0044 0.004395   0.174  0.679
princomp$case.ctrl.status                1 0.0260 0.025985   1.027  0.317
princomp$race:princomp$case.ctrl.status  1 0.0055 0.005531   0.219  0.643
Residuals                               38 0.9614 0.025300
1 observation deleted due to missingness
```

6.2   Missing Call Rate Differences

This step determines whether there are differences in missing call rates between cases and controls. As in section 6.1, we use simulated case-control status to demonstrate this step, since the HapMap II data does not contain information on cases and controls.

   Investigate the difference in mean missing call rate by case-control status, using the sample annotation variable `missing.e1`. Here, since the case-control status was randomly assigned, we do not expect to see a difference in any of the missing call rates with respect to case-control status.

```
> lm.all <- lm(scanAnnot$missing.e1 ~ scanAnnot$status)
> summary(aov(lm.all))

                  Df    Sum Sq   Mean Sq F value Pr(>F)
scanAnnot$status   1 0.0000647 6.474e-05    4.65 0.0343 *
Residuals         73 0.0010163 1.392e-05
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
2 observations deleted due to missingness

> boxplot(scanAnnot$missing.e1 ~ scanAnnot$status, ylab =
+   "Mean missing call rate", main="Mean missing call rate by case status")
```
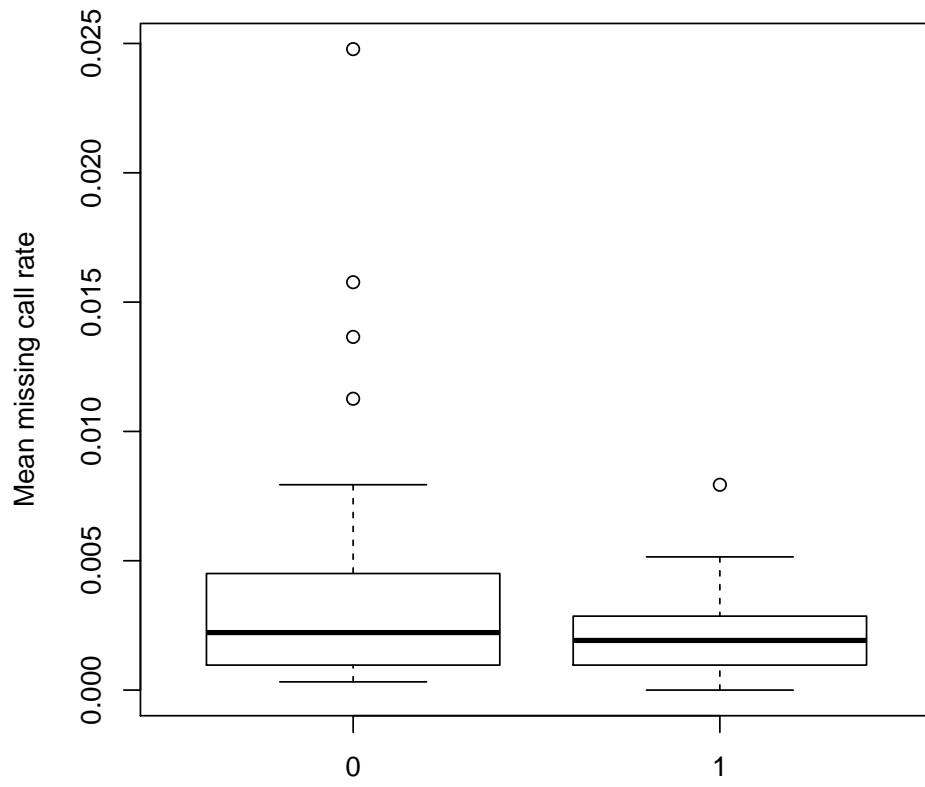
Mean missing call rate by case status

7   Chromosome Anomaly Detection

This step looks for large chromosomal anomalies that may be filtered out during the final analysis.

7.1   B Allele Frequency filtering

Create an `IntensityData` object and a `GenotypeData` object.

```
> blfile <- system.file("extdata", "illumina_bl.gds", package="GWASdata")
> blgds <- GdsIntensityReader(blfile)
> blData <-  IntensityData(blgds, snpAnnot=snpAnnot, scanAnnot=scanAnnot)
> genofile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
> genogds <- GdsGenotypeReader(genofile)
> genoData <-  GenotypeData(genogds, snpAnnot=snpAnnot, scanAnnot=scanAnnot)
```

Identify some low quality samples by looking at the standard deviation of BAF.

```
> baf.sd <- sdByScanChromWindow(blData, genoData, var="BAlleleFreq")
> med.baf.sd <- medianSdOverAutosomes(baf.sd)
> low.qual.ids <- med.baf.sd$scanID[med.baf.sd$med.sd > 0.05]
```

Decide which SNPs to exclude based on genome build.

```
> chrom <- getChromosome(snpAnnot, char=TRUE)
> pos <- getPosition(snpAnnot)
> hla.df <- get(data(HLA.hg18))
> hla <- chrom == "6" & pos >= hla.df$start.base & pos <= hla.df$end.base
> xtr.df <- get(data(pseudoautosomal.hg18))
> xtr <- chrom == "X" & pos >= xtr.df["X.XTR", "start.base"] &
+   pos <= xtr.df["X.XTR", "end.base"]
> centromeres <- get(data(centromeres.hg18))
> gap <- rep(FALSE, nrow(snpAnnot))
> for (i in 1:nrow(centromeres)) {
+   ingap <- chrom == centromeres$chrom[i] & pos > centromeres$left.base[i] &
+     pos < centromeres$right.base[i]
+   gap <- gap | ingap
+ }
> ignore <- snpAnnot$missing.n1 == 1 #ignore includes intensity-only and failed snps
> snp.exclude <- ignore | hla | xtr | gap
> snp.ok <- snpAnnot$snpID[!snp.exclude]
```

We use circular binary segmentation to find change points in BAF.

```
> scan.ids <- scanAnnot$scanID[1:10]
> chrom.ids <- 21:23
> baf.seg <- anomSegmentBAF(blData, genoData, scan.ids=scan.ids,
+   chrom.ids=chrom.ids, snp.ids=snp.ok, verbose=FALSE)
> head(baf.seg)
```

```
   scanID chromosome left.index right.index num.mark seg.mean
1    280         21          4         998      294   0.1669
2    280         22       1009        2000      302   0.1524
3    280         23       2020        2987      297   0.1587
4    281         21          4         998      293   0.1516
5    281         22       1009        2000      301   0.1410
6    281         23       2020        2987      297   0.1452
```

Filter segments to detect anomalies, treating the low quality samples differently.

```
> baf.anom <- anomFilterBAF(blData, genoData, segments=baf.seg,
+   snp.ids=snp.ok, centromere=centromeres, low.qual.ids=low.qual.ids,
+   verbose=FALSE)
> names(baf.anom)

[1] "raw"       "filtered"  "base.info" "seg.info"

> baf.filt <- baf.anom$filtered
> head(baf.filt)

   scanID chromosome left.index right.index num.mark seg.mean   sd.fac sex
19    286         22       1154        1163       10 0.401500 2.813941   M
23    287         22       1154        1163       10 0.371368 2.429629   M
   merge homodel.adjust left.base right.base frac.used
19 FALSE          FALSE  21110596   21276825         1
23 FALSE           TRUE  21110596   21276825         1
```

7.2 Loss of Heterozygosity

We look for Loss of Heterozygosity (LOH) anomalies by identifying homozygous runs with change in LRR. Change points in LRR are found by circular binary segmentation. Known anomalies from the BAF detection are excluded.

```
> loh.anom <- anomDetectLOH(blData, genoData, scan.ids=scan.ids,
+   chrom.ids=chrom.ids, snp.ids=snp.ok, known.anoms=baf.filt,
+   verbose=FALSE)
> names(loh.anom)

[1] "raw"          "raw.adjusted" "filtered"     "base.info"    "segments"
[6] "merge"

> loh.filt <- loh.anom$filtered
> head(loh.filt)

NULL
```

7.3 Statistics

Calculate statistics for the anomalous segments found with the BAF and LOH methods.

```
> # create required data frame
> baf.filt$method <- "BAF"
> if (!is.null(loh.filt)) {
+   loh.filt$method <- "LOH"
+   cols <- intersect(names(baf.filt), names(loh.filt))
+   anoms <- rbind(baf.filt[,cols], loh.filt[,cols])
+ } else {
+   anoms <- baf.filt
+ }
> anoms$anom.id <- 1:nrow(anoms)
> stats <- anomSegStats(blData, genoData, snp.ids=snp.ok, anom=anoms,
+                        centromere=centromeres)
> names(stats)

 [1] "scanID"                 "chromosome"
 [3] "left.index"             "right.index"
 [5] "num.mark"               "seg.mean"
 [7] "sd.fac"                 "sex"
 [9] "merge"                  "homodel.adjust"
[11] "left.base"              "right.base"
[13] "frac.used"              "method"
[15] "anom.id"                "nmark.all"
[17] "nmark.elig"             "nbase"
[19] "non.anom.baf.med"       "non.anom.lrr.med"
[21] "non.anom.lrr.mad"       "anom.baf.dev.med"
[23] "anom.baf.dev.5"         "anom.baf.dev.mean"
[25] "anom.baf.sd"            "anom.baf.mad"
[27] "anom.lrr.med"           "anom.lrr.sd"
[29] "anom.lrr.mad"           "nmark.baf"
[31] "nmark.lrr"              "cent.rel"
[33] "left.most"              "right.most"
[35] "left.last.elig"         "right.last.elig"
[37] "left.term.lrr.med"      "right.term.lrr.med"
[39] "left.term.lrr.n"        "right.term.lrr.n"
[41] "cent.span.left.elig.n"  "cent.span.right.elig.n"
[43] "cent.span.left.bases"   "cent.span.right.bases"
[45] "cent.span.left.index"   "cent.span.right.index"
[47] "bafmetric.anom.mean"    "bafmetric.non.anom.mean"
[49] "bafmetric.non.anom.sd"  "nmark.lrr.low"
```
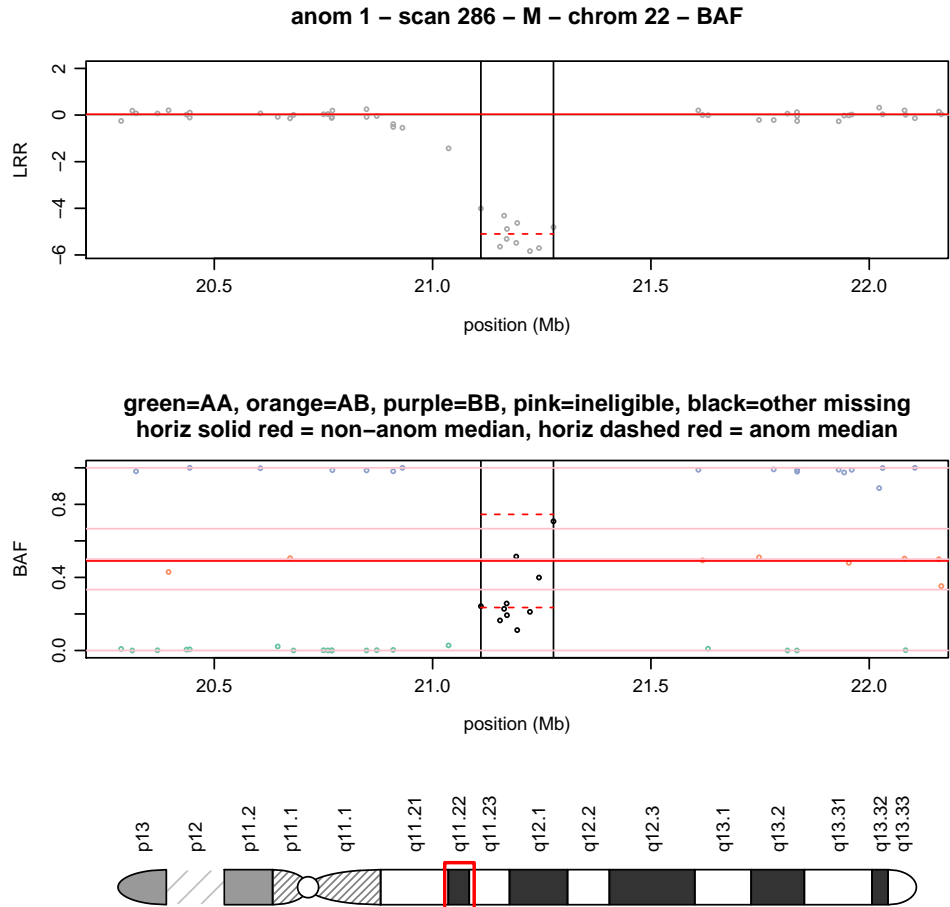
Plot the anomalies with relevant statistics, one anomaly per plot. Each plot has two parts: upper part is a graph of LRR and lower part is a graph of BAF.

```
> snp.not.ok <- snpAnnot$snpID[snp.exclude]
> anomStatsPlot(blData, genoData, anom.stats=stats[1,],
+   snp.ineligible=snp.not.ok, centromere=centromeres, cex.leg=1)
```

anom 1 – scan 286 – M – chrom 22 – BAF

green=AA, orange=AB, purple=BB, pink=ineligible, black=other missing
horiz solid red = non–anom median, horiz dashed red = anom median

7.4 Identify low quality samples

To identify low quality samples, one measure we use is the standard deviation of BAF and LRR. BAF results were found previously, now we find results for LRR. Unlike for BAF, all genotypes are included.

```
> lrr.sd <- sdByScanChromWindow(blData, var="LogRRatio", incl.hom=TRUE)
> med.lrr.sd <- medianSdOverAutosomes(lrr.sd)
```

We also need the number of segments found using circular binary segmentation in anomaly detection.

```
> baf.seg.info <- baf.anom$seg.info
> loh.seg.info <- loh.anom$base.info[,c("scanID", "chromosome", "num.segs")]
```

71

We identify low quality samples separately for BAF and LOH, using different threshold parameters. A SnpAnnotationDataFrame with an "eligible" column is required. BAF detected anomalies for low quality BAF samples tend to have higher false positive rate. LOH detected anomalies for low quality LOH samples tend to have higher false positive rate.

```
> snpAnnot$eligible <- !snp.exclude
> baf.low.qual <- anomIdentifyLowQuality(snpAnnot, med.baf.sd, baf.seg.info,
+   sd.thresh=0.1, sng.seg.thresh=0.0008, auto.seg.thresh=0.0001)
> loh.low.qual <- anomIdentifyLowQuality(snpAnnot, med.lrr.sd, loh.seg.info,
+   sd.thresh=0.25, sng.seg.thresh=0.0048, auto.seg.thresh=0.0006)
```

Close the `IntensityData` and `GenotypeData` objects.

```
> close(blData)
> close(genoData)
```

7.5 Filter anomalies

We can set genotypes in anomaly regions to missing for future analyses (such as Hardy-Weinberg equilibrium and association tests). We use the function `setMissingGenotypes` to create a new GDS file with anomaly regions set to `NA`. We recommend inspecting the plots from `anomStatsPlot` for large anomalies (e.g., > 5 Mb) to identify those anomalies that case genotyping errors. We can also exclude certain samples, such as duplicates, low quality samples, and samples with unresolved identity issues.

```
> # anomalies to filter
> anom.filt <- stats[,c("scanID", "chromosome", "left.base", "right.base")]
> # whole.chrom column is required and can be used for sex chromosome
> #   anomalies such as XXX
> anom.filt$whole.chrom <- FALSE
> # select unique subjects
> subj <- scanAnnot$scanID[!scanAnnot$duplicated]
> subj.filt.file <- "subj_filt.gds"
> setMissingGenotypes(genofile, subj.filt.file, anom.filt,
+                     file.type="gds", sample.include=subj, verbose=FALSE)
> (gds <- GdsGenotypeReader(subj.filt.file))

File: /tmp/Rtmp3T8PWN/Rbuild20974b120b55/GWASTools/vignettes/subj_filt.gds (60.7K)
+    [  ]
|--+ sample.id   { Int32 43 LZMA_ra(87.2%), 157B }
|--+ snp.id    { Int32 3300 LZMA_ra(25.9%), 3.3K }
|--+ snp.chromosome   { UInt8 3300 LZMA_ra(3.45%), 121B } *
|--+ snp.position   { Int32 3300 LZMA_ra(68.4%), 8.8K }
|--+ snp.rs.id   { Str8 3300 LZMA_ra(33.6%), 11.0K }
|--+ snp.allele    { Str8 3300 LZMA_ra(8.83%), 1.1K }
\--+ genotype    { Bit2 3300x43, 34.6K } *

> close(gds)
```

72

8   SNP Quality Checks

This step finds SNPs that may not be suitable for use in GWAS studies due to genotyping artifacts. Three methods are used to look at the genotyping error rates for each SNP: duplicate sample discordance, Mendelian error rates and deviation from Hardy-Weinberg equilibrium.

8.1   Duplicate Sample Discordance

This step calculates the discordance of genotype calls between samples that are duplicates. Genotype discordance is evaluated by comparing the genotypes of samples that were genotyped more than once. We can examine the discordance rate with respect to samples or SNPs. The discordance rate for a pair of samples is the fraction of genotype calls that differ over all SNPs for which both calls are non-missing. The discordance rate for a SNP is the number of calls that differ divided by the number of duplicate pairs in which both calls are non-missing.

Keep the samples with a low enough value for the missing call rate, `missing.e1`. The threshold chosen here is 0.05.

```
> scan.excl <- scanAnnot$scanID[scanAnnot$missing.e1 >= 0.05]
> length(scan.excl)

[1] 0
```

We make a vector of SNP `snpIDs` with `missing.n1 = 1` to exclude from the comparison. We then call the `duplicateDiscordance` function and save the output file. This function finds subjectIDs for which there is more than one scanID. To look at the discordance results, we will calculate the percentage value and look at the summary of the values for each of the duplicate pairs. We will plot the rates color coded by continental ancestry, since experience has shown the values often differ based upon the population group.

```
> snp.excl <- snpAnnot$snpID[snpAnnot$missing.n1 == 1]
> length(snp.excl)

[1] 151

> genofile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
> genoGDS <- GdsGenotypeReader(genofile)
> genoData <- GenotypeData(genoGDS, snpAnnot=snpAnnot, scanAnnot=scanAnnot)
> dupdisc <- duplicateDiscordance(genoData, subjName.col="subjectID",
+   scan.exclude=scan.excl, snp.exclude=snp.excl)
> names(dupdisc)

[1] "discordance.by.snp"     "discordance.by.subject" "correlation.by.subject"

> head(dupdisc$discordance.by.snp)

  snpID discordant npair n.disc.subj discord.rate
1 999465          0    33           0            0
2 999493          0    31           0            0
```

```
3 999512              0    34           0            0
4 999561              0    34           0            0
5 999567              0    34           0            0
6 999569              0    34           0            0

> length(dupdisc$discordance.by.subject)

[1] 34

> dupdisc$discordance.by.subject[[2]]

               282           283
282 0.0000000000 0.0003265839
283 0.0003265839 0.0000000000

> # each entry is a 2x2 matrix, but only one value of each
> # is important since these are all pairs
> npair <- length(dupdisc$discordance.by.subject)
> disc.subj <- rep(NA, npair)
> subjID <- rep(NA, npair)
> race <- rep(NA, npair)
> for (i in 1:npair) {
+   disc.subj[i] <- dupdisc$discordance.by.subject[[i]][1,2]
+   subjID[i] <- names(dupdisc$discordance.by.subject)[i]
+   race[i] <- scanAnnot$race[scanAnnot$subjectID == subjID[i]][1]
+ }
> dat <- data.frame(subjID=subjID, disc=disc.subj, pop=race,
+                   stringsAsFactors=FALSE)
> summary(dat$disc)

     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.0000000 0.0000000 0.0000000 0.0002660 0.0003225 0.0019828

> # Assign colors for the duplicate samples based on population group.
> dat$col <- NA
> dat$col[dat$pop == "CEU"] <- "blue"
> dat$col[dat$pop == "YRI"] <- "red"
> dat <- dat[order(dat$disc),]
> dat$rank <- 1:npair

> # Plot the sample discordance rates color coded by race.
> plot(dat$disc, dat$rank, col=dat$col, ylab="rank",
+   xlab="Discordance rate between duplicate samples",
+   main="Duplicate Sample Discordance by Continental Ancestry")
> legend("bottomright", unique(dat$pop), pch=rep(1,2), col=unique(dat$col))
```
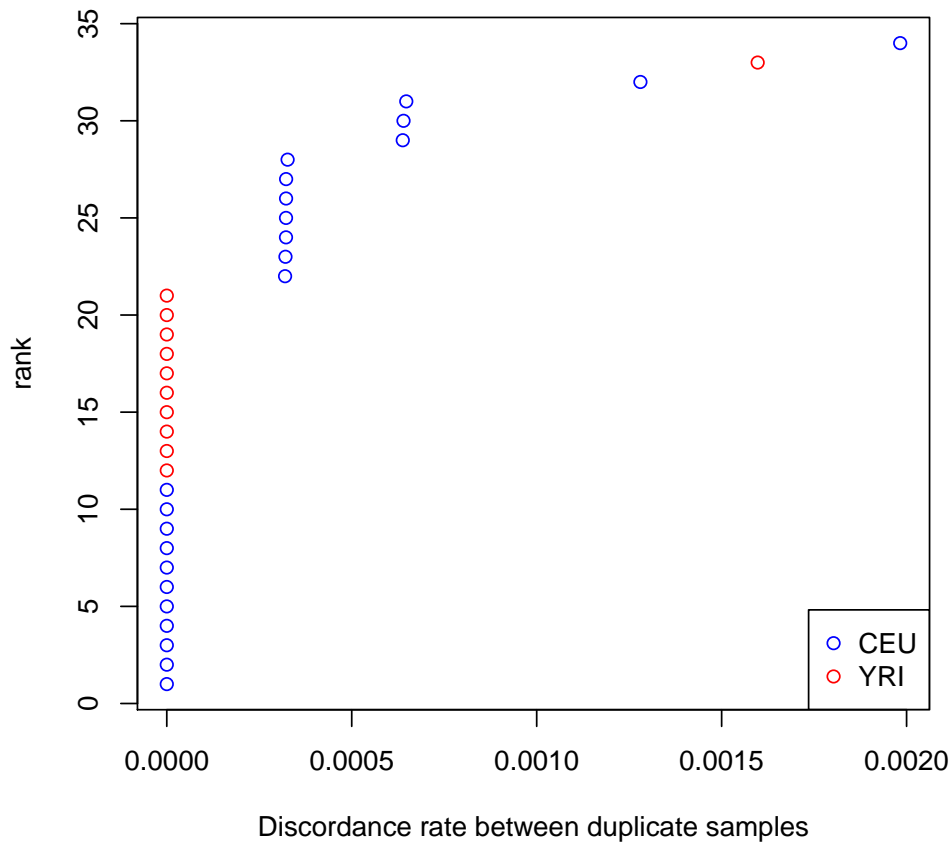
Duplicate Sample Discordance by Continental Ancestry

Discordance rate between duplicate samples

Genotyping error rates can be estimated from duplicate discordance rates. The genotype at any SNP may be called correctly, or miscalled as either of the other two genotypes. If α and β are the two error rates, the probability that duplicate genotyping instances of the same participant will give a discordant genotype is $2[(1 - \alpha - \beta)(\alpha + \beta) + \alpha\beta]$. When α and β are very small, this is approximately $2(\alpha + \beta)$ or twice the total error rate. Potentially, each true genotype has different error rates (i.e. three α and three β parameters), but here we assume they are the same. A rough estimate of the mean error rate is half the median discordance rate over all sample pairs.

Duplicate discordance estimates for individual SNPs can be used as a SNP quality filter. The challenge here is to find a level of discordance that would eliminate a large fraction of SNPs with high error rates, while retaining a large fraction with low error rates. The probability of observing $> x$ discordant genotypes in a total of n pairs of duplicates can be calculated using the binomial distribution.

```
> duplicateDiscordanceProbability(npair)

       error=1e-05   error=1e-04   error=0.001    error=0.01
dis>0 6.797706e-04  6.777101e-03  6.575540e-02  4.942376e-01
dis>1 2.243009e-07  2.234113e-05  2.147291e-03  1.459858e-01
```

75

```
dis>2 4.784862e-11 4.763922e-08 4.559737e-05 2.961468e-02
dis>3 7.366668e-15 7.382418e-11 7.051524e-07 4.476031e-03
dis>4 0.000000e+00 8.858958e-14 8.452530e-09 5.304432e-04
dis>5 0.000000e+00 9.581360e-17 8.167288e-11 5.100698e-05
dis>6 0.000000e+00 1.025893e-17 6.533692e-13 4.077516e-06
dis>7 0.000000e+00 1.019048e-17 4.402201e-15 2.758531e-07
```

8.2 Mendelian Error Checking

This step calculates and examines the Mendelian error rates. Mendelian errors are detected in parent-offspring trios or pairs as offspring genotypes that are inconsistent with Mendelian inheritance. We use the `mendelErr` function to calculate a Mendelian error rate per SNP. Lastly some checks are done on Mendelian error rates per family.

To call the Mendelian error checking function, we first must create a `mendelList` object. We will call `mendelList` that creates a list of trios, checking for any sex inconsistencies among annotated father and mother samples. Then, `mendelListAsDataFrame` puts this list into a data frame for easier checking. Finally, we can call the `mendelErr` function to find the Mendelian errors for SNPs with `missing.n1` less than 0.05.

```
> men.list <- with(pData(scanAnnot), mendelList(family, subjectID,
+   father, mother, sex, scanID))
> res <- mendelListAsDataFrame(men.list)
> head(res)

  offspring father mother
1       329    331    332
2       329    331    333
3       330    331    332
4       330    331    333
5       334    336    338
6       334    336    339

> dim(res)

[1] 82  3

> # Only want to use SNPs with missing.n1 < 0.05
> snp.excl <- snpAnnot$snpID[snpAnnot$missing.n1 >= 0.05]
> length(snp.excl)

[1] 206

> mend <- mendelErr(genoData, men.list, snp.exclude=snp.excl)
> names(mend)

[1] "trios"     "all.trios" "snp"

> head(mend$trios)
```

```
    fam.id  child.id Men.err.cnt Men.cnt mtDNA.err mtDNA.cnt chr1 chr2 chr3 chr4
1       4 200047857           1  3006.5         0      87.0    0    0    0    0
2       5 200102386           2  3003.0         0      86.5    0    0    0    0
3       9 200066330           1  3003.5         0      86.5    0    0    0    0
4      12 200013233           0  2999.5         0      82.0    0    0    0    0
5      28 200034659           0  3002.5         0      64.0    0    0    0    0
6    1334 200016815           3  2988.5         0      87.0    0    0    0    0
  chr5 chr6 chr7 chr8 chr9 chr10 chr11 chr12 chr13 chr14 chr15 chr16 chr17
1    0    0    0    0    0     0     0     0     0     0     0     0     0
2    0    0    0    0    0     0     0     0     0     0     0     0     0
3    0    0    0    0    0     0     0     0     0     0     0     0     0
4    0    0    0    0    0     0     0     0     0     0     0     0     0
5    0    0    0    0    0     0     0     0     0     0     0     0     0
6    0    0    0    0    0     0     0     0     0     0     0     0     0
  chr18 chr19 chr20 chr21 chr22 chrX chrXY chrY
1     0     0     0     0   0.0  0.0     1    0
2     0     0     0     0   2.0  0.0     0    0
3     0     0     0     0   1.0  0.0     0    0
4     0     0     0     0   0.0  0.0     0    0
5     0     0     0     0   0.0  0.0     0    0
6     0     0     0     1   1.5  0.5     0    0

> names(mend$snp)

[1] "check.cnt" "error.cnt"
```

Mendelian Errors per SNP

The Mendelian error rate is calculated for each SNP by dividing the number of errors per SNP for all trios by the number of trios used in the error checking.

```
> # Calculate the error rate
> err <- mend$snp$error.cnt / mend$snp$check.cnt
> table(err == 0, useNA="ifany")

FALSE   TRUE
   11   3083

> plot(err, rank(err), xlab="Error Rate (fraction)",
+   ylab="rank", main="Mendelian Error Rate per SNP, ranked")
```

Mendelian Error Rate per SNP, ranked
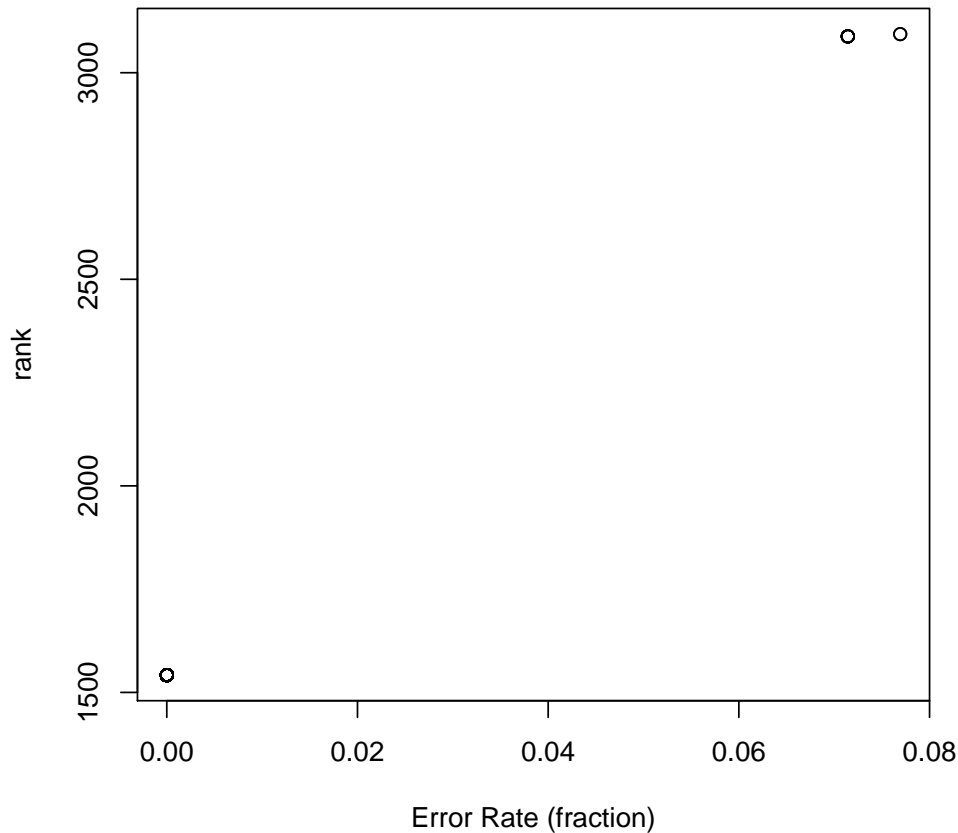
Next we will look at the Mendelian error rates among the trios we have in the HapMap data. Looking at the summary of the number of families with at least one error over all SNPs, we can see the maximum number of errors per SNP. Next, we can look at subsets of SNPs with greater than 0 and 1 errors per SNP. Finally, for those SNPs that have valid trios to detect errors, we get the fraction of SNPs with no errors.

```
> fam <- mend$snp$error.cnt
> n <- mend$snp$check.cnt
> summary(fam)

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.000000 0.000000 0.000000 0.003555 0.000000 1.000000

> # SNPs with errors
> length(fam[n > 0 & fam > 0])

[1] 11

> # SNPs for which more than one family has an error
> length(fam[n > 0 & fam > 1])
```

```
[1] 0

> # Get the SNPs with valid trios for error detection
> val <- length(fam[n > 0])
> noerr <- length(fam[n > 0 & fam == 0])
> # Divide to get fraction with no errors
> noerr / val

[1] 0.9964447
```

We add the Mendelian error values to the SNP annotation. The number of families with at least one error per SNP, `mend$snp$error.cnt`, gets saved as `mendel.err.count`. The number of valid families for checking, `mend$snp$check.cnt`, gets saved as `mendel.err.sampsize`.

```
> snp.sel <- match(names(mend$snp$error.cnt), snpAnnot$snpID)
> snpAnnot$mendel.err.count[snp.sel] <- mend$snp$error.cnt
> snpAnnot$mendel.err.sampsize[snp.sel] <- mend$snp$check.cnt
> allequal(snpAnnot$snpID, sort(snpAnnot$snpID))

[1] TRUE

> # The high number of NA values is due to the filtering out of SNPs
> #    before the Mendelian error rate calculation
> sum(is.na(snpAnnot$mendel.err.count))

[1] 206

> sum(is.na(snpAnnot$mendel.err.sampsize))

[1] 206

> varMetadata(snpAnnot)["mendel.err.count", "labelDescription"] <-
+   paste("number of Mendelian errors detected in trios averaged over",
+         "multiple combinations of replicate genotyping instances")
> varMetadata(snpAnnot)["mendel.err.sampsize", "labelDescription"] <-
+   "number of opportunities to detect Mendelian error in trios"
```

To further investigate SNPs with a high Mendelian error rate, we will make genotype cluster plots for the 9 SNPs with the highest Mendelian error rate. green indicates a sample with an "AA" genotype, orange is an "AB" genotype and purple is a "BB" genotype. The black X marks indicate a sample with a missing genotype for that SNP. We expect the plots to lack defined genotype clusters, leading to a poor call rate.

```
> # Get a vector of SNPs to check
> snp <- pData(snpAnnot)
> snp$err.rate <- snp$mendel.err.count /
+   snp$mendel.err.sampsize
> snp <- snp[order(snp$err.rate, decreasing=TRUE),]
```

79

```
> snp <- snp[1:9,]
> xyfile <- system.file("extdata", "illumina_qxy.gds", package="GWASdata")
> xyGDS <- GdsIntensityReader(xyfile)
> xyData <- IntensityData(xyGDS, snpAnnot=snpAnnot, scanAnnot=scanAnnot)
> pdf(file="DataCleaning-mendel.pdf")
> par(mfrow = c(3,3))
> mtxt <- paste("SNP", snp$rsID, "\nMendelian Error Rate",
+     format(snp$err.rate, digits=5))
> genoClusterPlot(xyData, genoData, snpID=snp$snpID, main.txt=mtxt,
+                 cex.main=0.9)
> dev.off()
> close(xyData)
```
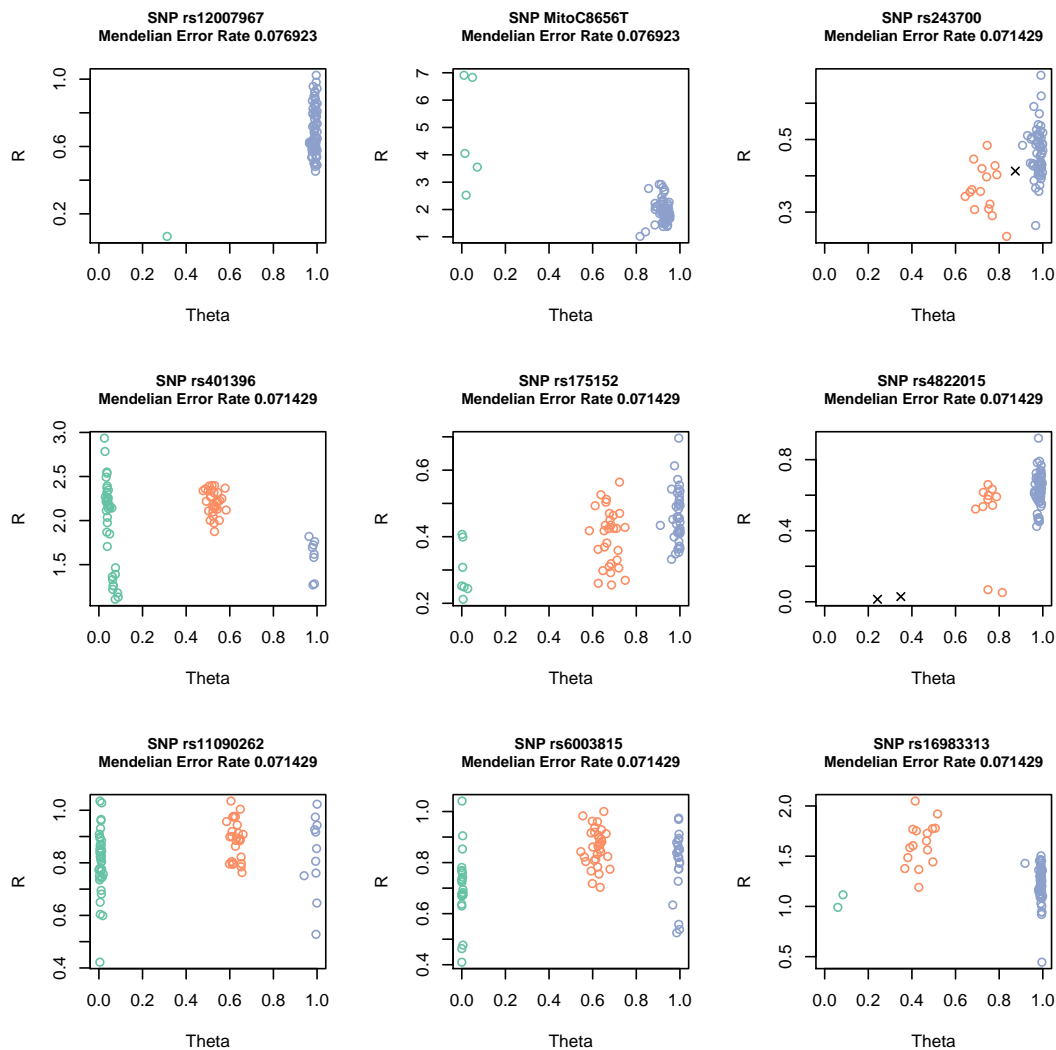
Mendelian Errors per Family

This section does some analyses on the Mendelian Errors for each family (trio). The variable `all.trios` contains results of all combinations of duplicate samples. The variable `trios` contains the averages of unique trios (averages of duplicates from `all.trios`).

```
> # Calculate the fraction of SNPs with an error for each trio
> trios <- mend$trios
> trios$Mend.err <- trios$Men.err.cnt/trios$Men.cnt
> summary(trios$Mend.err)

     Min.    1st Qu.    Median      Mean   3rd Qu.      Max.
0.0000000 0.0000000 0.0000000 0.0001909 0.0002916 0.0010038

> # Start by pulling out the vectors needed from `trios'
> tmp <- trios[, c("fam.id", "Mend.err")]; dim(tmp)

[1] 14  2

> # Change fam.id to match the sample annotation column name
> names(tmp) <- c("family", "Mend.err.rate.fam")
> # Merge the variables into the sample annotation file
> scanAnnot$mend.err.rate.fam <- NA
> for (i in 1:nrow(tmp)) {
+   ind <- which(is.element(scanAnnot$family, tmp$family[i]))
+   scanAnnot$mend.err.rate.fam[ind] <- tmp$Mend.err.rate.fam[i]
+ }
> head(scanAnnot$mend.err.rate.fam)

[1] 0 0 0 0 0 0

> varMetadata(scanAnnot)["mend.err.rate.fam", "labelDescription"] <-
+   "Mendelian error rate per family"
```

The Mendelian error rate per family, broken up by continental ancestry, could illuminate issues with SNPs that may not be accurately called across all ethnicities for the minor allele. We will plot the Mendelian error rate per family, color coded by population group. The error rates are higher for the YRI families as a whole, which is expected due to the higher level of genetic diversity.

```
> # Get the families that have non-NA values for the family
> #    Mendelian error rate
> fams <- pData(scanAnnot)[!is.na(scanAnnot$mend.err.rate.fam) &
+   !duplicated(scanAnnot$family), c("family",
+   "mend.err.rate.fam", "race")]
> dim(fams)

[1] 12  3
```

```
> table(fams$race, useNA="ifany")

CEU YRI
  7   5


> # Assign colors for the different ethnicities in these families
> pcol <- rep(NA, nrow(fams))
> pcol[fams$race == "CEU"] <- "blue"
> pcol[fams$race == "YRI"] <- "red"

> plot(fams$mend.err.rate.fam*100, rank(fams$mend.err.rate.fam),
+    main="Mendelian Error rate per Family, ranked",
+    xlab="Mendelian error rate per family (percent)",
+    ylab="rank", col=pcol)
> legend("bottomright", c("CEU", "YRI"), pch=c(1,1), col=c("blue", "red"))
```

Mendelian Error rate per Family, ranked
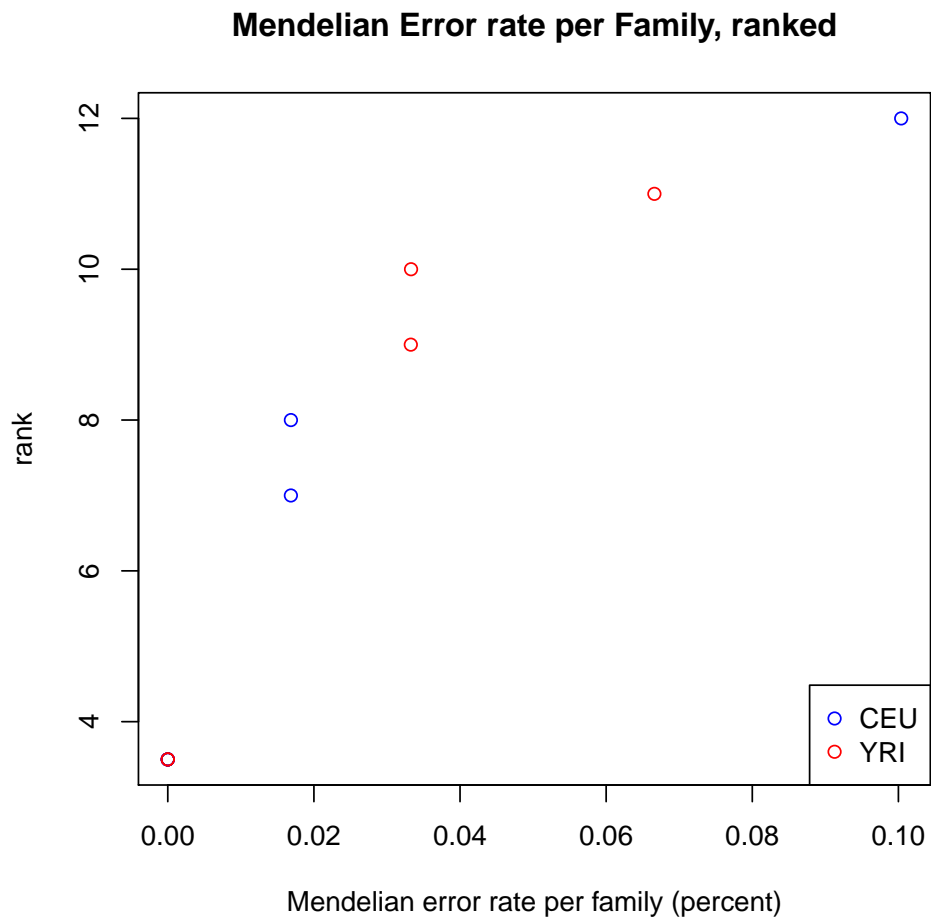
8.3 Hardy-Weinberg Equilibrium Testing

This section uses Fisher's exact test to examine each SNP for departure from Hardy-Weinberg Equilibrium. For each SNP, p-values are obtained; those SNPs with extremely low values will be considered for filtering. QQ-plots of the p-values are made for both the autosomes and X chromosome.

To run the Hardy-Weinberg test, we will filter out duplicates and non-founders. We will run `exactHWE` for the samples with European continental ancestry only, although the process is just the same for all population groups. The X chromosome must be run separately from the autosomes since it filters out males.

```
> head(pData(scanAnnot)[,c("father", "mother")])

    father     mother
1         0          0
2         0          0
3 200099417 200191449
4 200099417 200191449
5         0          0
6         0          0

> nonfounders <- scanAnnot$father != 0 &
+                scanAnnot$mother != 0
> table(nonfounders)

nonfounders
FALSE   TRUE
   51     26

> scan.excl <- scanAnnot$scanID[scanAnnot$race != "CEU" |
+    nonfounders | scanAnnot$duplicated]
> length(scan.excl)

[1] 60

> chr <- getChromosome(genoData)
> auto <- range(which(chr %in% 1:22))
> X <- range(which(chr == 23))
> hwe <- exactHWE(genoData, scan.exclude=scan.excl, snpStart=auto[1], snpEnd=auto[2])
> hweX <- exactHWE(genoData, scan.exclude=scan.excl, snpStart=X[1], snpEnd=X[2])
> hwe <- rbind(hwe, hweX)
> close(genoData)
```

We will look at the values calculated from the function call to `exactHWE`, which include p-values, minor allele frequency, and genotype counts for each SNP on each of the chromosome types.

```
> names(hwe)
```

```
[1] "snpID"          "chr"           "nAA"          "nAB"          "nBB"
[6] "MAF"            "minor.allele" "f"            "pval"

> dim(hwe)

[1] 3000    9

> # Check on sample sizes for autosomes and X chromosome
> hwe$N <- hwe$nAA + hwe$nAB + hwe$nBB
> summary(hwe$N[is.element(hwe$chr,1:22)])

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00   17.00   17.00   16.93   17.00   17.00

> summary(hwe$N[is.element(hwe$chr,23)])

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   8.000   8.000   7.597   8.000   8.000

> hwe$pval[1:10]

 [1]          NA          NA 1.0000000 1.0000000 1.0000000 0.5372636 0.1788856
 [8] 1.0000000 1.0000000 1.0000000

> sum(is.na(hwe$pval[hwe$chr == 23])) # X

[1] 300

> hwe$MAF[1:10]

 [1]         NaN 0.00000000 0.13333333 0.26470588 0.05882353 0.20588235
 [7] 0.11764706 0.23529412 0.35294118 0.35294118

> hwe[1:3, c("nAA", "nAB", "nBB")]

  nAA nAB nBB
1   0   0   0
2   0   0  17
3   0   4  11
```

Next we want to estimate the inbreeding coefficient per SNP calculated using the minor allele frequencies and the total sample number count. A histogram shows the distribution is centered around 0, which indicates there is most likely no significant population substructure.

```
> summary(hwe$f)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
-1.0000 -0.1806 -0.0667 -0.0405  0.1005  1.0000     515
```

```
> hist(hwe$f, main="Histogram of the Inbreeding Coefficient
+   For CEU Samples", xlab="Inbreeding Coefficient")
```

Histogram of the Inbreeding Coefficient
For CEU Samples
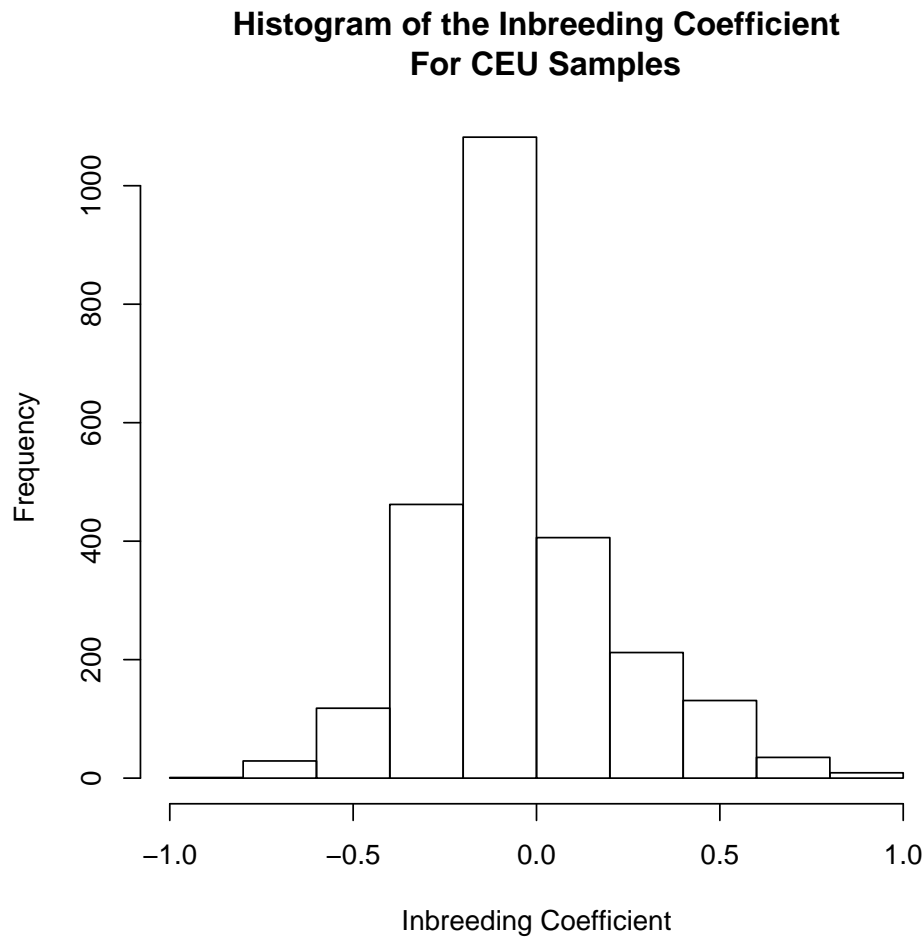

```
> # Check the MAF of those SNPs with f=1
> chkf <- hwe[!is.na(hwe$f) & hwe$f==1,]; dim(chkf)

[1]  9 10

> summary(chkf$MAF)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.05882 0.05882 0.05882 0.10212 0.12500 0.25000
```

To see at what value the SNPs begin to deviate from the Hardy-Weinberg expected values, we will make QQ-plots that exclude SNPs where MAF = 0. We plot of the observed p-values vs. the expected p-values for the autosomes and X chromosome separately by calling the function qqPlot.

```
> hwe.0 <- hwe[hwe$MAF > 0,]; dim(hwe.0)
```

```
[1] 2539    10

> # Only keep the autosomal SNPs for first plot
> pval <- hwe.0$pval[is.element(hwe.0$chr, 1:22)]
> length(pval)

[1] 1785

> pval <- pval[!is.na(pval)]
> length(pval)

[1] 1785

> # X chromosome SNPs for plot 2
> pval.x <- hwe.0$pval[is.element(hwe.0$chr, 23)]
> length(pval.x)

[1] 700

> pval.x <- pval.x[!is.na(pval.x)]
> length(pval.x)

[1] 700

> pdf(file = "DataCleaning-hwe.pdf")
> par(mfrow=c(2,2))
> qqPlot(pval=pval, truncate = FALSE, main="Autosomes, all")
> qqPlot(pval=pval, truncate = TRUE, main="Autosomes, truncated")
> qqPlot(pval=pval.x, truncate = FALSE, main="X chromosome, all")
> qqPlot(pval=pval.x, truncate = TRUE, main="X chromosome, truncated")
> dev.off()
```

Autosomes, all

Autosomes, truncated

X chromosome, all

X chromosome, truncated
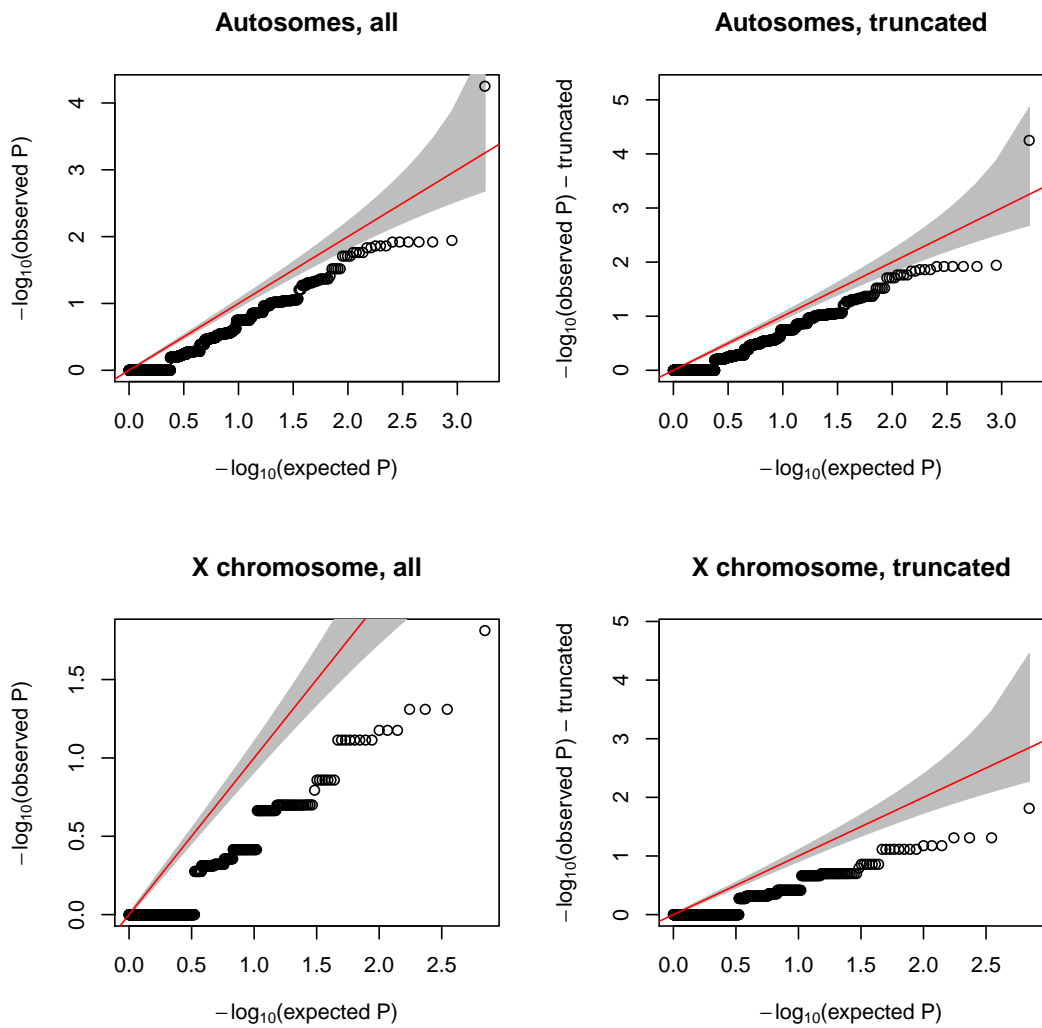
We plot the p-values against MAF for all SNPs with MAF greater than zero.

```
> plot(hwe.0$MAF, -log10(hwe.0$pval),
+   xlab="Minor Allele Frequency", ylab="-log(p-value)",
+   main="Minor Allele Frequency vs\nP-value")
```

Minor Allele Frequency vs
P−value

−log(p−value) *(y-axis)*

Minor Allele Frequency *(x-axis)*
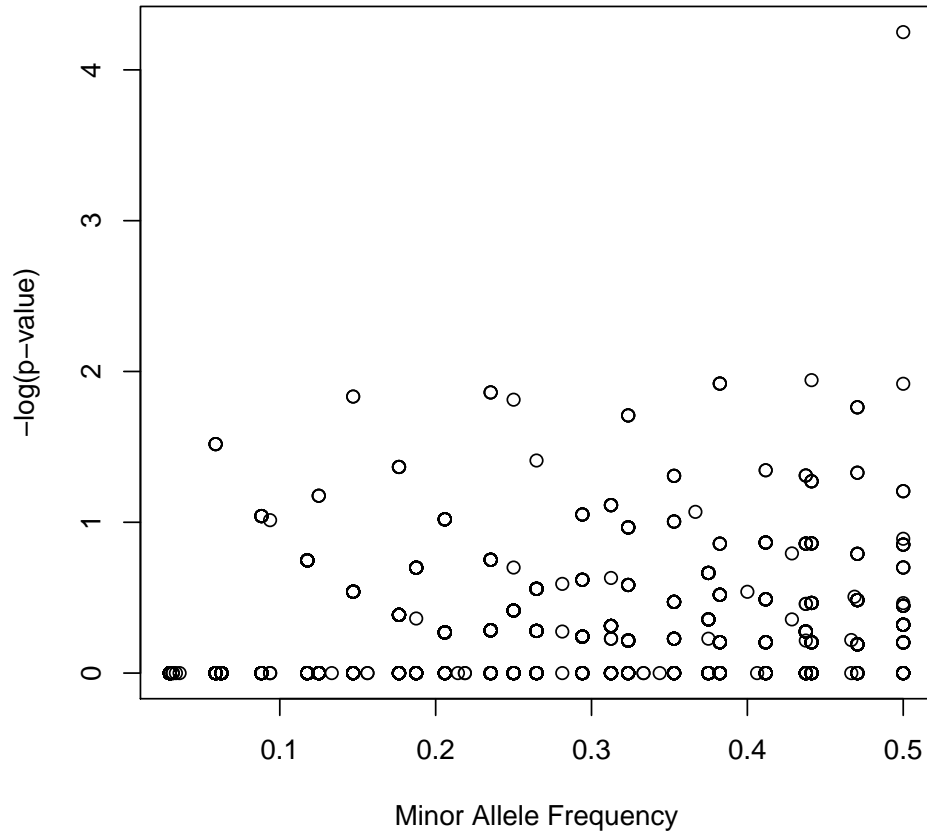
9   Preliminary Association Tests

The final step in the data cleaning process is to perform preliminary association tests. This step creates and examines QQ, 'Manhattan' signal, and genotype cluster plots. If significant SNPs appear as a result of the association test, SNP cluster plots must be examined to determine if the association is driven from a poorly clustering SNP. Note that HapMap data do not come with phenotypic outcomes, thus, for purposes of the tutorial we use simulated binary outcomes instead. The tests conducted are usually logistic regression based tests; the samples are filtered by quality criteria and only unrelated subjects are included. In the code below we do not include any covariates in the logistic regression as these data are not part of a case control study. For data in the GENEVA project and other GWA studies we discuss which variables should be considered for inclusion as covariates in the preliminary association tests. The determination is made by analyzing a model including these covariates but without genotypes; covariates with significant effects are then included in the final model.

9.1   Association Test

To run the association test, we call the function `assocRegression`. (For survival analysis, the function `assocCoxPH` may be used.) We will use a logistic regression model with status as the outcome, and sex and the first principal component as covariates.

We will use the filtered GDS file with unique subjects only that we made in section 7.5. Here we use all unique subjects, but we would use the argument `scan.exclude` for those samples we wish to filter out for the association test (such as low-quality or related samples). Typically, we do not filter out SNPs for the association test – we run all SNPs and then filter the results. The omission of filters may cause some SNPs to return significant p-values for association. In this example, we will use the `snpStart` and `snpEnd` arguments to select a few SNPs on each chromosome.

```
> genoGDS <- GdsGenotypeReader(subj.filt.file)
> subjAnnot <- scanAnnot[scanAnnot$scanID %in% getScanID(genoGDS),]
> subjAnnot$sex <- as.factor(subjAnnot$sex)
> subjAnnot$EV1 <- pca$eigenvect[match(subjAnnot$scanID, pca$sample.id), 1]
> genoData <- GenotypeData(genoGDS, scanAnnot=subjAnnot)
> chr <- getChromosome(genoData)
> assoc.list <- lapply(unique(chr), function(x) {
+     ## Y chromsome only includes males, cannot have sex as a covariate
+     covar <- ifelse(x == 25, "EV1", c("sex", "EV1"))
+     start <- which(chr == x)[1]
+     assocRegression(genoData, outcome="status", covar=covar, model.type="logistic",
+                     snpStart=start, snpEnd=start+50)
+ })
> assoc <- do.call(rbind, assoc.list)
> close(genoData)
```

After running the association test on the selected subset of SNPs and samples we must analyze the results to determine if any probes with significant p-values are spurious or truly associated with the phenotype of interest. Quantile-quantile, 'Manhattan' and SNP cluster plots will all be used to further understand those probes with significant p-values.

9.2 QQ Plots

We create QQ plots of the ordered Wald test p-values versus the ordered expected p-values. Given that the samples were split randomly between cases and controls, not surprisingly there are no outliers visible in the QQ plot.

```
> qqPlot(pval=assoc$Wald.pval,
+   truncate=TRUE, main="QQ Plot of Wald Test p-values")
```

QQ Plot of Wald Test p−values
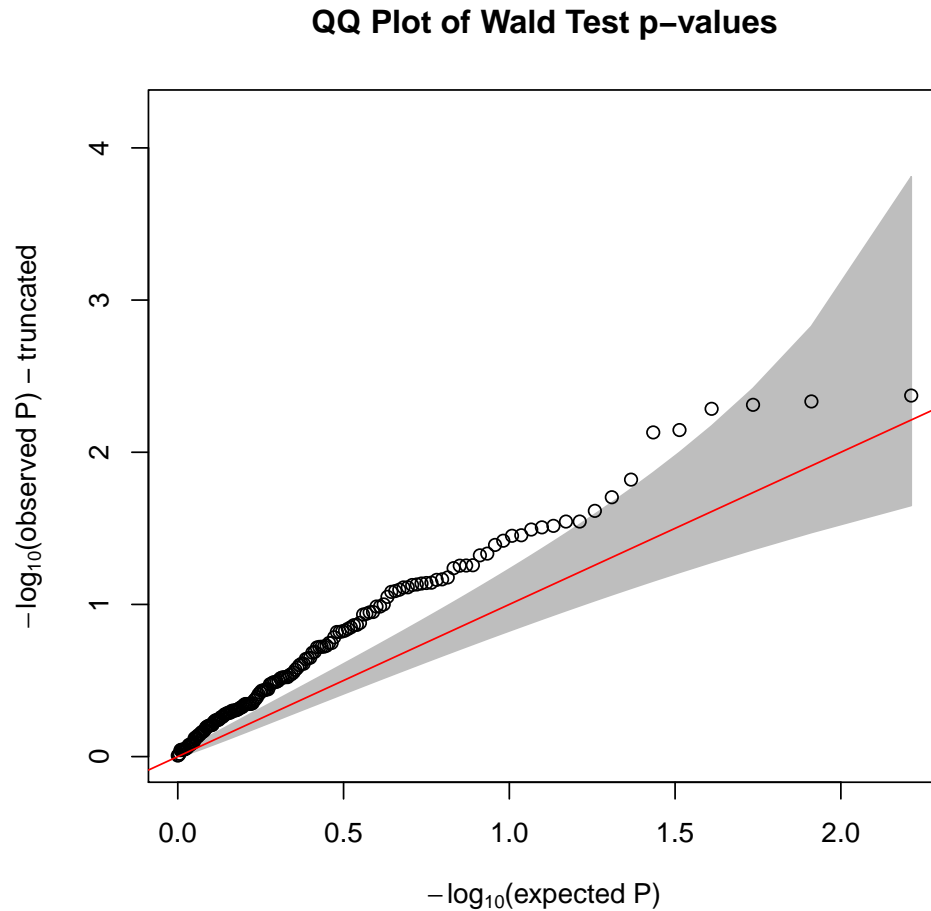
9.3 "Manhattan" Plots of the P-Values

To create the 'Manhattan' plots, we will call the function `manhattanPlot`. We take the negative log transformation of the p-values and plot them for each probe.

```
> chrom <- getChromosome(snpAnnot, char=TRUE)
> snp.sel <- getSnpID(snpAnnot) %in% assoc$snpID
> manhattanPlot(assoc$Wald.pval, chromosome=chrom[snp.sel])
```
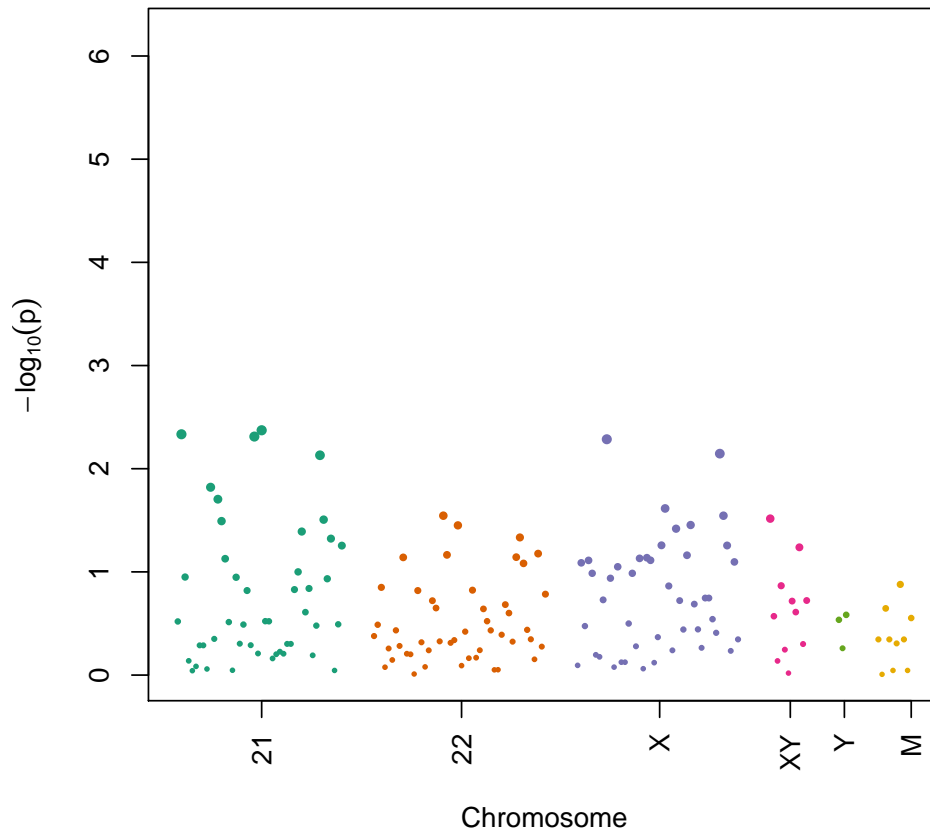
9.4 SNP Cluster Plots

Next, we will create SNP cluster plots for the probes with significant p-values. It is important to examine cluster plots of all top hits, as poor clusters not picked up by other quality checking steps may still show up as having low p-values. We plot SNPs with the 9 most significant p-values from the Wald test.

```
> # Identify SNPs with lowest p-values
> snp <- pData(snpAnnot)[snp.sel, c("snpID", "rsID")]
> snp$pval <- assoc$Wald.pval
> snp <- snp[order(snp$pval),]
> snp <- snp[1:9,]
> xyfile <- system.file("extdata", "illumina_qxy.gds", package="GWASdata")
> xyGDS <- GdsIntensityReader(xyfile)
> xyData <- IntensityData(xyGDS, snpAnnot=snpAnnot, scanAnnot=scanAnnot)
> genofile <- system.file("extdata", "illumina_geno.gds", package="GWASdata")
> genoGDS <- GdsGenotypeReader(genofile)
```

```
> genoData <- GenotypeData(genoGDS, snpAnnot=snpAnnot, scanAnnot=scanAnnot)
> pdf(file="DataCleaning-cluster.pdf")
> par(mfrow = c(3,3))
> mtxt <- paste("SNP", snp$rsID, "\np =", format(snp$pval, digits=4))
> genoClusterPlot(xyData, genoData, snpID=snp$snpID, main.txt=mtxt)
> dev.off()
> close(xyData)
> close(genoData)
```
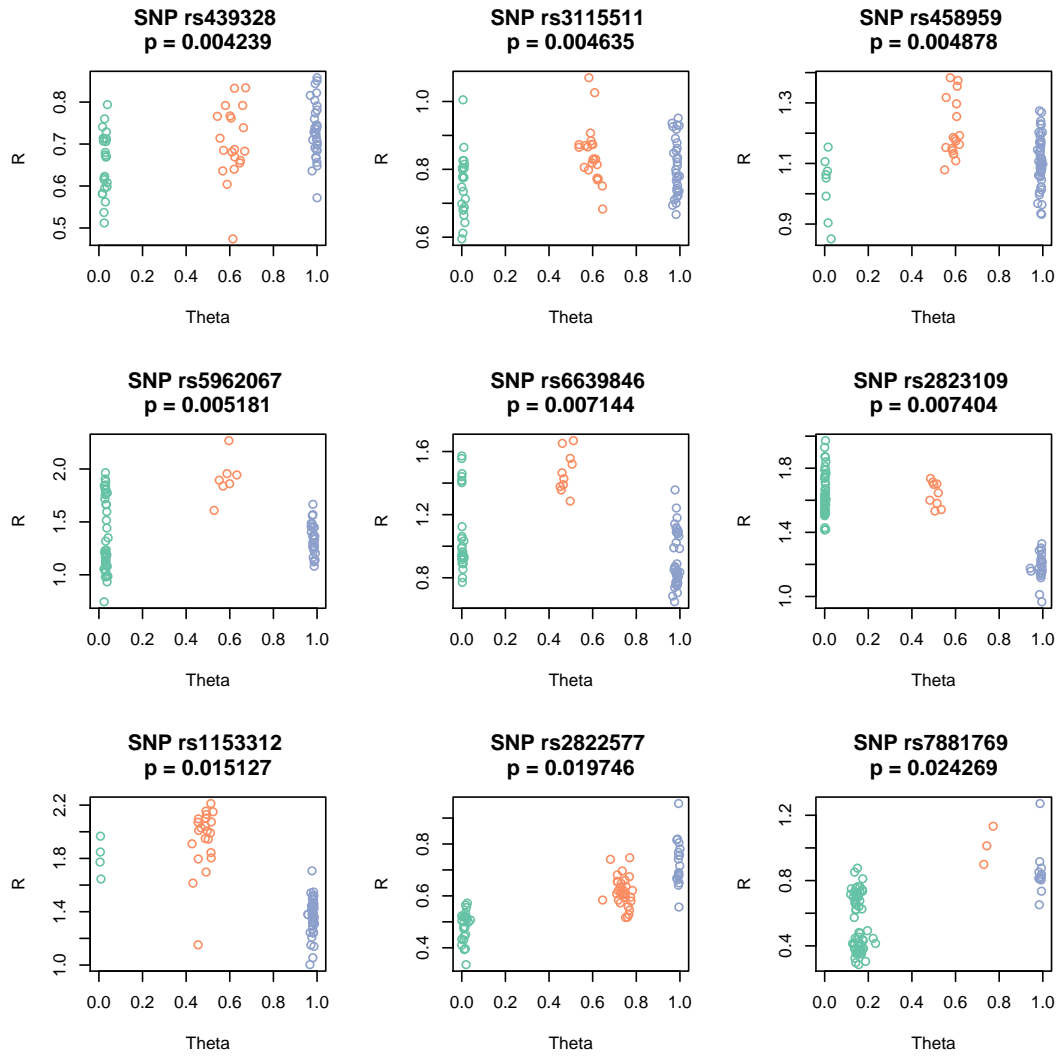

These cluster plots look like high-quality SNPs with well-defined clusters (orange=AA, green=AB, fuschia=BB).

10 Acknowledgements