GWAS Data Cleaning

GENEVA Coordinating Center
Department of Biostatistics
University of Washington

October 31, 2011

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 these vignettes 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 `SNPAnnotationDataFrame` 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 AnnotationDataFrame, 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 are stored in the Network Common Data Format (NetCDF). NetCDF is a set of software libraries and machine-independent data formats, designed specifically for large sets of array-oriented scientific data, and its use is widespread in those sciences that require very large data sets. It is maintained by the Unidata program at the University Corporation for Atmospheric Research (UCAR). This portable binary file format is convenient since it allows for efficient multi-dimensional arrayed data (although we only use up to two dimensions).

In the GWASTools package, access to the NetCDF files is provided by the `NcdfGenotypeReader` and `NcdfIntensityReader` classes. These classes are built on top of the ncdf package and provide access to a standard set of variables defined for GWAS data. Additionally, the NetCDF files and SNP and scan annotation can be linked through the `GenotypeData` and `IntensityData` classes, which have slots for a `NcdfGenotypeReader` (or `NcdfIntensityReader`) 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 NetCDF 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)
- `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 NetCDF files, so in order to link the SNP annotation with the NetCDF 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(affy_snp_annot)
> # Create a SnpAnnotationDataFrame
> snpAnnot <- SnpAnnotationDataFrame(affy_snp_annot)
> # names of columns
> varLabels(snpAnnot)

[1] "snpID"      "chromosome" "position"   "rsID"       "probeID"
[6] "missing.n1"

> # data
> head(pData(snpAnnot))

        snpID chromosome position      rsID       probeID missing.n1
554407 869828         21 13733610 rs3132407 SNP_A-8340403          0
554588 869844         21 13852569 rs2775671 SNP_A-8340413          0
554565 869864         21 14038583 rs2775018 SNP_A-8340427          0
554544 869889         21 14136579 rs3115511 SNP_A-8340440          0
554990 869922         21 14396024 rs2822404 SNP_A-8340775          0
137187 869925         21 14404476 rs1556276 SNP_A-1968967          0

> # Add metadata to describe the columns
> meta <- varMetadata(snpAnnot)
> meta[c("snpID", "chromosome", "position", "rsID", "probeID"),
+   "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",
+   "unique ID from Affymetrix")
> varMetadata(snpAnnot) <- meta
```

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 gender and race. The initial sample annotation file is created from the raw data supplied by the genotyping center and/or study investigator, providing a mapping from the raw data file(s) for each sample scan to

other sample information such as sex, coded as `M` and `F`, ethnicity, unique scan identifier, called `scanID`, and unique subject identifier. 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).

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

 [1] "scanID"     "subjectID"  "family"     "father"     "mother"
 [6] "CoriellID"  "race"       "sex"        "status"     "genoRunID"
[11] "plate"      "alleleFile" "chpFile"    "missing.e1" "duplicated"
[16] "het.A"      "het.X"

> # data
> head(pData(scanAnnot))

   scanID subjectID family father mother CoriellID race sex status
3       3 200150062     28      0      0   NA18912  YRI   F      0
5       5 200122600   1341      0      0   NA07034  CEU   M      1
14     14 200122151     58      0      0   NA19222  YRI   F      0
15     15 200033736      9      0      0   NA18508  YRI   F      1
17     17 200116780   1344      0      0   NA12056  CEU   M      1
28     28 200003216     28      0      0   NA18913  YRI   M      0
                                            genoRunID              plate
3  GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A03_31250 GAINmixHapMapAffy2
5  GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A05_31282 GAINmixHapMapAffy2
14 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B02_31236 GAINmixHapMapAffy2
15 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B03_31252 GAINmixHapMapAffy2
17 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B05_31284 GAINmixHapMapAffy2
28 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_C04_31270 GAINmixHapMapAffy2
                                                                    alleleFile
3  GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A03_31250.BIRDSEED.ALLELE_SUMMARY.TXT
5  GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A05_31282.BIRDSEED.ALLELE_SUMMARY.TXT
14 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B02_31236.BIRDSEED.ALLELE_SUMMARY.TXT
15 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B03_31252.BIRDSEED.ALLELE_SUMMARY.TXT
17 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B05_31284.BIRDSEED.ALLELE_SUMMARY.TXT
28 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_C04_31270.BIRDSEED.ALLELE_SUMMARY.TXT
                                                                        chpFile
3  GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A03_31250.BIRDSEED.CHP.TXT
5  GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A05_31282.BIRDSEED.CHP.TXT
14 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B02_31236.BIRDSEED.CHP.TXT
15 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B03_31252.BIRDSEED.CHP.TXT
```

```
17 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B05_31284.BIRDSEED.CHP.TXT
28 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_C04_31270.BIRDSEED.CHP.TXT
     missing.e1 duplicated     het.A     het.X
3  0.0327272727      FALSE 0.2828485 0.2805611
5  0.0018181818      FALSE 0.3071142 0.0000000
14 0.0336363636      FALSE 0.2886546 0.2955912
15 0.0345454545      FALSE 0.2935273 0.2816901
17 0.0024242424      FALSE 0.2597793 0.0000000
28 0.0006060606      FALSE 0.2897898 0.0000000

> # Add metadata to describe the columns
> meta <- varMetadata(scanAnnot)
> meta[c("scanID", "subjectID", "family", "father", "mother",
+   "CoriellID", "race", "sex", "status", "genoRunID", "plate",
+   "alleleFile", "chpFile"), "labelDescription"] <-
+    c("unique integer 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",
+   "data file with intensities",
+   "data file with genotypes and quality scores")
> varMetadata(scanAnnot) <- meta
```

2.4 Creating the NetCDF Files

The data for genotype calls, allelic intensities and other variables such as BAlleleFrequency are stored as NetCDF files. More information on the NetCDF file format in general can be found at http://www.unidata.ucar.edu/software/netcdf/. The GWASTools package depends on the ncdf library. Documentation for the ncdf R library can be found at http://cran.r-project.org/web/packages/ncdf/ncdf.pdf. The ncdf library provides a convenient R interface to create, populate and extract data from NetCDF files.

For each study, three different NetCDF files are created to be used in subsequent cleaning and analysis steps. This format is used for the ease with which multi-dimensional arrays of data can be stored and accessed.

All NetCDF files created have two dimensions, one called `snp` and one titled `sample`. The `snp` dimension is of the same length as the number of probes that were released from the genotyping center and listed in the SNP annotation file. The `sample` dimension is of length equal to the number of genotyping scans released as listed in the sample annotation file. Further, all NetCDF files have three variables in common: `sampleID`, `chromosome` and `position`. The `sampleID` is

used for indexing the columns of the two dimensional values stored in the NetCDF files (genotype calls, for example). The `sampleID` ordering must match the `scanID` values as listed in the sample annotation file, see Section 2.3. The index to the SNP probes in the NetCDF file is the `snpID`, which is stored as values of the SNP dimension. 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 file, see 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 NetCDF file. We recommend always checking the ordering of these variables before writing new versions of the SNP or sample annotation data files.

Genotype NetCDF Files

The genotype NetCDF 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 NetCDF 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 NetCDF file for Broad

- The first step in creating a NetCDF file is to create a 'shell' NetCDF file to which the data will be written. The function `ncdfCreate` creates the two dimensions and three common variables as described above. It also assigns values of the SNP dimension as `snpID` values and populates the `chromosome` and `position` variables.

  In this case, we also want the `genotype` variable to be created, so the `vars` argument must be set to `"genotype"`. In the interest of computational feasibility, only 3 samples will be written in this step.

```
> geno.nc.file <- "tmp.geno.nc"
> # get snp annotation data frame for function
> snp <- affy_snp_annot[,c("snpID", "chromosome", "position")]
> ncdfCreate(ncdf.filename = geno.nc.file,
+        snp.annotation = snp,
+        variables = c("genotype"),
+        array.name = "AffyGenomeWideSNP_6",
+        genome.build = "36",
+        n.samples = 3,
+        precision = "single")

[[1]]
[1] 65536
```

- Now the shell has been created so we can call the `ncdfAddData` function to populate the NetCDF with the genotype data stored in the raw data files. The data are written to the NetCDF file one sample at a time and, simultaneously, the corresponding sample identifier `scanID` is written to the `sampleID` variable. The `chpFile` variable from the sample annotation file holds the name of the CHP file for each sample scan; these are the files we must read in to get genotype data for each sample.

```
> # first 3 samples only
> scan.annotation <- affy_scan_annot[1:3, c("scanID", "genoRunID", "chpFile")]
> names(scan.annotation) <- c("scanID", "scanName", "file")
> # indicate which column of SNP annotation is referenced in data files
> snp.annotation <- affy_snp_annot[,c("snpID", "probeID")]
> names(snp.annotation) <- c("snpID", "snpName")
```

- The arguments to `ncdfAddData` must be created. We need `col.nums`, which is an integer vector indicating which columns of the raw text file contain variables for input.

```
> col.nums <- as.integer(c(2,3,5,6))
> names(col.nums) <- c("snp","geno","a1","a2")
> # Define a path to the raw data CHP text files which are read by
> #   ncdfAddData to access the raw genotypic data
> path <- system.file("extdata", "affy_raw_data",
+   package="GWASdata")
```

- All required arguments have been defined so we are ready to call `ncdfAddData`. This function will take the previously created genotype NetCDF file and populate it with the genotype data from the CHP text files and with the sample identifier `scanID` corresponding to that CHP file in the sample annotation data.frame. A set of diagnostic values are written and stored in `diag.geno`, so we must look at those to ensure no errors occurred. For more information on the function `ncdfAddData`, such as argument specification, diagnostics recorded and examples, please see the function documentation.

```
> diag.geno.file <- "diag.geno.RData"
> diag.geno <- ncdfAddData(path = path,
+   ncdf.filename = geno.nc.file,
+   snp.annotation = snp.annotation,
+   scan.annotation = scan.annotation,
+   sep.type = "\t",
+   skip.num = 1,
+   col.total = 6,
+   col.nums = col.nums,
+   scan.name.in.file = -1,
+   scan.start.index = 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 NetCDF file
> table(diag.geno$chk)

1
3
```

- Although the diagnostic values indicated no issues, we will open the NetCDF file and extract a small sampling of data. This illustrates the use of the `NcdfGenotypeReader` class for retrieving data from a NetCDF file.

```
> (genofile <- NcdfGenotypeReader(geno.nc.file))
```

```
[1] "file tmp.geno.nc has 2 dimensions:"
[1] "sample   Size: 3"
[1] "snp   Size: 3300"
[1] "------------------------"
[1] "file tmp.geno.nc has 4 variables:"
[1] "int sampleID[sample]  Longname:sampleID Missval:0"
[1] "int position[snp]  Longname:position Missval:-1"
[1] "int chromosome[snp]  Longname:chromosome Missval:-1"
[1] "byte genotype[snp,sample]  Longname:genotype Missval:-1"

> # Take out genotype data for the first 3 samples and
> #   the first 5 SNPs
> (genos <- getGenotype(genofile, snp=c(1,5), scan=c(1,3)))

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

> # Close the NetCDF file
> close(genofile)

[[1]]
[1] 65536
```

- Run the function `ncdfCheckGenotype` to check that the NetCDF file contains the same data as the raw data files.

```
> check.geno.file <- "check.geno.RData"
> check.geno <- ncdfCheckGenotype(path = path,
+   ncdf.filename = geno.nc.file,
+   snp.annotation = snp.annotation,
+   scan.annotation = scan.annotation,
+   sep.type = "\t",
+   skip.num = 1,
+   col.total = 6,
+   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"          "snp.order"    "geno.chk"

> # 'snp.order' is a vector indicating whether (1) or not (0) the snp ids
> #   are in the same order in each file.
> table(check.geno$snp.order)

1
3

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

1
3
```

Intensity NetCDF Files

The intensity NetCDF 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 NetCDF for all samples, for all SNPs. (A separate NetCDF file will store the BAlleleFreq and LogRRatio data.)

Aside from the three variables held in common with all NetCDF files (`sampleID, chromosome` and `position`), the intensity and quality data are written to the intensity NetCDF 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 NetCDF and diagnostics are written.

Affymetrix data are provided in two files per genotyping scan. The CHP file holds the genotype calls, used to create the genotype NetCDF file, as well as the confidence score, which is written to the `quality` variable in the intensity NetCDF file. The normalized X and Y intensity data are stored in the `allele_summary` files in the format of two rows per SNP, one for each allelic probe. A separate function `ncdfAddIntensity` reads these data and writes them to the NetCDF file. Thus, when writing the intensity NetCDF file using Affymetrix data, there are two function calls needed, each of which opens and reads from the two sets of files.

Illumina data are provided in one file per genotyping scan. The confidence score, genotype call, normalized intensities as well as the BAlleleFrequency and LogRRatio values are all stored in one file. Because of this, we do not need to invoke the `ncdfAddIntensity` function when reading in Illumina data to the intensity NetCDF.

Creating the Intensity NetCDF file for Broad

- The first step in creating the intensity NetCDF is to create a 'shell' NetCDF file to which the data will be written. We can use the same function used for creating the shell genotype NetCDF, `ncdfCreate`. The two dimensions and three common variables will be created along with the X, Y and quality variables; the `vars` argument is set to write these. For a reasonable computation time, we will create the file to hold data for 3 samples only.

```
> qxy.nc.file <- "tmp.qxy.nc"
> # get snp annotation data frame for function
> snp <- affy_snp_annot[,c("snpID", "chromosome", "position")]
> ncdfCreate(ncdf.filename = qxy.nc.file,
+        snp.annotation = snp,
+        variables = c("quality","X","Y"),
+        array.name = "AffyGenomeWideSNP_6",
+        genome.build = "36",
+        n.samples = 3,
+        precision = "single")

[[1]]
[1] 65536
```

- We next call the `ncdfAddData` function to populate the intensity NetCDF with the X and Y intensities and the quality score stored in the raw CHP text files. The sample identifier `scanID` is also populated at the same time. The `chpFile` variable from the sample annotation file holds the name of the CHP file for each sample scan. From this file we can get all needed data for the intensity NetCDF file.

```
> # first 3 samples only
> scan.annotation <- affy_scan_annot[1:3, c("scanID", "genoRunID", "chpFile")]
> names(scan.annotation) <- c("scanID", "scanName", "file")
> # indicate which column of SNP annotation is referenced in data files
> snp.annotation <- affy_snp_annot[,c("snpID", "probeID")]
> names(snp.annotation) <- c("snpID", "snpName")
```

- The arguments to `ncdfAddData` must be created. We need `col.nums`, which is an integer vector indicating which columns of the raw text file contain variables for input.

```
> col.nums <- as.integer(c(2,4))
> names(col.nums) <- c("snp","qs")
> # Define a path to the raw data CHP text files which are read by
> #   ncdfAddData to access the raw genotypic data
> path <- system.file("extdata", "affy_raw_data",
+   package="GWASdata")
```

- All required arguments have been defined so we are ready to call `ncdfAddData`. This function will take the previously created intensity NetCDF file and populate it with the quality score from the CHP text files. Recall the intensity values are stored in a different set of files so those will be populated in the next step. A set of diagnostic values are written and stored in `diag.qual`.

```
> diag.qual.file <- "diag.qual.RData"
> diag.qual <- ncdfAddData(path = path,
+   ncdf.filename = qxy.nc.file,
+   snp.annotation = snp.annotation,
```

```
+   scan.annotation = scan.annotation,
+   sep.type = "\t",
+   skip.num = 1,
+   col.total = 6,
+   col.nums = col.nums,
+   scan.name.in.file = -1,
+   scan.start.index = 1,
+   diagnostics.filename = diag.qual.file,
+   verbose = FALSE)
```

- As alluded to above, the normalized X and Y intensity values are stored in the `allele files`. To write the intensities, we will call `ncdfAddIntensity` with similar arguments to the `ncdfAddData` function. For further explanation of this function, please refer to the function help documentation.

```
> scan.annotation <- affy_scan_annot[1:3, c("scanID", "genoRunID", "alleleFile")]
> names(scan.annotation) <- c("scanID", "scanName", "file")
> diag.xy.file <- "diag.xy.RData"
> diag.xy <- ncdfAddIntensity(path = path,
+   ncdf.filename = qxy.nc.file,
+   snp.annotation = snp.annotation,
+   scan.annotation = scan.annotation,
+   scan.start.index = 1,
+   n.consecutive.scans = 3,
+   diagnostics.filename = diag.xy.file,
+   verbose = FALSE)
```

- We will open the NetCDF file and extract a small sampling of data. In this case we use the `NcdfIntensityReader` class.

```
> # Open the NetCDF file we just created
> (intenfile <- NcdfIntensityReader(qxy.nc.file))

[1] "file tmp.qxy.nc has 2 dimensions:"
[1] "sample   Size: 3"
[1] "snp   Size: 3300"
[1] "------------------------"
[1] "file tmp.qxy.nc has 6 variables:"
[1] "int sampleID[sample]  Longname:sampleID Missval:0"
[1] "int position[snp]  Longname:position Missval:-1"
[1] "int chromosome[snp]  Longname:chromosome Missval:-1"
[1] "float quality[snp,sample]  Longname:quality Missval:-9999"
[1] "float X[snp,sample]  Longname:X Missval:-9999"
[1] "float Y[snp,sample]  Longname:Y Missval:-9999"
```

```
> # Take out the normalized X intensity values for the first
> #     5 SNPs for the first 3 samples
> (xinten <- getX(intenfile, snp=c(1,5), scan=c(1,3)))

          [,1]      [,2]      [,3]
[1,]  501.2622  385.5622  356.8760
[2,]  614.1541  651.9782  710.6095
[3,] 1968.4933 2550.7141 2265.3674
[4,] 1607.2856  293.1671 2906.5942
[5,]  398.2835 1902.5592 1355.5342

> # Close the NetCDF file
> close(intenfile)

[[1]]
[1] 65536
```

- Run the function `ncdfCheckIntensity` to check that the NetCDF file contains the same data as the raw data files.

```
> scan.annotation <- affy_scan_annot[1:5,
+   c("scanID", "genoRunID", "chpFile", "alleleFile")]
> names(scan.annotation) <- c("scanID", "scanName", "file", "inten.file")
> check.qxy.file <- "check.qxy.RData"
> check.qxy <- ncdfCheckIntensity(path = path,
+   intenpath = path,
+   ncdf.filename = qxy.nc.file,
+   snp.annotation = snp.annotation,
+   scan.annotation = scan.annotation,
+   sep.type = "\t",
+   skip.num = 1,
+   col.total = 6,
+   col.nums = col.nums,
+   scan.name.in.file = -1,
+   check.scan.index = 1:3,
+   n.scans.loaded = 3,
+   affy.inten = TRUE,
+   diagnostics.filename = check.qxy.file,
+   verbose = FALSE)
```

BAlleleFrequency and LogRRatio NetCDF Files

The BAlleleFrequency and LogRRatio NetCDF file stores these values for every sample by SNP. For Affymetrix data, these values must be calculated, but for Illumina data these values are calculated by the BeadStudio software and may be provided by the genotyping center. For the purposes of this tutorial, we will be calculating the BAlleleFrequency and LogRRatio values from the Broad

data as explained in the following manner. 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

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

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

Illumina data may come with R and θ values for each sample, where 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). AffyMetrix data uses X and Y values from a Cartesian system. To transform to the R and θ polar coordinates, we use equation 1.

Regardless of which platform our data come from, we are able to find the R and θ values for every sample and every SNP. It is from these values that we calculate the LogRRatio and BAlleleFrequency. The LogRRatio 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.

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

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

The BAlleleFrequency 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 BAlleleFrequency value. Note the BAlleleFrequency 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 BAlleleFrequency is calculated

$$\text{BAlleleFrequency} = \begin{cases} 0 & \text{if } \theta_n < \theta_{AA} \\ \frac{(1/2)(\theta_n - \theta_{AA})}{\theta_{AB} - \theta_{AA}} & \text{if } \theta_{AA} \leq \theta_n < \theta_{AB} \\ \frac{1}{2} + \frac{(1/2)(\theta_n - \theta_{AB})}{\theta_{BB} - \theta_{AB}} & \text{if } \theta_{AB} \leq \theta_n < \theta_{BB} \\ 1 & \text{if } \theta_n \geq \theta_{BB} \end{cases}$$

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, $BAlleleFrequency \in [0, 1]$ 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.

[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).

We use the BAlleleFrequency and LogRRatio 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 BAlleleFrequency values are recommended to be filtered out, for the genotype data are not reliable in these situations. Because of these applications, the BAlleleFrequency and LogRRatio values are a salient part of the data cleaning steps.

In addition to the three variables held in common with all NetCDF files (`sampleID, chromosome` and `position`), the BAlleleFrequency and LogRRatio values are calculated and written to this NetCDF in a two dimensional format, with SNPs corresponding to rows and samples corresponding to columns. Because we have already completed the creation of both the genotype and intensity NetCDF files, we simply use those files to access the data. The BAlleleFrequency and LogRRatio values are calculated in subsets for efficiency and written to the corresponding subset indices in the NetCDF file.

Creating the BAlleleFrequency and LogRRatio NetCDF file for Broad

- The first step in creating the BAlleleFrequency NetCDF is to create a 'shell' NetCDF file to which the data will be written. We can use the same function used for creating the other shell NetCDF files, `ncdfCreate`. The two dimensions and three common variables will be created along with the BAlleleFrequency and LogRRatio variables. In order to allow for a reasonable computation time, we will create the file to hold data for 3 samples only.

```
> bl.nc.file <- "tmp.bl.nc"
> # get snp annotation data frame for function
> snp <- affy_snp_annot[,c("snpID", "chromosome", "position")]
> ncdfCreate(ncdf.filename = bl.nc.file,
+        snp.annotation = snp,
+        variables = c("BAlleleFreq","LogRRatio"),
+        array.name = "AffyGenomeWideSNP_6",
+        genome.build = "36",
+        n.samples = 3,
+        precision = "single")

[[1]]
[1] 65536
```

- We now will calculate the BAlleleFrequency and LogRRatio values for each sample by SNP and write these values to the NetCDF by calling the function `BAFfromGenotypes`. We will also select "by.study" as the call method, so all 3 samples have their genotype clusters called together. In normal usage, we recommend calling Affymetrix genotypes "by.plate" (in which case the `plate.name` argument is passed to the function). For more detail regarding the `BAFfromGenotypes` function, please see the function documentation. After the function is complete, we will look at a few values to ensure the file was created successfully.

```
> xyNC <- NcdfIntensityReader(qxy.nc.file)
> genoNC <- NcdfGenotypeReader(geno.nc.file)
```

```
> BAFfromGenotypes(xyNC, genoNC,
+             bl.ncdf.filename = bl.nc.file,
+             min.n.genotypes = 0,
+             call.method ="by.study")

[[1]]
[1] 196608

> close(xyNC)

[[1]]
[1] 65536

> close(genoNC)

[[1]]
[1] 131072

> # Open the NetCDF file we just created
> (blfile <- NcdfIntensityReader(bl.nc.file))

[1] "file tmp.bl.nc has 2 dimensions:"
[1] "sample   Size: 3"
[1] "snp   Size: 3300"
[1] "------------------------"
[1] "file tmp.bl.nc has 5 variables:"
[1] "int sampleID[sample]  Longname:sampleID Missval:0"
[1] "int position[snp]  Longname:position Missval:-1"
[1] "int chromosome[snp]  Longname:chromosome Missval:-1"
[1] "float BAlleleFreq[snp,sample]  Longname:BAlleleFreq Missval:-9999"
[1] "float LogRRatio[snp,sample]  Longname:LogRRatio Missval:-9999"

> # Look at the BAlleleFrequency values for the first 5 SNPs
> (baf <- getBAlleleFreq(blfile, snp=c(1,5), scan=c(1,3)))

     [,1] [,2] [,3]
[1,]   NA    1   NA
[2,]  0.0   NA  0.0
[3,]  0.0   NA   NA
[4,]  0.5    1  0.0
[5,]  1.0    0  0.5

> # Close the NetCDF file
> close(blfile)

[[1]]
[1] 65536
```

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 gender. The function returns a list, with one element that holds the missing counts for each SNP and one element that holds the sex counts; we will save these results to be used later.

```
> library(GWASTools)
> library(GWASdata)
> # sex is required for this function, so we load the scan annotation
> data(affy_scan_annot)
> scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
> # open the netCDF file and create a GenotypeData object
```

```
> ncfile <- system.file("extdata", "affy_geno.nc",
+   package="GWASdata")
> nc <- NcdfGenotypeReader(ncfile)
> genoData <-  GenotypeData(nc, scanAnnot=scanAnnot)
> # Calculate the number of missing calls for each snp over all samples
> #     for each gender separately
> miss <- missingGenotypeBySnpSex(genoData)
> # Examine the results
> length(miss)

[1] 3

> names(miss)

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

> head(miss$missing.counts)

       M F
869828 0 0
869844 0 0
869864 0 0
869889 0 0
869922 0 0
869925 0 0

> miss$scans.per.sex

 M  F
27 20

> head(miss$missing.fraction)

869828 869844 869864 869889 869922 869925
     0      0      0      0      0      0
```

- 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 gender mis-annotations because the Y chromosome SNP missing call rates may change. We add the missing call rates to the SNP annotation table and save a new version.

```
> # get snp annotation
> data(affy_snp_annot)
> snpAnnot <- SnpAnnotationDataFrame(affy_snp_annot)
> 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.
0.000000 0.000000 0.000000 0.003007 0.000000 0.468100
```

- We will create some plots of the missing call rates so we can easily identify any outliers. Plots will be shown of Y chromosome SNP missing call rates for males, and overall SNP missing call rates. 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.

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

[1] 0

> # 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] 0.9815

> # 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] 0.989

> # 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] 0.99
```

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. Since we have both sample and SNP annotation loaded, we create a `GenotypeData` object including both these annotations to ensure consistency. 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 and one element holding the number of SNPs per chromosome. We check these values to ensure they are as expected.

```
> hist(snpAnnot$missing.n1[snpAnnot$chromosome == 25],
+      xlab="Fraction of Y chr SNP missing calls",
+      main="Y Chromosome SNP Missing Call Rate for Males")
```

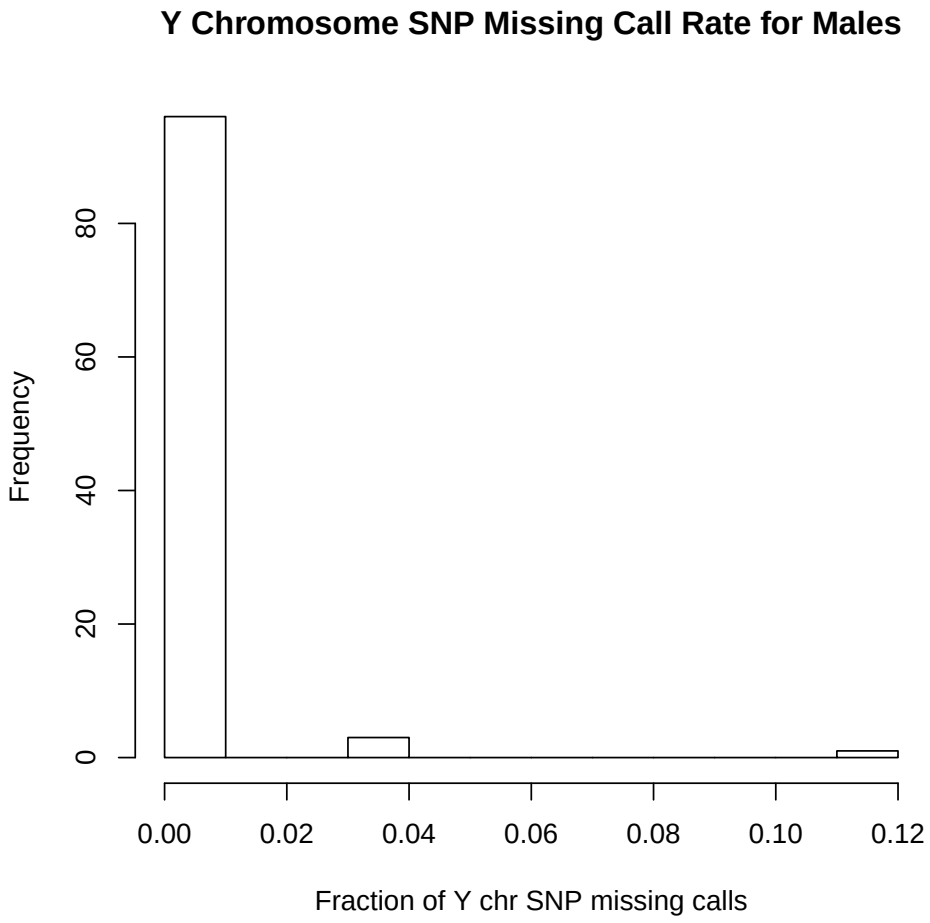


Figure 1: Missing call rate over all probes on the Y chromosome for the male samples only. The missing call rate for all females over the Y chromosome is identically 1. Although it may seem as if three samples have a higher missing call rate, one must note the small scale on the x-axis. There are no samples here that have an unacceptable missing call rate.

```
> # Want to exclude all SNP probes with 100% missing call rate
> # Check on how many SNPs to exclude
> sum(abs(snpAnnot$missing.n1 - 1) < 1e-6)

[1] 0

> sum(snpAnnot$missing.n1 == 1)

[1] 0
```

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

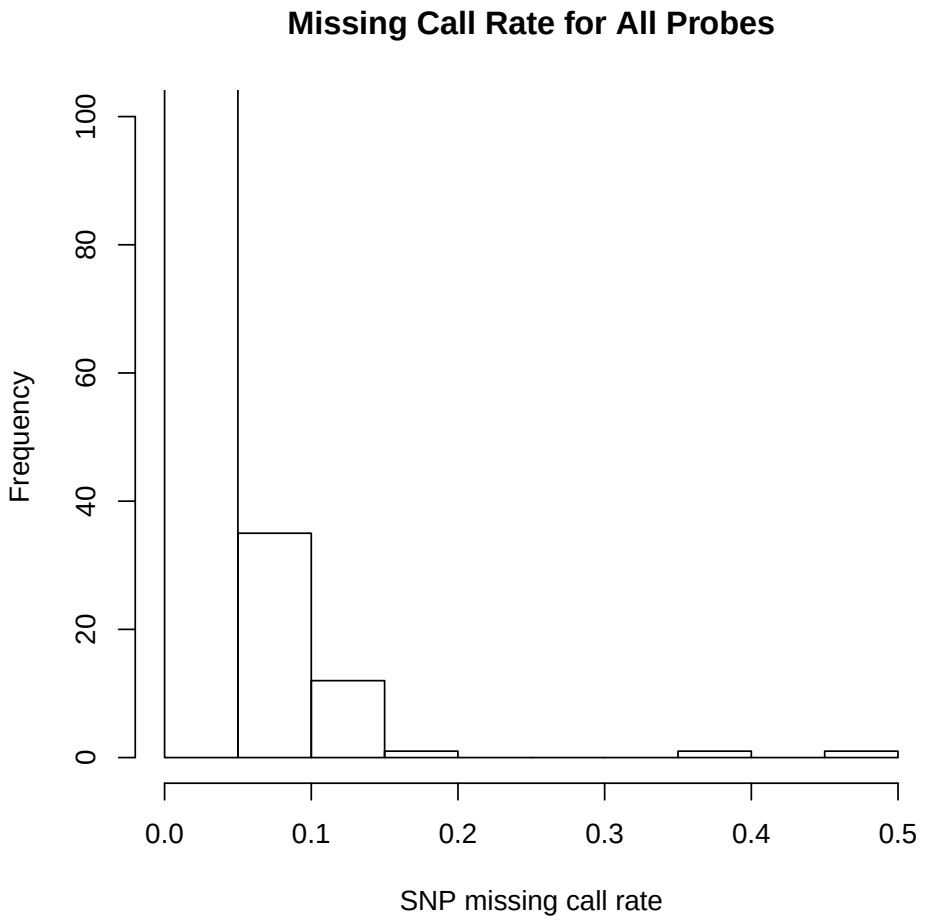


Figure 2: Histogram of overall missing call rate per SNP. Note the y-axis has been truncated in order to show the higher values of the missing call rate on the x-axis.

```
> # Create a variable that contains the IDs of these SNPs to exclude
> snpexcl <- snpAnnot$snpID[abs(snpAnnot$missing.n1 - 1) < 1e-6 |
+                           snpAnnot$missing.n1 == 1]
> length(snpexcl)

[1] 0

> # Use the missingGenotypeByScanChrom function
> miss <- missingGenotypeByScanChrom(genoData, snp.exclude=snpexcl)
> length(miss)
```

```
[1] 3

> names(miss)

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

> head(miss$missing.counts)

   21 22 X XY   Y M
3   2  4 2  0 100 0
5   0  4 2  0   0 0
14  2  6 2  1 100 0
15  4  3 6  1 100 0
17  2  4 1  0   0 1
28  1  1 0  0   0 0

> head(miss$snps.per.chr)

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

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

[1] 3300

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

snps
   0
```

- `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)

           3            5           14           15           17           28
0.0025000000 0.0018181818 0.0034375000 0.0043750000 0.0024242424 0.0006060606

> # 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.
0.0006061 0.0018180 0.0028120 0.0030250 0.0040620 0.0078790
```

- 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 gender differences. Finally, we calculate the number of samples with a missing call rate greater than 0.05. In this case, there are no 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.

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

     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.0006061 0.0016670 0.0018180 0.0026150 0.0033330 0.0078790

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

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.001875 0.002812 0.003438 0.003578 0.004062 0.006562

> # 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`. We then write a new version of the sample annotation with these added variables.

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

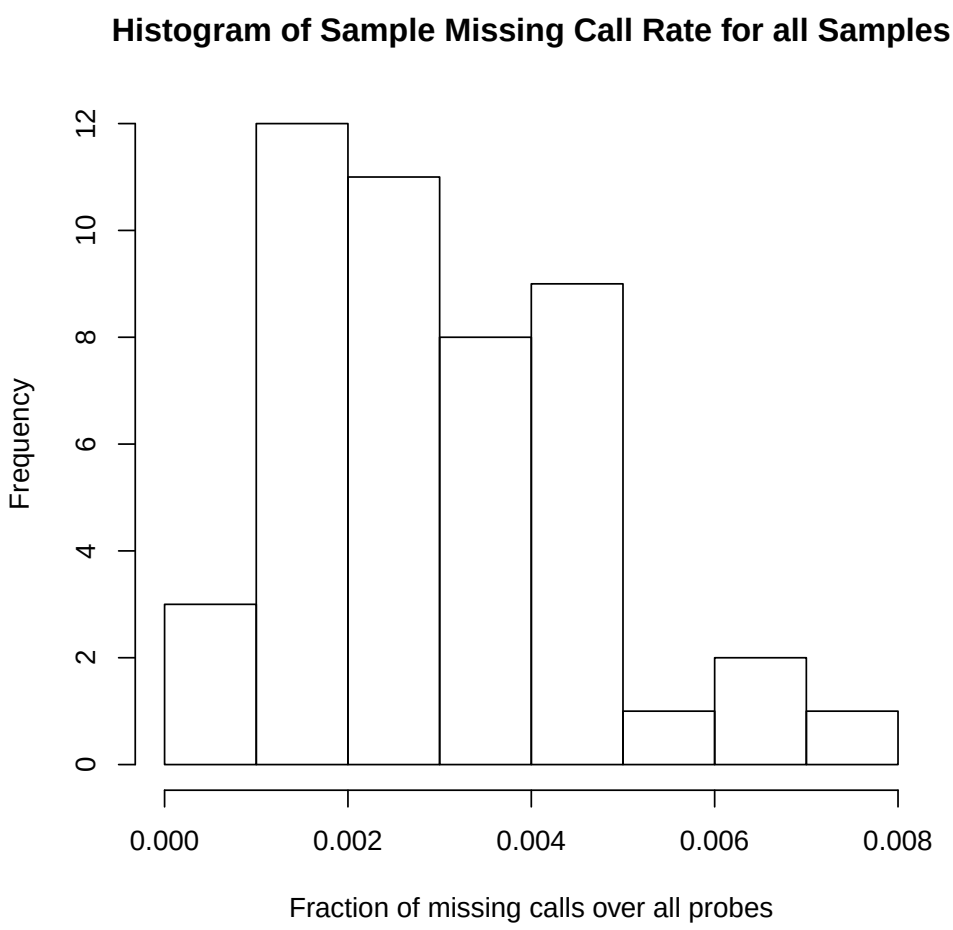


Figure 3: Histogram of missing call rate for all samples over all probes.

```
> 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.
0.001000 0.002000 0.003000 0.003266 0.004000 0.011500

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

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.000000 0.001000 0.002000 0.002426 0.004000 0.009000
```

```
> # 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, exclude=NULL)

FALSE  TRUE  <NA>
   43     4     0

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

[1] TRUE
```

Calculate `missing.n2`

- This step calculates `missing.n2`, which is the missing call rate per SNPs with `missing.e1` less than 0.05 over all samples. This calculation is done separately for each sex as in `missing.n1` in Section 3.1. We read in the sample annotation file which holds the `missing.e1` variable. In most 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 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). We review the values returned from the function and save the results.

```
> sum(scanAnnot$missing.e1 > 0.05)

[1] 0

> # Even though there are no samples with missing.e1 > 0.05,
> # we will go through the full process for the tutorial
```

```
> # 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)
> length(miss)

[1] 3

> names(miss) # Check that all SNPs are listed

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

> dim(miss$missing.counts)

[1] 3300    2

> miss$scans.per.sex

 M  F
27 20

> dim(miss$missing.fraction)

NULL

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

[1] TRUE

> 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.
0.000000 0.000000 0.000000 0.003007 0.000000 0.468100
```

- As we did in the step for calculating `missing.n1`, we will create some plots of the missing call rates so we can easily identify any outliers. Plots will be shown of Y chromosome SNP missing call rates for males, and overall SNP missing call rates. 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.

```
> # Find the number of SNPs with every call missing
> length(snpAnnot$missing.n2[snpAnnot$missing.n2 == 1])
```

```
> hist(snpAnnot$missing.n2[snpAnnot$chromosome == 25],
+      xlab="Fraction of Y chr SNP missing calls",
+      main="Y Chromosome SNP Missing Call Rate for Males")
```

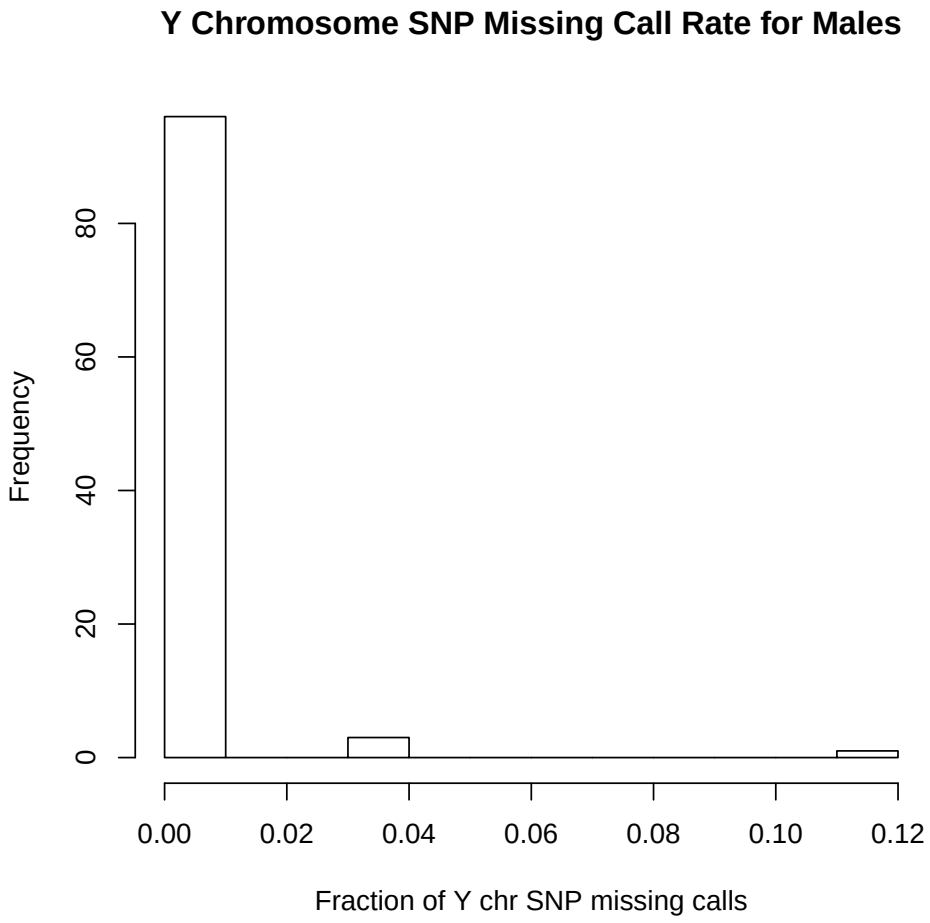


Figure 4: Missing call rate over all probes on the Y chromosome for the male samples excluding all samples with a missing call rate greater than 0.05. The missing call rate for all females over the Y chromosome is identically 1. Although it may seem as if three samples have a higher missing call rate, one must note the small scale on the x-axis. There are no samples here that have an unacceptable missing call rate.

```
[1] 0
```

```
> # Fraction of autosomal SNPs with missing call rate < .05
> x <- snpAnnot$missing.n2[snpAnnot$chromosome < 23]
> length(x[x < 0.05]) / length(x)
```

```
[1] 0.9815
```

```
> hist(snpAnnot$missing.n2, ylim=c(0,10000),
+      xlab="SNP missing call rate", main="Histogram of SNP Missing Call Rate")
```

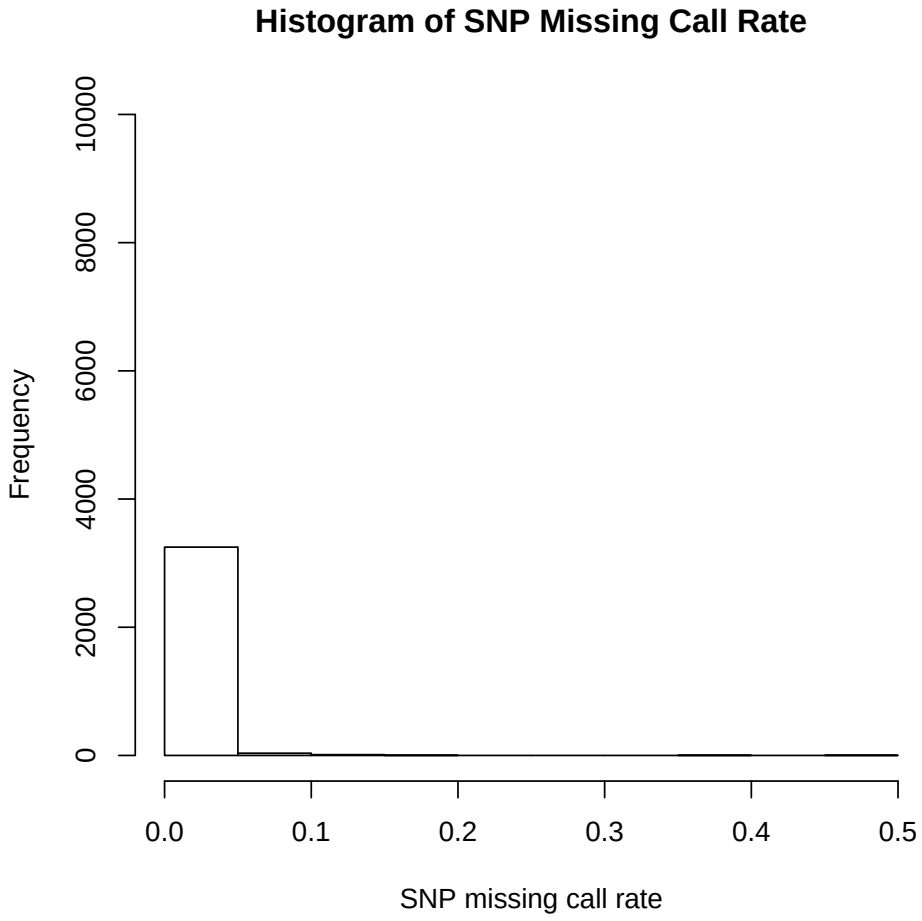


Figure 5: Histogram of overall missing call rate per SNP excluding samples with missing call rate greater than 0.05. Note the y-axis has been truncated in order to show the higher values of the missing call rate on the x-axis.

```
> # Fraction of X chromosome SNPs with missing call rate <.05
> x <- snpAnnot$missing.n2[snpAnnot$chromosome == 23]
> length(x[x < 0.05]) / length(x)

[1] 0.989

> # Fraction of Y chromosome SNPs with missing call rate <.05
> x <- snpAnnot$missing.n2[snpAnnot$chromosome == 25]
> length(x[x < 0.05]) / length(x)

[1] 0.99
```

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. We read in the new SNP annotation file which holds the `missing.n2` variable. Because we must exclude those SNPs with a sufficiently high missing call rate, we will create a vector of SNPs that have `missing.n2` greater than 0.05. Then, for the non-excluded SNPs we call the `missingGenotypeByScanChrom` function, which returns a list with one element holding the missing counts per sample by chromosome and one element holding the number of SNPs per chromosome. We check these values to ensure they are as expected.

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

[1] 50

> # Use the missingGenotypeByScanChrom function
> miss <- missingGenotypeByScanChrom(genoData, snp.exclude=snpexcl)
> length(miss)

[1] 3

> names(miss)

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

> head(miss$missing.counts)

   21 22 X XY  Y M
3   0  1 1  0 99 0
5   0  2 0  0  0 0
14  2  3 0  1 99 0
15  2  1 2  1 99 0
17  2  1 0  0  0 1
28  0  0 0  0  0 0

> head(miss$snps.per.chr)

 21  22   X  XY   Y   M
986 977 989  99  99 100

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

[1] 3250

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

```
snps
  50
```

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

```
           3            5           14           15           17           28
0.0006347191 0.0006153846 0.0019041574 0.0019041574 0.0012307692 0.0000000000
```

```
> # 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$missing.e2 <- miss$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 0.0006154 0.0012690 0.0015420 0.0019040 0.0055380
```

```
> # Make sure the samples are still in order
> allequal(scanAnnot$scanID, sort(scanAnnot$scanID))
```

```
[1] TRUE
```

- 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; we could identify that here. We also look at the summary of the missing call rate for females and males separately to ensure there are no large gender differences. Finally, we plot the samples sorted by missing call rate.

```
> # Look at missing.e2 for males
> summary(scanAnnot$missing.e2[scanAnnot$sex == "M"])
```

```
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.0000000 0.0003077 0.0006154 0.0013790 0.0020000 0.0055380
```

```
> # Look at missing.e2 for females
> summary(scanAnnot$missing.e2[scanAnnot$sex == "F"])
```

```
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.0006347 0.0012690 0.0015870 0.0017610 0.0019040 0.0041260
```

```
> 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")
```

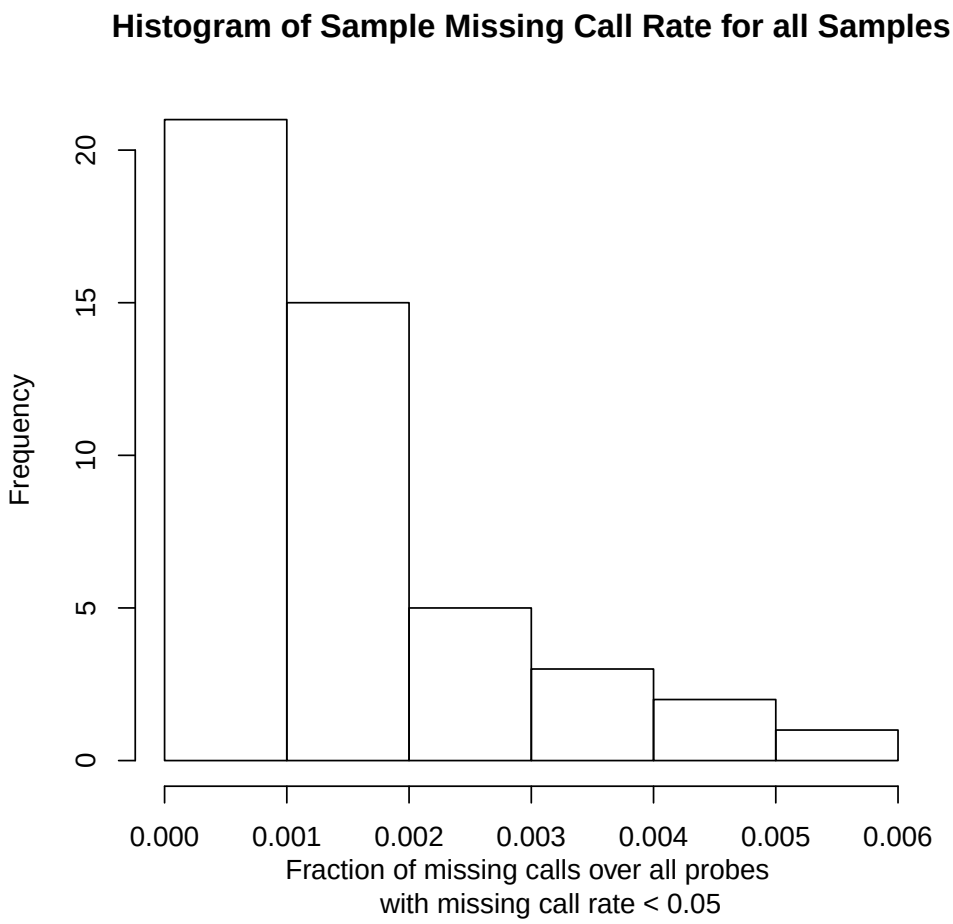


Figure 6: Histogram of missing call rate for all samples over all probes with missing call rate less than 0.05. There are 896,250 SNPs below this threshold. Note the highest sample has an overall missing call rate of 2 percent.

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 comparatively lower call rates. We find the distribution of number of samples per batch from the sample annotation file variable `geno.batch`. We calculate the mean missing call rate for all samples in each of the batches by simply taking the mean of all samples in a given batch of the missing call rate per sample value, stored in the sample annotation file as `missing.all`.

```
> varLabels(scanAnnot)
```

```
> plot(sort(scanAnnot$missing.e2), rank(sort(scanAnnot$missing.e2)),
+      xlab="Fraction of missing calls over probes with MCR < 0.05",
+      ylab="rank", cex.lab=0.75,
+      main="Sorted Fraction of Missing Call Rate \nOver All Probes with MCR < 0.05")
```

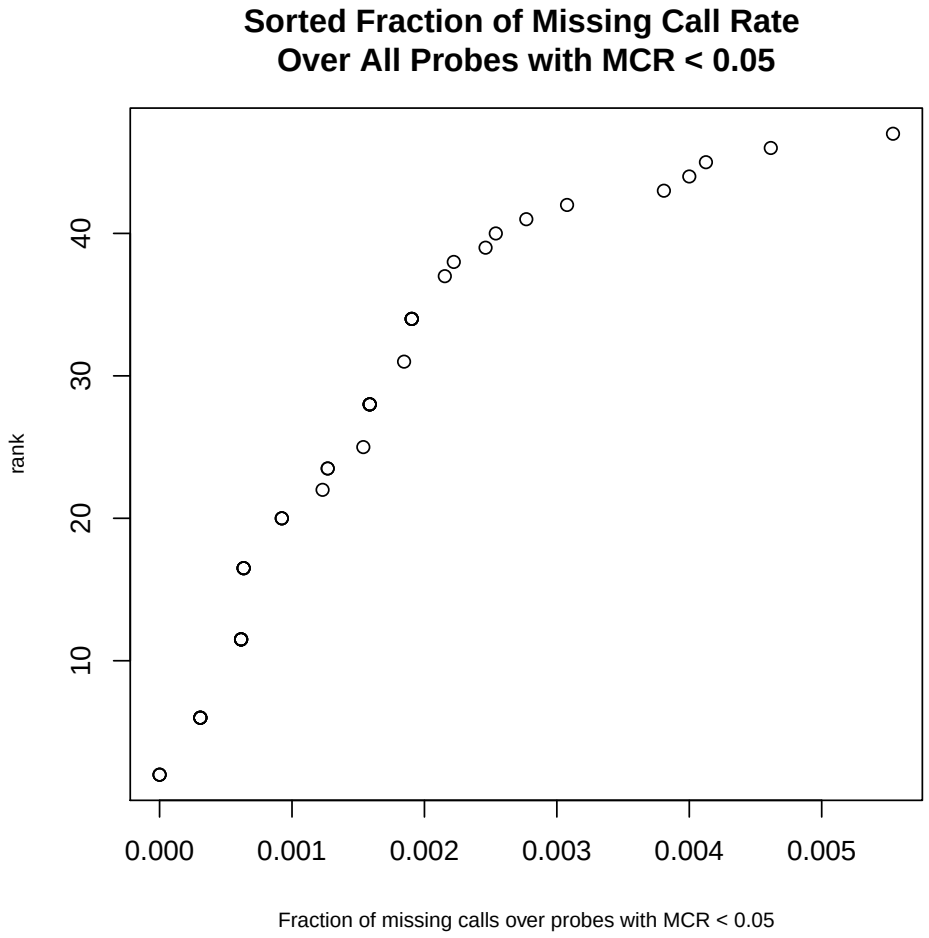


Figure 7: In increasing order, a plot of the fraction of missing calls for each sample over all probes with missing call rate less than 0.05.

```
 [1] "scanID"       "subjectID"    "family"       "father"       "mother"
 [6] "CoriellID"    "race"         "sex"          "status"       "genoRunID"
[11] "plate"        "alleleFile"   "chpFile"      "missing.e1"   "duplicated"
[16] "het.A"        "het.X"        "miss.e1.auto" "miss.e1.xchr" "missing.e2"

> # Check how many batches exist and how many samples are in each batch
> sum(is.na(scanAnnot$plate))

[1] 0
```

```
> length(unique(scanAnnot$plate))

[1] 3

> table(scanAnnot$plate, exclude=NULL)

GAINmixHapMapAffy1 GAINmixHapMapAffy2 GAINmixHapMapAffy3               <NA>
                15                 15                 17                  0

> # Plot the distribution of the number of samples per batch.
> barplot(table(scanAnnot$plate),
+         ylab="Number of Samples", xlab="Batch",
+         main="Distribution of Samples per Batch",
+         cex.axis=0.85, cex.names=0.75, cex.lab=0.9)
```

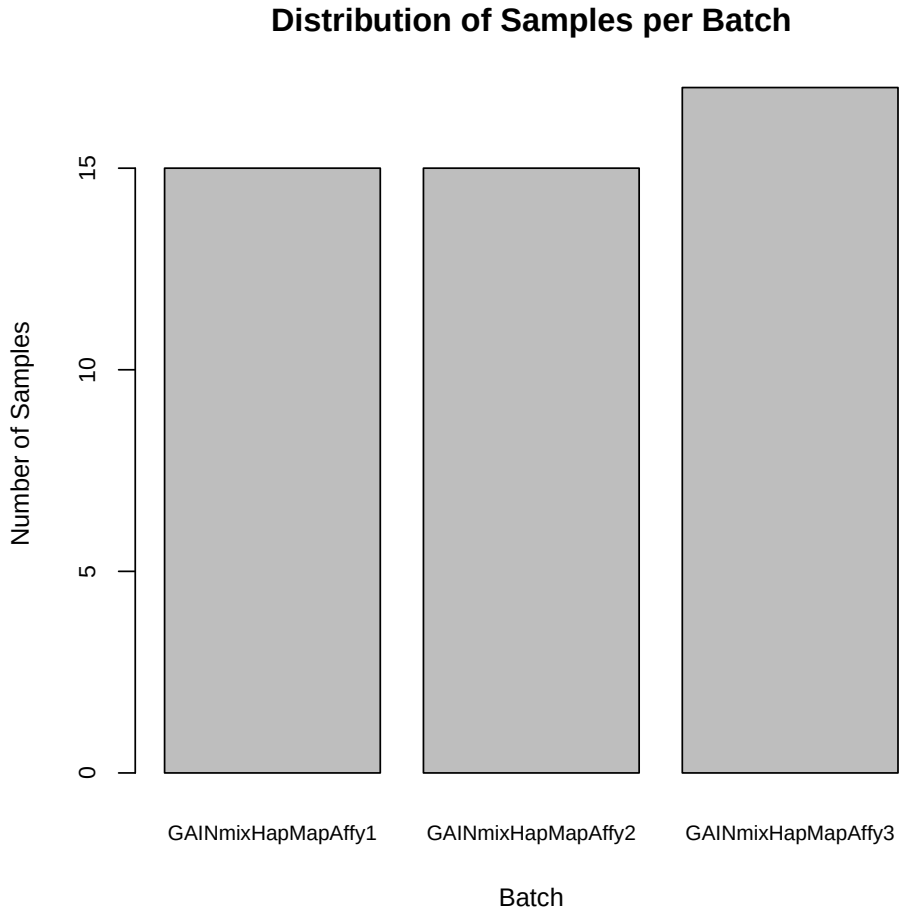


Figure 8: The distribution of the samples among the batches.

```
> # Examine the mean missing call rate per batch for all SNPs
> batches <- unique(scanAnnot$plate)
> bmiss <- rep(NA,length(batches)); names(bmiss) <- batches
> bn <- rep(NA,length(batches)); names(bn) <- batches
> for(i in 1:length(batches)) {
+   x <- scanAnnot$missing.e1[is.element(scanAnnot$plate, 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 3.7660e-07 3.7660e-07  1.2532 0.4642
Residuals  1 3.0051e-07 3.0051e-07
```

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 `plate` 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 2x2 tables for each SNP, comparing each plate with the other plates. This function returns the genomic inflation factors for each batch, as well as matrix of χ^2 values for each SNP.

```
> # Perform the Chi-square test using the batchChisqTest function
> res <- batchChisqTest(genoData, batchVar="plate", return.by.snp=TRUE)
> close(genoData)

[[1]]
[1] 65536
```

```
> plot(bn, bmiss,
+  xlab="Number of samples per batch", ylab="Mean missing call rate",
+  main="Mean Missing Call Rate vs\nSamples per Batch", cex.main=0.8)
> abline(y$coefficients)
```

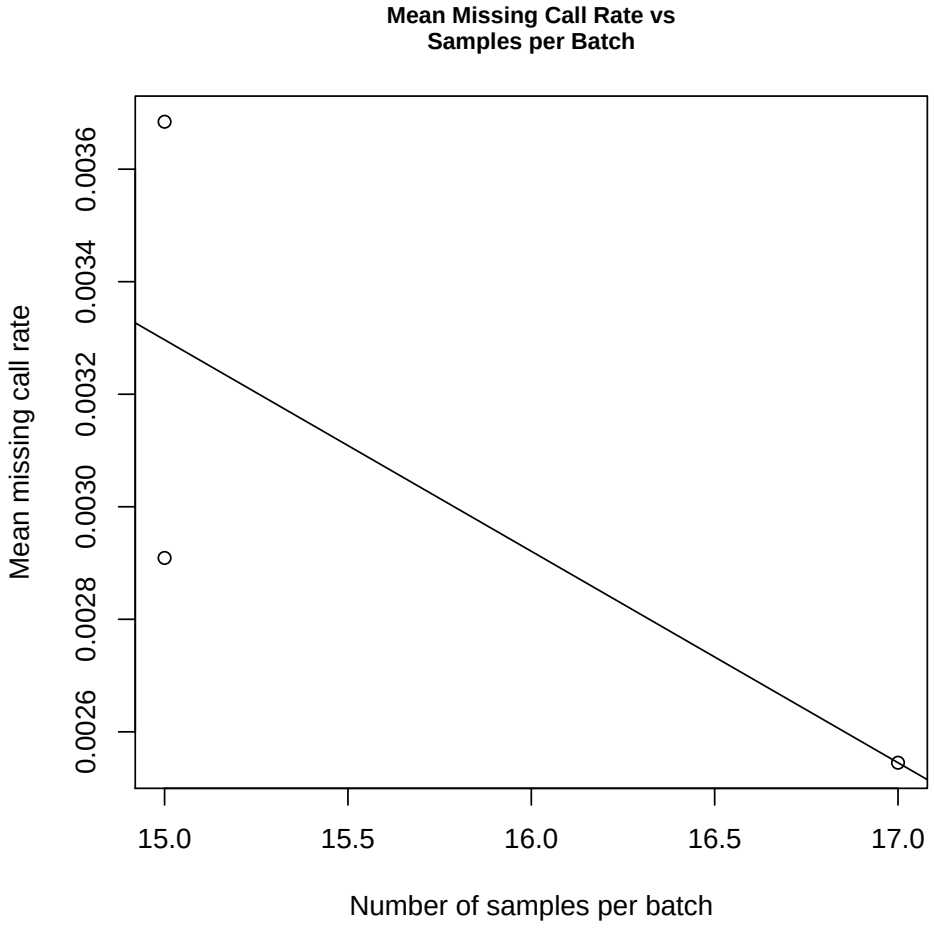


Figure 9: The number of samples per batch plotted against the mean missing call rate. The regression line has a nominal slope taken from the ANOVA results for the mean missing call rate per batch regressed on the number of samples per batch.

```
> # chi-square values for each SNP
> dim(res$chisq)

[1] 2000    3

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

```
GAINmixHapMapAffy1 GAINmixHapMapAffy2 GAINmixHapMapAffy3
         0.5704312          0.2425606          0.3074938

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

GAINmixHapMapAffy1 GAINmixHapMapAffy2 GAINmixHapMapAffy3
         0.8868969          0.4586820          0.6386302
```

- 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 ethnicity 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, exclude=NULL)
> x

 CEU  YRI <NA>
  29   18    0

> x[1] / sum(x)

      CEU
0.6170213

> x[2] / sum(x)

      YRI
0.3829787

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

      GAINmixHapMapAffy1 GAINmixHapMapAffy2 GAINmixHapMapAffy3
  CEU                  8                 10                 11
  YRI                  7                  5                  6

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

[1] 0.7167951
```

```
> # Check if ethnicity CEU has a batch effect
> # Calculate the fraction of samples in each batch that are CEU ethnicity
> batches <- unique(scanAnnot$plate)
> length(batches)

[1] 3

> eth <- rep(NA,length(batches)); names(eth) <- sort(batches)
> for(i in 1:length(batches)){
+   x <- scanAnnot$race[is.element(scanAnnot$plate, 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")
```

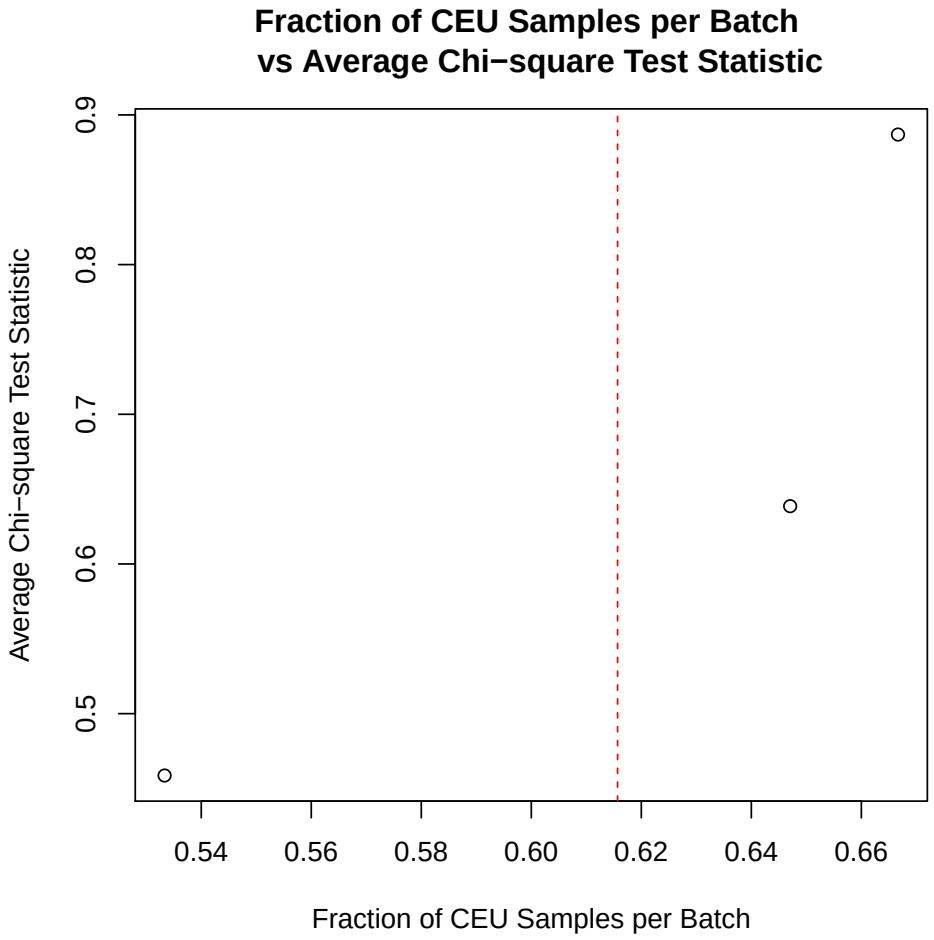


Figure 10: A plot of the fraction of samples in each batch from the CEU population group versus the average chi-square statistic for the particular batch. The red line is the average fraction of samples that are of European ancestry in a batch.

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 BAlleleFreq variance analysis; lastly, we check sample missingness and heterozygosities. We use Illumina data, since the B Allele Frequency and Log R Ratio are included in the dataset.

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.

```
> library(GWASTools)
> library(GWASdata)
> data(illumina_scan_annot)
> scanAnnot <- ScanAnnotationDataFrame(illumina_scan_annot)
> qxyfile <- system.file("extdata", "illumina_qxy.nc",
+   package="GWASdata")
> qualNC <- NcdfIntensityReader(qxyfile)
> qualData <- IntensityData(qualNC, scanAnnot=scanAnnot)
> genofile <- system.file("extdata", "illumina_geno.nc",
+   package="GWASdata")
> genoNC <- NcdfGenotypeReader(genofile)
> genoData <- GenotypeData(genoNC, scanAnnot=scanAnnot)
> qual.results <- qualityScoreByScan(qualData, genoData)
> close(qualData)

[[1]]
[1] 65536
```

Figure 11 shows 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.

4.2 BAlleleFreq variance analysis

The "BAlleleFreq" was previously calculated for each SNP-sample combination. "BAlleleFreq" is a standardized version of the polar coordinate angle. 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

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

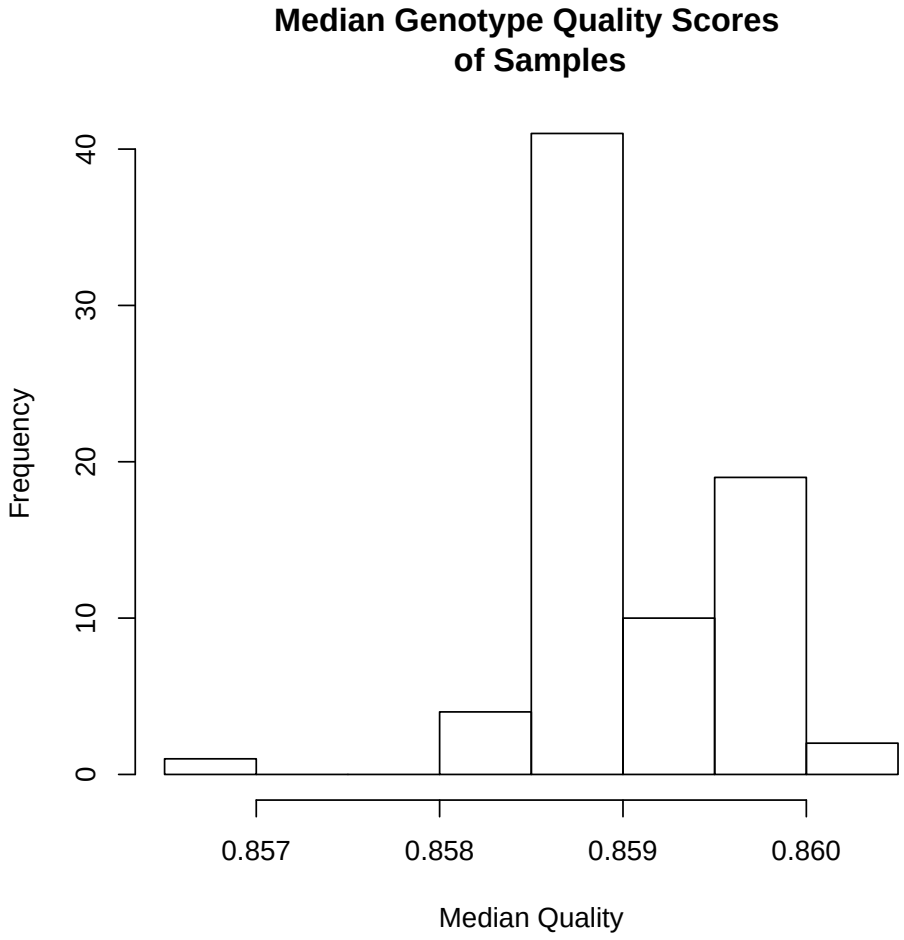


Figure 11: Median quality scores for each sample.

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 BAlleleFreq standard deviation, with "very high" defined as more than 4 standard deviations from the window's mean BAlleleFreq standard deviation over all samples. The output is a matrix listing all sample-chromosome pairs with high BAlleleFreq standard deviations, the number of windows with high

SDs in each pair, and the sample's sex. We examine plots of BAlleleFreq 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 BAlleleFreq and a `GenotypeData` object. Its output is a list of matrices, with one matrix for each chromosome containing the standard deviation of BAlleleFreq at each window in each scan.

```
> blfile <- system.file("extdata", "illumina_bl.nc",
+   package="GWASdata")
> blNC <- NcdfIntensityReader(blfile)
> blData <- IntensityData(blNC)
> nbins <- rep(12, 3)
> slidingBAF12 <- sdByScanChromWindow(blData, genoData, nbins=nbins)
> length(slidingBAF12)

[1] 3

> length(slidingBAF12)

[1] 3

> dim(slidingBAF12[[1]])

[1] 77 11

> dim(slidingBAF12[[2]])

[1] 77 11
```

- The function `meanBAFSDbyChromWindow` calculates the mean and standard deviation of the BAlleleFreq 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.

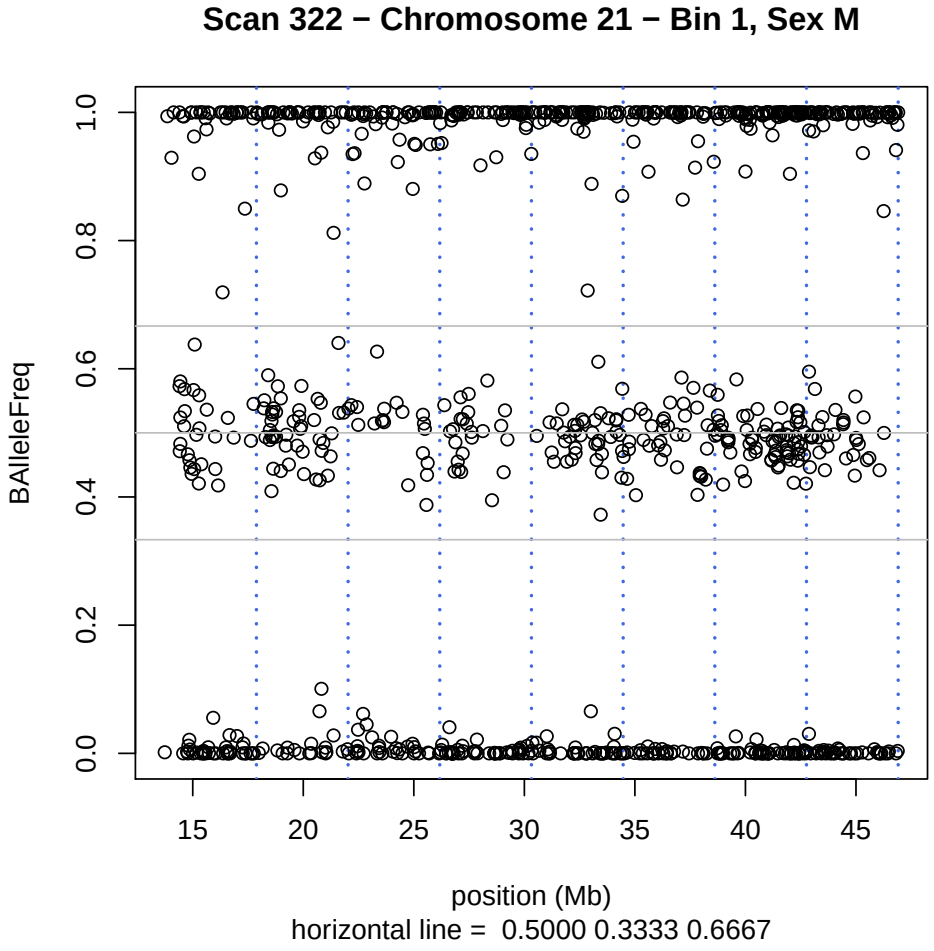
```
> sds.chr <- meanSdByChromWindow(slidingBAF12,
+                                scanAnnot$sex)
> sds.chr[["21"]]

           [,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 BAlleleFreq standard deviations compared to the same window in other samples. There are 80 sample-chromosome pairs that are flagged in this analysis, and seem to be evenly distributed across chromosomes.

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

[1] 6

> 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"

> sum(is.na(res12bin4sd))

[1] 0

> table(res12bin4sd[, 2])

21 22
 3  3

> all(res12bin4sd[res12bin4sd[, "sex"] == "F", "scanID"] %in%
+     scanAnnot$scanID[scanAnnot$sex == "F"])

[1] TRUE
```

```
> scanID <- as.integer(res12bin4sd[, "scanID"])
> chrom <- as.integer(res12bin4sd[, "chromosome"])
> chrom[res12bin4sd[, "chromosome"] == "X"] <- 23
> bincode <- paste("Bin", res12bin4sd[, "bin"], sep = " ")
> sexcode <- paste("Sex", res12bin4sd[, "sex"], sep = " ")
> code <- paste(bincode, sexcode, sep = ", ")
> plotind <- 1
> chromIntensityPlot(blData, scanID[plotind], chrom[plotind],
+                    code=code[plotind], type="BAF")
```



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

```
> close(blData)

[[1]]
[1] 65536
```

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 anomalies. Note that this step will be labor intensive for any reasonably large study. Better methods of detecting BAF anomalies are an active area of research.

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.

```
> 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)
```

- First, Figure 12 shows a boxplot of missingness in the autosomes, the X chromosome, and the pseudoautosomal region. Second, Figure 13 shows a boxplot of X chromosome missingness for each sex.

- We will call the function `hetByScanChrom` to calculate the heterozygosity. We want to store the heterozygosity calculations in the sample annotation, so after ensuring the ordering is correct we will write the heterozygosity values and store them in the sample annotation.

```
> # Calculate heterozygosity by scan by chromosome
> het.results <- hetByScanChrom(genoData)
> close(genoData)

[[1]]
[1] 131072

> # 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"
```

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

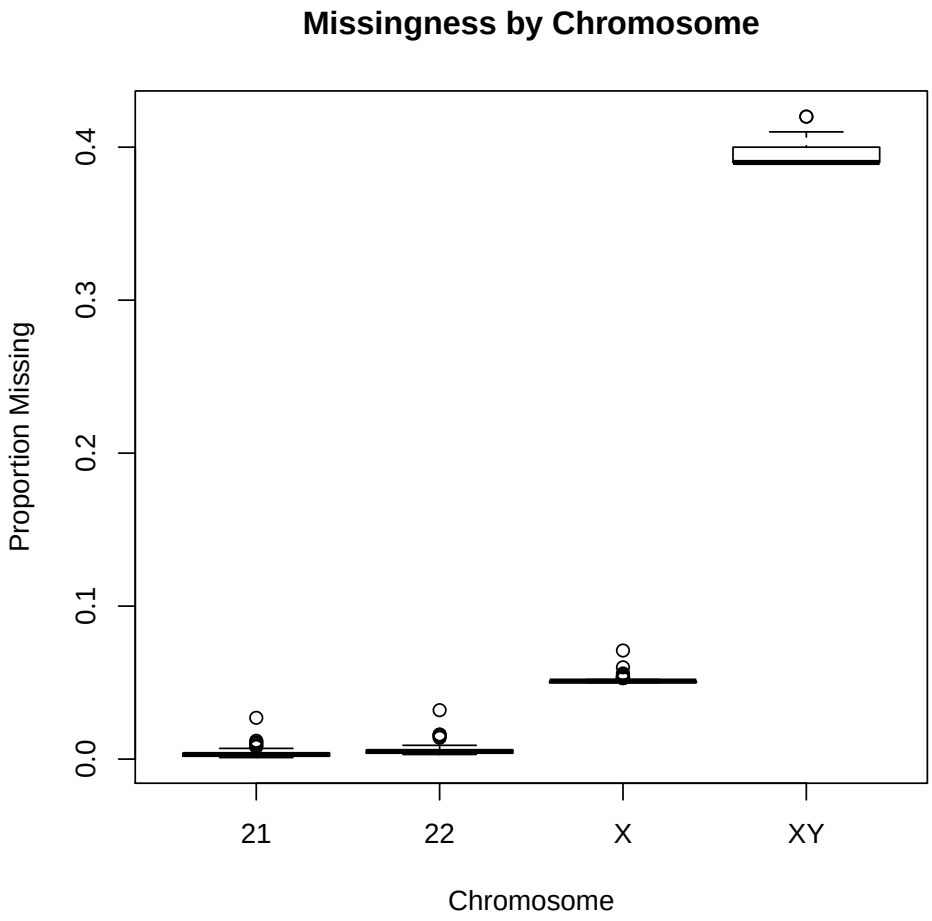


Figure 12: Boxplot of missingness by chromosome.

- There are two plots for heterozygosity. Figure 14 shows 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. Figure 15 shows a boxplot of female heterozygosity on the X chromosome, subsetted by population.

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

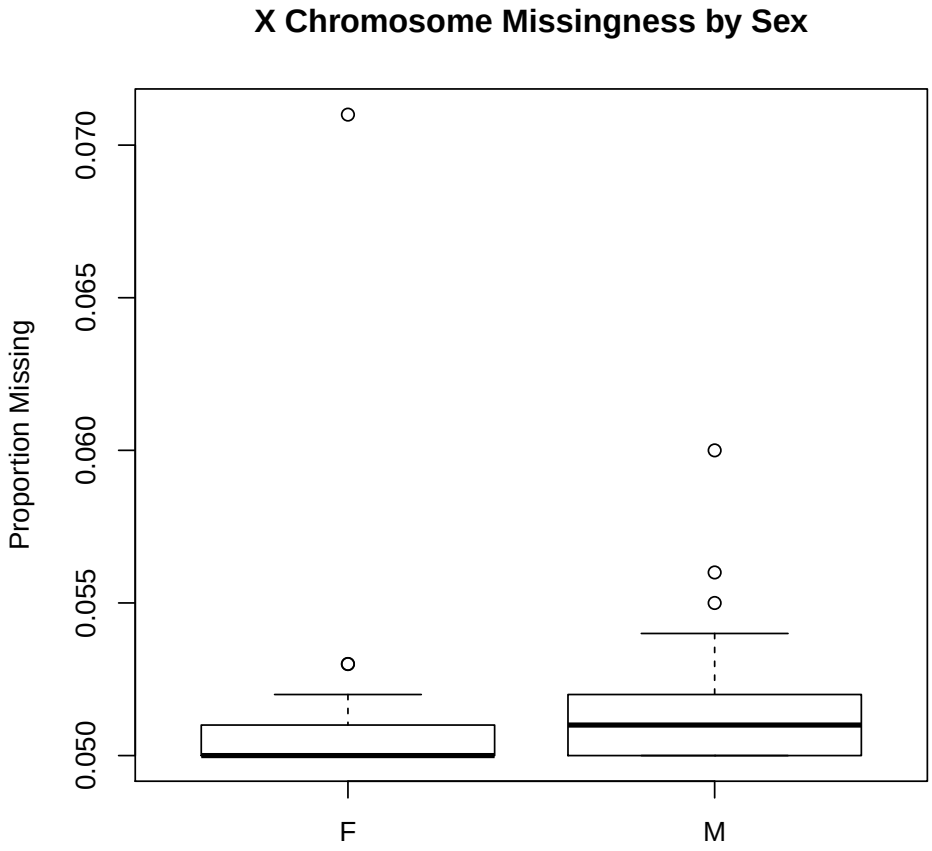


Figure 13: Boxplot of X chromosome missingness by sex.

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

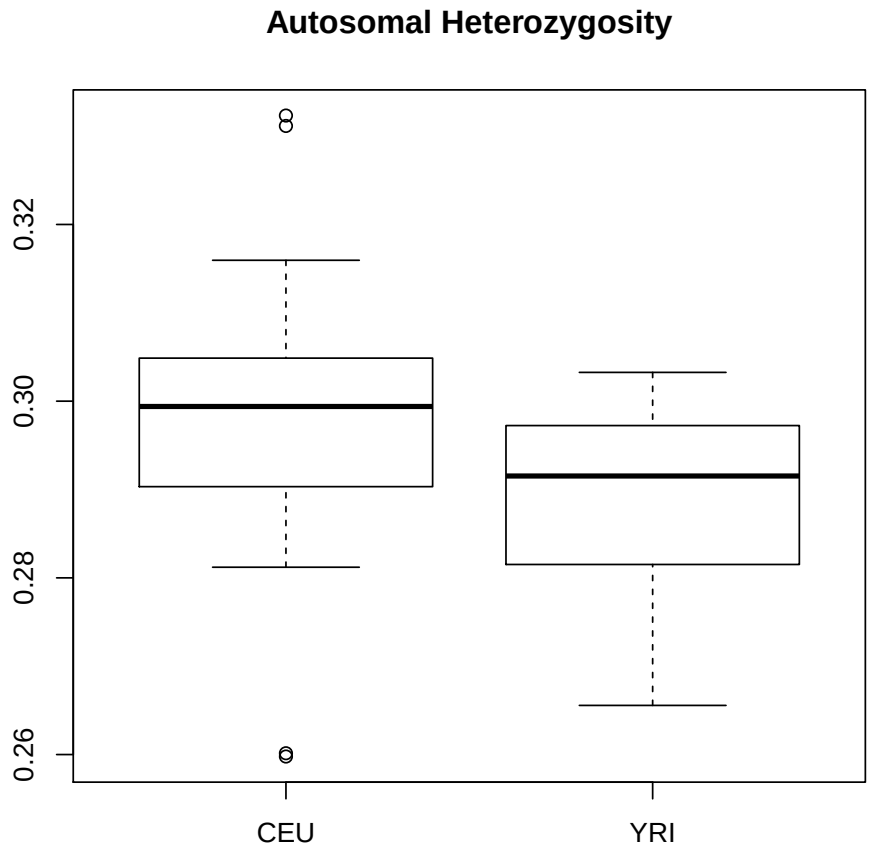


Figure 14: Boxplot of autosomal heterozygosity by continental ancestry.

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

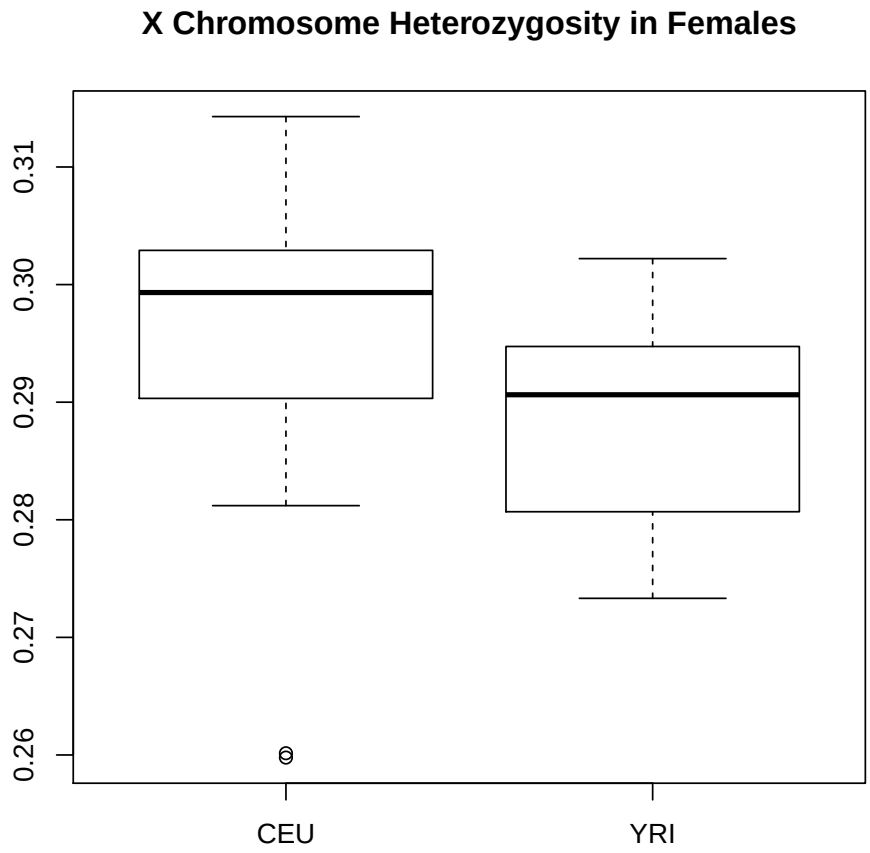


Figure 15: Boxplot of X chromosome heterozygosity by population.

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 gender discrepancies between the annotated and genetic genders 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 Gender Check

This section looks for discrepancies between the annotated and genetic genders. Gender identity 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 gender as well as sex chromosome aneuploidies. It is important to have accurate gender annotation not only for completeness but also for analyses which treat male and female samples separately. Any found gender mis-annotations are presented to the investigators in order to resolve discrepancies. If a genetic and recorded gender 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 i^{th} sample and the j^{th} chromosome is returned from the function.

```
> library(GWASTools)
> library(GWASdata)
> qxyfile <- system.file("extdata", "affy_qxy.nc",
+   package="GWASdata")
> intenNC <- NcdfIntensityReader(qxyfile)
> inten.by.chrom <- meanIntensityByScanChrom(intenNC)
> close(intenNC)

[[1]]
[1] 65536

> length(inten.by.chrom)

[1] 6

> 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 gender misannotation or sex chromosome aneuploidies. For the plots, we will create a color coding corresponding to the annotated gender, with blue for males and red for females. We will also load in the SNP annotation file 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] 47  6

> data(affy_scan_annot)
> scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
> # Check to be sure sample ordering is consistent
> allequal(scanAnnot$scanID, rownames(mninten))

[1] TRUE

> # Assign each gender a color
> xcol <- rep(NA, nrow(scanAnnot))
> xcol[scanAnnot$sex == "M"] <- "blue"
> xcol[scanAnnot$sex == "F"] <- "red"
> data(affy_snp_annot)
> snpAnnot <- SnpAnnotationDataFrame(affy_snp_annot)
> 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 file. It is assumed the sample annotation file holds these values. 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 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-gender.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),cex=0.8)
> plot(x2, y2, col=xcol, xlab=xintenlab,
+   ylab="X heterozygosity", main=main2, cex.main=0.8)
> legend("topleft", mf, col=cols, pch=c(1,1),cex=0.8)
> plot(x3, y3, col=xcol, ylab=yintenlab,
+   xlab="X heterozygosity", main=main3, cex.main=0.8)
> legend("topright", mf, col=cols, pch=c(1,1),cex=0.8)
> plot(x4,y4, col="red", xlab="Autosomal heterozygosity",
+   ylab="X heterozygosity", main=main4, cex.main=0.8)
> legend("topleft", mf, col=cols, pch=c(1,1),cex=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. This step calculates identity by descent (IBD) estimates for each population group using an expectation-maximization (EM) approach for maximum likelihood estimation (MLE) of the IBD probabilities `k1` and `k0` [3], and then graphs `k1` versus `k0`. The calculations are done using the `gdsfmt` and `SNPRelate` packages developed as part of the GENEVA project. `gdsfmt` is a library that uses a file format similar to the NetCDF files, and `SNPRelate` performs the IBD calculations using this data format.

[3] Weir, B. S., Anderson, A. D. & Hepler, A. B. Genetic relatedness analysis: modern data and new challenges. *Nature Reviews Genetics* **7**, 771-780 (October 2006).

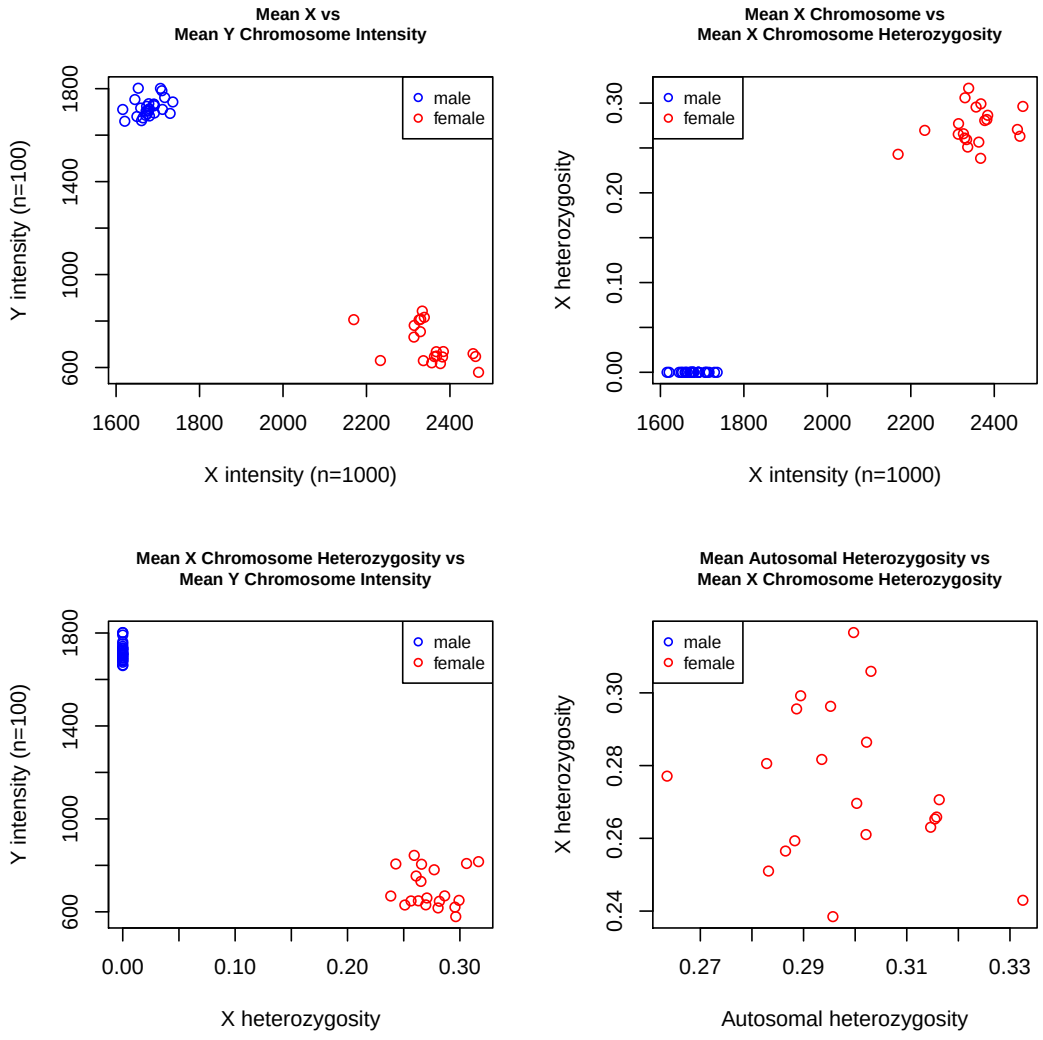


Figure 16: Intensity plots for all samples, with blue indicating an annotated male and red corresponding to annotated females

IBD Using Maximum Likelihood Estimation (MLE)

- We assume the sample annotation file has the `duplicated` variable, while the SNP file needs to have `chromosome` and `missing.n1` variables. We will convert the genotype NetCDF file to the `gds` format with the function `convertNcdfGds`. For more function information, please refer to the function documentation.

```
> library(gdsfmt)
> library(SNPRelate)

SNPRelate: 0.9.1
Streaming SIMD Extensions (SSE) supported.
```

```
> ncfile <- system.file("extdata", "affy_geno.nc",
+   package="GWASdata")
> gdsfile <- "broad.hapmap.geno.gds"
> convertNcdfGds(ncfile, gdsfile)
```

- SNP selection for the IBD estimation analysis has proved to be a salient component of the analysis. One may consider choosing SNPs based upon patterns of linkage, minor allele frequency or position along the genome. Here we simply use autosomal SNPs with missing call rate less than 0.05 to include in the MLE IBD estimation.

```
> snp.auto <- pData(snpAnnot)[snpAnnot$chromosome < 23, ]
> snp.auto.nonmiss <- snp.auto[snp.auto$missing.n1 < 0.05, ]
> snp.select <- snp.auto.nonmiss$snpID
> length(snp.select)

[1] 1963
```

- Next, we perform the IBD estimation for the CEU population group. The `snpgdsIBDMLE` function gives square matrices with rows and columns equal to the number of samples where entries are the `k0` and `k1` coefficient estimates, respectively, for each pairwise combination of samples. Note for larger studies this is a significant analysis since every possible pair of samples must be considered.

```
> sample.sel <- scanAnnot$scanID[scanAnnot$race == "CEU"]
> length(sample.sel)

[1] 29

> gdsobj <- openfn.gds(gdsfile)
> ibd <- snpgdsIBDMLE(gdsobj, sample.id=sample.sel,
+                     snp.id=snp.select, method="EM")

Identity-By-Descent analysis (MLE) on SNP genotypes:
Removing 202 SNPs (monomorphic, < MAF, or > missing rate)
Working space: 29 samples, 1761 SNPs
        Use 1 CPU cores.
MLE IBD:        the sum of all working genotypes = 50550

> closefn.gds(gdsobj)
> dim(ibd$k0)

[1] 29 29

> ibd$k0[1:5,1:5]
```

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

- We can now find the expected relationships between samples in the CEU group 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 `pedigreeClean` is called, which checks for basic pedigree errors such as not all mothers annotated as female or not all fathers as male, or that there exist no individual ids that appear in both the mother and father columns. Please see the function documentation for more information on this function.

```
> samp <- pData(scanAnnot)[scanAnnot$race == "CEU",
+   c("family", "subjectID", "father", "mother", "sex")]
> dim(samp)

[1] 29  5

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

   family   individ    father    mother sex
5    1341 200122600         0         0   M
17   1344 200116780         0         0   M
29   1334 200019634         0         0   F
32   1334 200118596         0         0   M
42   1408 200074814 200094287 200019401   F

> pedigreeClean(samp)

NULL
```

- The functions that determine expected relationships require no duplicates in the pedigree. This will be verified by a function call to `pedigreeFindDuplicates`. We will also call `pedigreeCheck` which determines if there are any singleton families, mothers/fathers whose sex does not match, impossible relationships, or subfamilies.

```
> dups <- pedigreeFindDuplicates(samp)
> dups

$dups.mismatch
[1] family  individ copies
<0 rows> (or 0-length row.names)
```

```
$dups.match
  family   individ copies
1   1341 200099417      2
2   1408 200019401      2
3   1362 200169440      2
```

```
> uni.samp <- pedigreeDeleteDuplicates(samp, dups$dups.match)
> fc <- pedigreeCheck(uni.samp)
```

- The output is a list consisting of vectors and a matrix: `one.person` is vector of family ids for one-person families. `mismatch.sex` consists of family ids where sex of mother and/or father is incorrect. `impossible.related` consists of family ids where either child is mother of self or an individual is both child and mother of same person subfamilies. `ident` is a matrix with family id (families with subfamilies), subfamily identifier, individual ids of persons in the subfamily. (In `subfamilies.ident` the individual id's include individuals identified as mother or father who may not be in `individ`.)

```
> length(fc$one.person)

[1] 0

> length(fc$mismatch.sex)

[1] 0

> length(fc$impossible.related)

[1] 0

> length(fc$subfamilies.ident)

[1] 3
```

- There are multiple subfamilies identified, so we will need to assign new family IDs to the subfamililes.

```
> subf <- fc$subfamilies.ident
> head(subf)

  family subfamily   individ
1   1341         1 200122600
2   1341         1 200015835
3   1341         1 200039107
4   1341         2 200030290
5   1341         2 200191449
6   1341         2 200099417
```

```
> table(subf$family)

1341 1362
   6    6

> subf.ids <- subf$individ[subf$subfamily == 2]
> newfam <- uni.samp$individ %in% subf.ids
> uni.samp$family[newfam] <- uni.samp$family[newfam] + 10000
> table(uni.samp$family)

 1334  1340  1341  1344  1347  1362  1408 11341 11362
    3     3     2     3     3     3     3     3     3

> pedigreeCheck(uni.samp)

NULL
```

- 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 are: UN = unrelated, PO = parent/offspring, FS = full siblings, HS = half siblings, and FC = first cousins. Families where mothers and fathers are related are also looked for among the family annotations. It is this function output with annotated expected relationships that we will compare with the relationship estimates calculated earlier and saved in the R object `ibd`.

```
> rels <- pedigreePairwiseRelatedness(uni.samp)
> length(rels$inbred.fam)

[1] 0

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

   Individ1  Individ2 relation kinship family
1 200122600 200015835       PO    0.25   1341
2 200116780 200071490       PO    0.25   1344
3 200116780 200005043        U    0.00   1344
4 200071490 200005043       PO    0.25   1344
5 200019634 200118596        U    0.00   1334

> table(relprs$relation)

PO  U
17  8
```

- In order to plot the IBD coefficient estimates color coded by expected relationships, some data manipulation must occur. The samples must be coded in terms of subject id and each pair of samples must be annotated with the expected relationship.

```
> relprs$s12 <- paste(relprs$Individ1, relprs$Individ2)
> ibd$subjectID <- scanAnnot$subjectID[is.element(
+   scanAnnot$scanID, ibd$sample.id)]
> sample1.temp <- rep(ibd$subjectID, length(ibd$subjectID))
> sample1.temp <- matrix(sample1.temp, nrow = dim(ibd$k0)[1])
> sample2.temp <- t(sample1.temp)
> ibd$sample1.subjectID <- as.array(sample1.temp)
> ibd$sample2.subjectID <- as.array(sample2.temp)
> ibd$s12 <- paste(ibd$sample1.subjectID, ibd$sample2.subjectID)
> ibd$s21 <- paste(ibd$sample2.subjectID, ibd$sample1.subjectID)
> ibd$rel <- array("U", length(ibd$s12))
> ibd$rel[is.element(ibd$s12, relprs$s12[is.element(relprs$relation,
+   "PO")]) | is.element(ibd$s21, relprs$s12[is.element(
+   relprs$relation, "PO")])] <- "PO"
> ibd$rel[ibd$sample1.subjectID == ibd$sample2.subjectID] <- "Dup"
> table(ibd$rel, exclude=NULL)

 Dup   PO    U <NA>
  35   40  766    0
```

- 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, see Figure 17. The orange line segments span the values for the expected `k0`, `k1` values, ± 2 standard deviations.

IBD Using Method of Moments (MoM)

IBD analysis can also be performed using the PLINK Method of Moments (MoM) approach. For the MoM approach, we use a selection of autosomal SNPs with missing call rate less than 5 percent and minor allele frequency > 0 that are spaced at least 15 kb apart.

- We calculate the allele frequency in the set of samples that will be analyzed (in this case, YRI samples). The function `apartSnpSelection` randomly selects SNPs for which no pair of SNPs is closer than a user specified threshold; we choose 15 kb here.

```
> # Allele frequency calculation in YRI samples
> # open the NetCDF file and create a GenotypeData object
> nc <- NcdfGenotypeReader(ncfile)
> genoData <-  GenotypeData(nc, scanAnnot=scanAnnot, snpAnnot=snpAnnot)
> # sample selection - unduplicated YRI samples
> sample.excl <- scanAnnot$scanID[scanAnnot$race != "YRI" |
+                                 scanAnnot$duplicated]
> length(sample.excl)

[1] 30

> afreq <- alleleFrequency(genoData, scan.exclude=sample.excl)
> close(genoData)
```

```
> ibdPlot(ibd$k0, ibd$k1, relation=ibd$rel,
+   main="HapMap CEU IBD Estimates \nEM Algorithm")
```

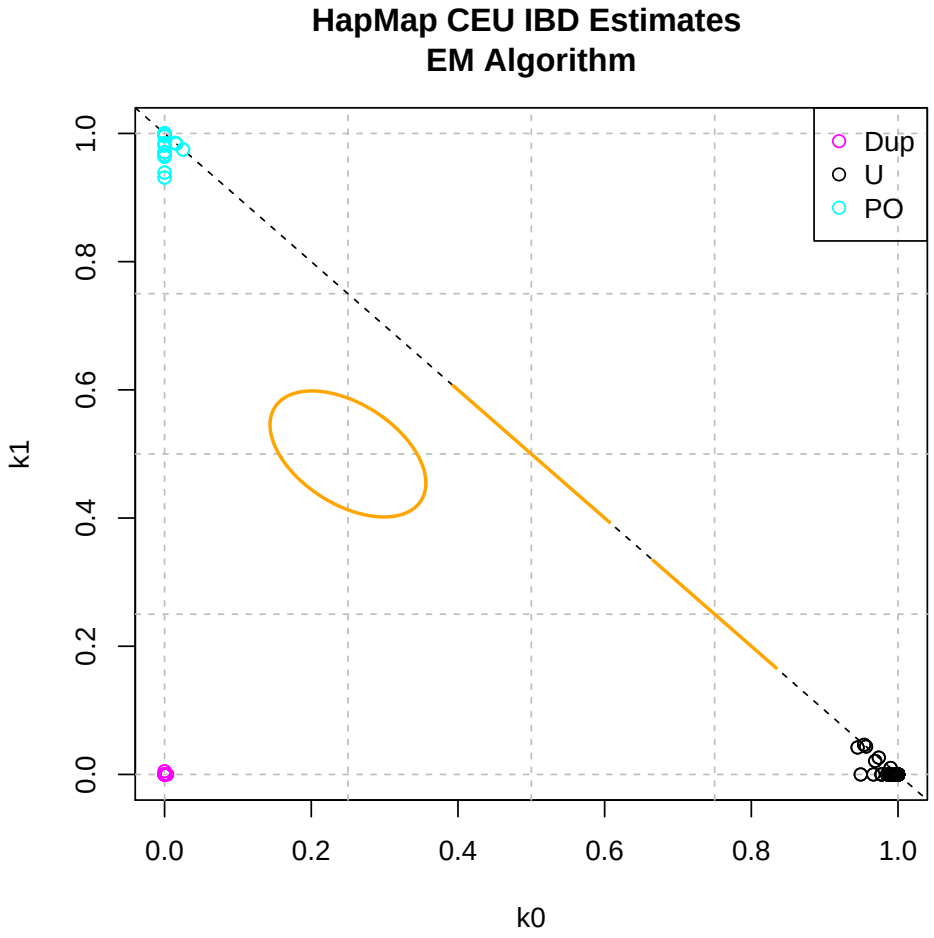


Figure 17: IBD estimates for the CEU samples.

```
[[1]]
[1] 65536

> # Add allele frequency to SNP annotation
> afreq <- afreq[,"all"] # males and females combined
> snpAnnot$A.freq.yri <- afreq
> varMetadata(snpAnnot)["A.freq.yri", "labelDescription"] <-
+   "frequency of allele A in YRI subjects"
> # Initial selection: autosomal SNPs with low missing call rate and MAF>0
> init.sel <- snpAnnot$missing.n1 < 0.05 & snpAnnot$chromosome < 23 &
+   afreq > 0 & afreq < 1
> sum(init.sel)
```

```
[1] 1661

> # Call the function to select SNPs no closer than 15000 bases
> set.seed(1001)
> n <- 15000
> rsnp2 <- apartSnpSelection(snpAnnot$chromosome, snpAnnot$position,
+                            min.dist=n, init.sel=init.sel)
> sum(rsnp2)

[1] 1108

> # Add the SNP selection to the SNP annotation file
> snpAnnot$r15kb <- rsnp2
> varMetadata(snpAnnot)["r15kb", "labelDescription"] <- paste(
+   "Logical indicator for autosomal SNPs with missing.n1<0.05",
+   "and YRI MAF>0, at least 15 kb apart")
```

- We now use the function `snpgdsIBDMoM` to perform IBD for the YRI samples using the PLINK MoM approach.

```
> sample.sel <- scanAnnot$scanID[scanAnnot$race == "YRI"]
> snp.select <- snpAnnot$snpID[snpAnnot$r15kb]
> gdsobj <- openfn.gds(gdsfile)
> ibd <- snpgdsIBDMoM(gdsobj, sample.id=sample.sel,
+                       snp.id=snp.select)

Identity-By-Descent analysis (PLINK method of moment) on SNP genotypes:
Removing 0 SNPs (monomorphic, < MAF, or > missing rate)
Working space: 18 samples, 1108 SNPs
        Use 1 CPU cores.
PLINK IBD:        the sum of all working genotypes = 19921

> closefn.gds(gdsobj)
> dim(ibd$k0)

[1] 18 18

> ibd$k0[1:5,1:5]

          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] 0.0000000 1.0000000 0.9115525 1.0000000 0.9227441
[2,] 1.0000000 0.0000000 0.5881596 0.7764557 0.8061041
[3,] 0.9115525 0.5881596 0.0000000 0.7690751 0.7035948
[4,] 1.0000000 0.7764557 0.7690751 0.0000000 0.8136060
[5,] 0.9227441 0.8061041 0.7035948 0.8136060 0.0000000
```

- We can now find the expected relationships between samples in the YRI group 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. As before, we use the function `pedigreeClean` to check for pedigree errors.

```
> samp <- pData(scanAnnot)[scanAnnot$race == "YRI",
+   c("family", "subjectID", "father", "mother", "sex")]
> dim(samp)

[1] 18  5

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

   family   individ father mother sex
3      28 200150062      0      0   F
14     58 200122151      0      0   F
15      9 200033736      0      0   F
28     28 200003216      0      0   M
50      9 200140995      0      0   M

> pedigreeClean(samp)

NULL
```

- As with the Europeans, we use `pedigreeFindDuplicates` to identify duplicates and and `pedigreeCheck` to look for singleton families, mothers/fathers whose sex does not match, impossible relationships, or subfamilies. In studies with family data, sorting pedigree inconsistencies is a convoluted task and it is important to utilize automated methods to help detect any inconsistencies.

```
> dups <- pedigreeFindDuplicates(samp)
> dups

$dups.mismatch
[1] family  individ copies
<0 rows> (or 0-length row.names)

$dups.match
  family   individ copies
1     12 200160551      2

> uni.samp <- pedigreeDeleteDuplicates(samp, dups$dups.match)
> fc <- pedigreeCheck(uni.samp)
> (subf <- fc$subfamilies.ident)
```

```
  family subfamily   individ
1     58         1 200122151
2     58         2 200105428
```

- There is a subfamily with two unrelated people (likely founders), so we remove this family from the pedigree.

```
> uni.samp <- uni.samp[!(uni.samp$family %in% subf$family),]
> table(uni.samp$family)

 4  5  9 12 28
 3  3  3  3  3

> pedigreeCheck(uni.samp)

NULL
```

- Now from the verified sample list excluding duplicate samples, we can calculate the expected relationships among the samples by calling the function `pedigreePairwiseRelatedness`.

```
> rels <- pedigreePairwiseRelatedness(uni.samp)
> length(rels$inbred.fam)

[1] 0

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

   Individ1  Individ2 relation kinship family
1 200150062 200003216        U    0.00     28
2 200150062 200034659       PO    0.25     28
3 200003216 200034659       PO    0.25     28
4 200033736 200140995        U    0.00      9
5 200033736 200066330       PO    0.25      9

> table(relprs$relation)

PO  U
10  5

> relprs$s12 <- paste(relprs$Individ1, relprs$Individ2)
> ibd$subjectID <- scanAnnot$subjectID[is.element(
+   scanAnnot$scanID, ibd$sample.id)]
> sample1.temp <- rep(ibd$subjectID, length(ibd$subjectID))
> sample1.temp <- matrix(sample1.temp, nrow = dim(ibd$k0)[1])
> sample2.temp <- t(sample1.temp)
> ibd$sample1.subjectID <- as.array(sample1.temp)
```

```
> ibd$sample2.subjectID <- as.array(sample2.temp)
> ibd$s12 <- paste(ibd$sample1.subjectID, ibd$sample2.subjectID)
> ibd$s21 <- paste(ibd$sample2.subjectID, ibd$sample1.subjectID)
> ibd$rel <- array("U", length(ibd$s12))
> ibd$rel[is.element(ibd$s12, relprs$s12[is.element(relprs$relation,
+   "PO")]) | is.element(ibd$s21, relprs$s12[is.element(
+   relprs$relation, "PO")])] <- "PO"
> ibd$rel[ibd$sample1.subjectID == ibd$sample2.subjectID] <- "Dup"
> table(ibd$rel, exclude=NULL)

 Dup   PO    U <NA>
  20   22  282    0
```

- 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, see Figure 18. Several samples in the YRI population show higher than expected relatedness.[4]

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 ethnicity, we can identify any individuals whose recorded self-identified 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.

- We must load the SNP and sample annotation files to select autosomal SNPs with values in `missing.n1` less than 0.05. Here we use all SNPs that meet these criteria, but it is generally advisable to select a set of SNPs within which each pair has a low level of linkage disequilibrium (e.g. $r^2 < 0.2$). We must also ensure no duplicate samples are used for the principal component calculations. A color code based upon sample continental ancestry is assigned for plotting; CEU is coded as black, YRI as red. The `snpgdsPCA` function is called with the SNP and sample subsets to calculate the first 32 eigenvectors. To read more about this function, please see the function documentation.

```
> snp.sel <- snpAnnot$snpID[snpAnnot$missing.n1 < 0.05 &
+                           snpAnnot$chromosome < 23]
> sample.sel <- scanAnnot$scanID[scanAnnot$duplicated == FALSE]
> gdsobj <- openfn.gds(gdsfile)
> pca <- snpgdsPCA(gdsobj, sample.id=sample.sel,
+                  snp.id=snp.sel)
```

[4] A second generation human haplotype map of over 3.1 million SNPs, supplementary information. The International HapMap Consortium, *Nature* **449**, 851-861 (18 October 2007).

```
> ibdPlot(ibd$k0, ibd$k1, relation=ibd$rel,
+   main="HapMap YRI IBD Estimates \nPLINK MOM Approach")
```

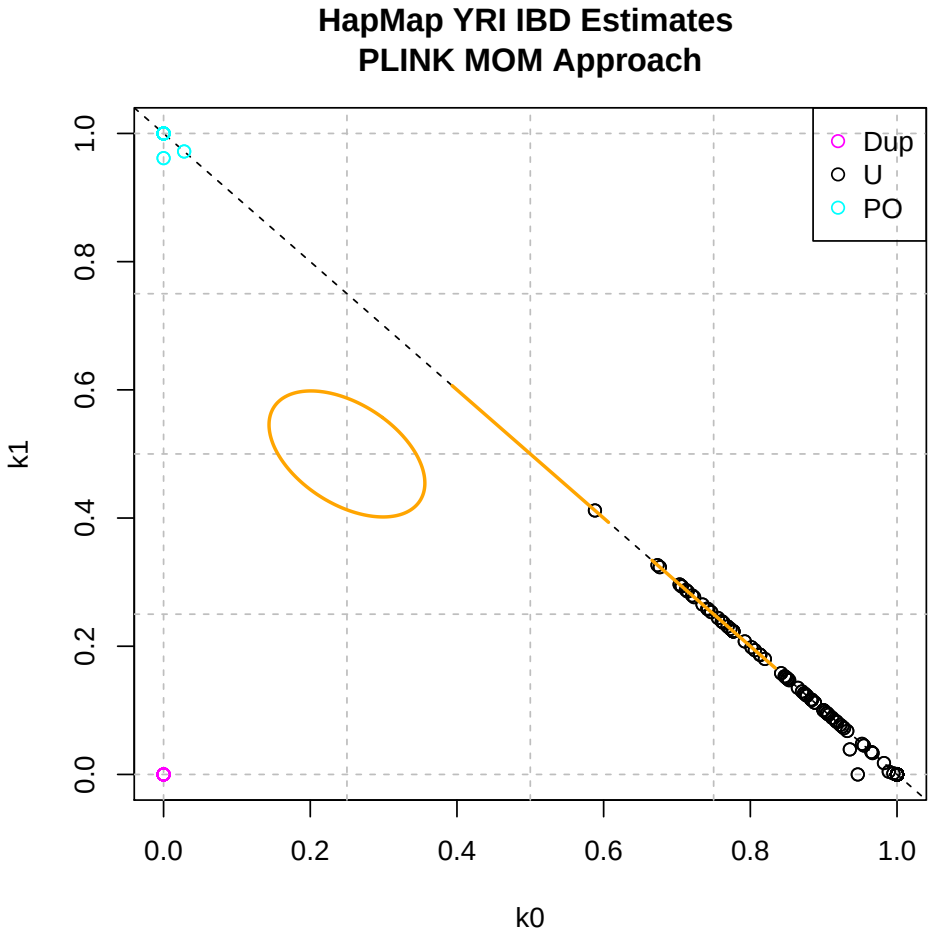


Figure 18: Method of Moment (MoM) estimates for the YRI samples. The yellow bars are the expected positions (+/- 2 SD) for full siblings, half siblings and first cousins.

```
Principal Component Analysis (PCA) on SNP genotypes:
Removing 34 SNPs (monomorphic, < MAF, or > missing rate)
Working space: 43 samples, 1929 SNPs
        Use 1 CPU cores.
PCA:        the sum of all working genotypes = 82146
PCA:        Mon Oct 31 23:32:14 2011        Begin (eigenvalues and eigenvectors)
PCA:        Mon Oct 31 23:32:14 2011        End (eigenvalues and eigenvectors)

> closefn.gds(gdsobj)
> names(pca)
```

```
[1] "sample.id" "snp.id"    "eigenval"  "eigenvect" "TraceXTX"  "Bayesian"
[7] "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$race == "CEU"] <- "black"
> cols[samp$race == "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. Figure 20 shows this plot for all samples, colored by ethnicity. The genetic diversity in the YRI group is apparent in the later eigenvectors, while the remaining groups remain in clusters throughout.

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

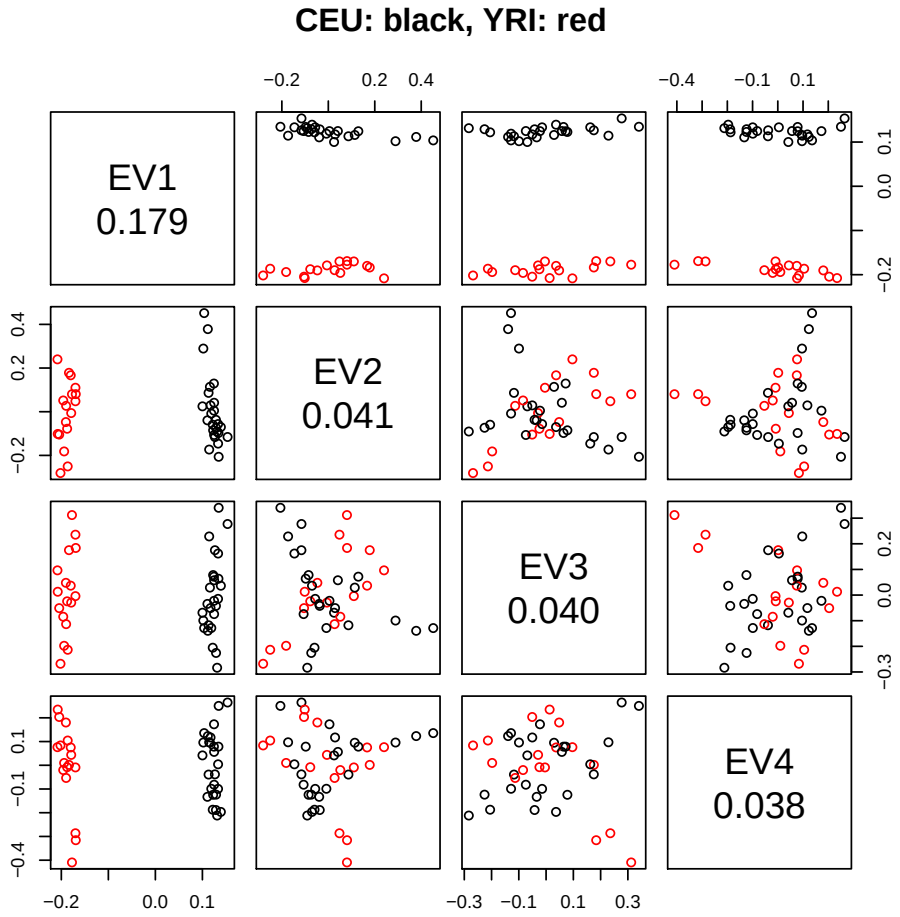


Figure 19: First four eigenvectors calculated from the principal components analysis.

```
> 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: black, 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 = "."))
```

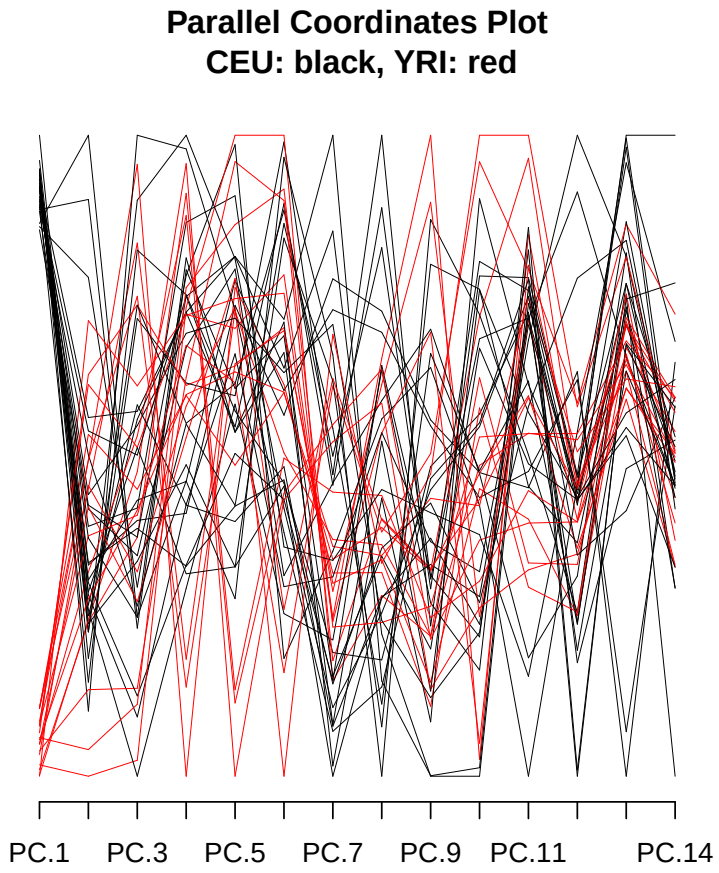


Figure 20: Parallel coordinates plot of first 14 eigenvectors, colored by ethnicity.

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. It is clear to see the first two eigenvectors hold the largest percentage of variance among the population, although the total variance accounted for is still less the one-quarter of the total.

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

 [1] 17.8731497  4.0574681  3.9585417  3.8104927  3.6380240  3.6258353
 [7]  3.5497928  3.4758859  3.3797425  3.2797279  3.2434189  3.0479071
[13]  2.9600071  2.5249894  2.4476519  2.3350368  2.2863042  2.2264246
[19]  2.1647080  2.1281596  2.0644437  1.9390781  1.9332345  1.8964952
[25]  1.8668741  1.8247075  1.7828902  1.7547660  1.0881429  0.7167384
[31]  0.6979683  0.6716671

> 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] <- "red"
```

- Plot the principal component pairs for the first three PCs, by case-control status.
- Do 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")
```

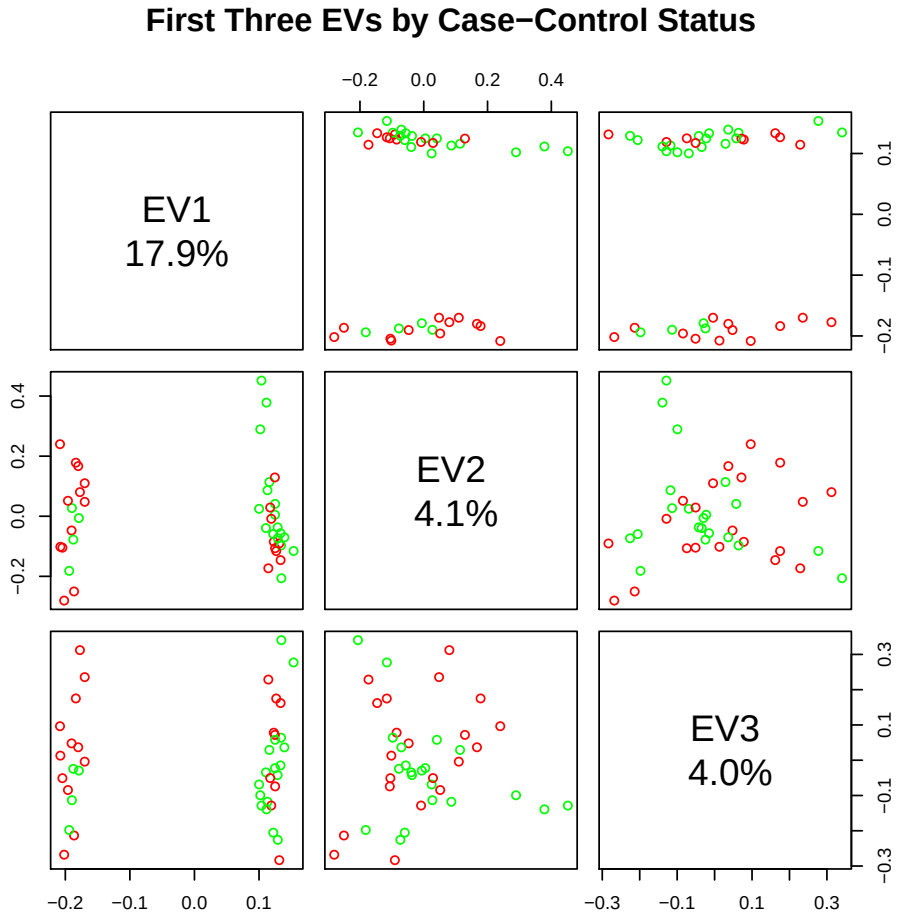


Figure 21: The values from the first three eigenvectors plotted for each sample, color coded by case-control status. Green denotes case status and red indicates a control.

```
> 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$race                            1 0.9647  0.9647 6137.368 <2e-16 ***
princomp$case.ctrl.status                1 0.0000  0.0000    0.003  0.959
princomp$race:princomp$case.ctrl.status  1 0.0000  0.0000    0.149  0.702
Residuals                               38 0.0060  0.0002
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

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

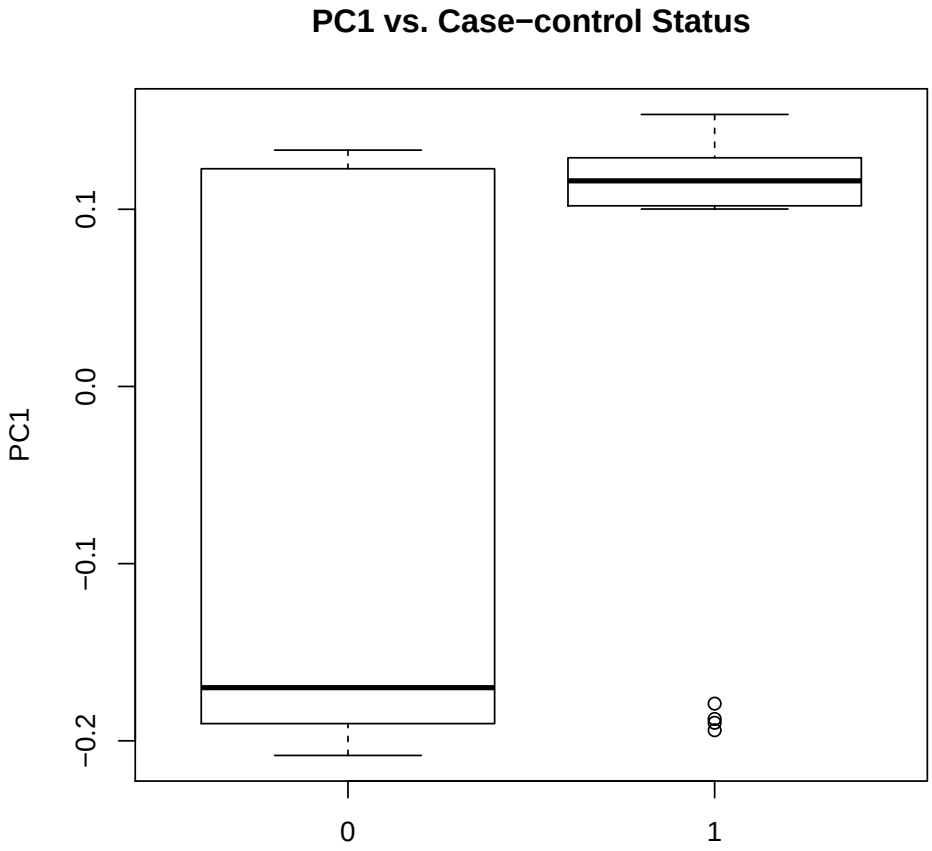


Figure 22: A boxplot of the values of the first eigenvector stratified by case-control status, where 0 indicates a control and 1 indicates a case.

```
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$race                         1 0.0014 0.00140   0.058  0.811
princomp$case.ctrl.status             1 0.0171 0.01707   0.706  0.406
princomp$race:princomp$case.ctrl.status  1 0.0558 0.05582   2.308  0.137
Residuals                            38 0.9191 0.02419
1 observation deleted due to missingness
```

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

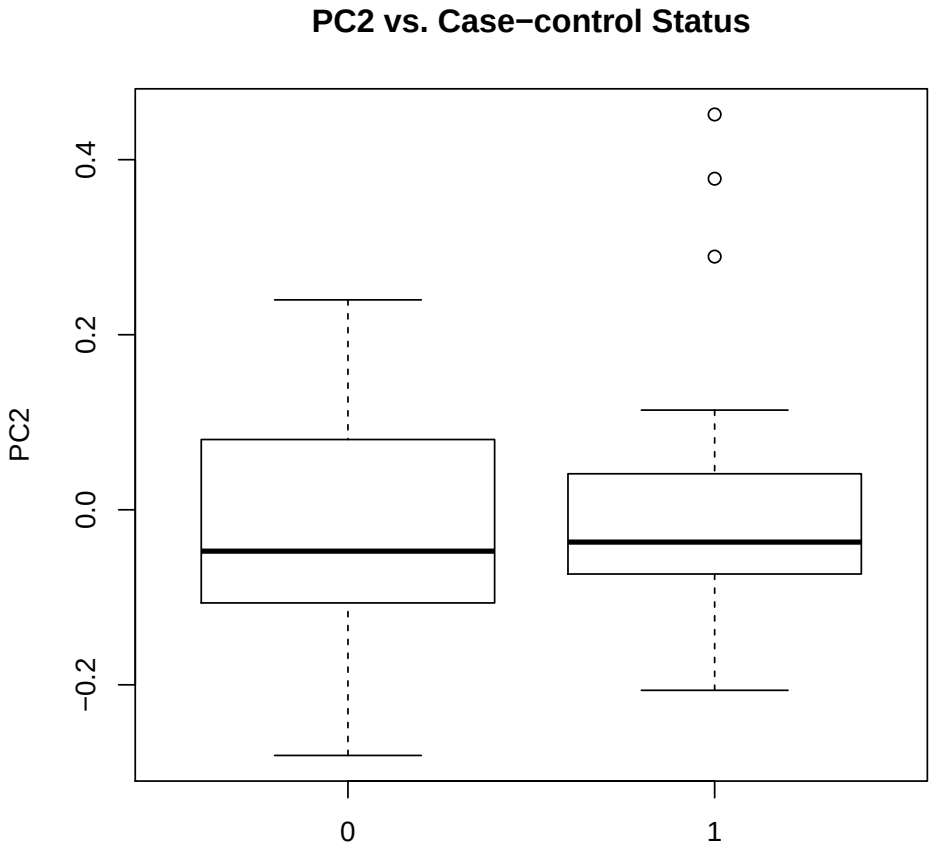


Figure 23: A boxplot of the values of the second eigenvector stratified by case-control status, where 0 indicates a control and 1 indicates a case.

```
> 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$race                          1 0.0000 0.00000   0.000  0.998
princomp$case.ctrl.status              1 0.0361 0.03610   1.496  0.229
princomp$race:princomp$case.ctrl.status  1 0.0123 0.01233   0.511  0.479
Residuals                             38 0.9170 0.02413
1 observation deleted due to missingness
```

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

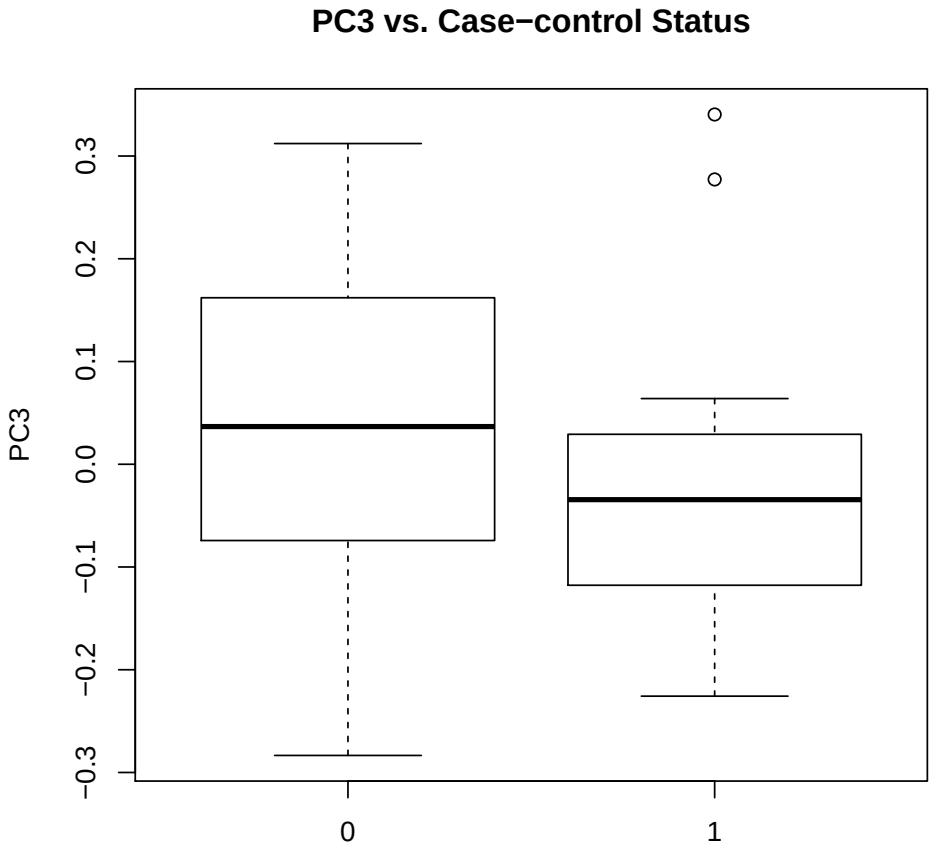


Figure 24: A boxplot of the values of the third eigenvector stratified by case-control status, where 0 indicates a control and 1 indicates a case.

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 0.000129 0.0001288   0.527  0.472
Residuals         41 0.010015 0.0002443
4 observations deleted due to missingness

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

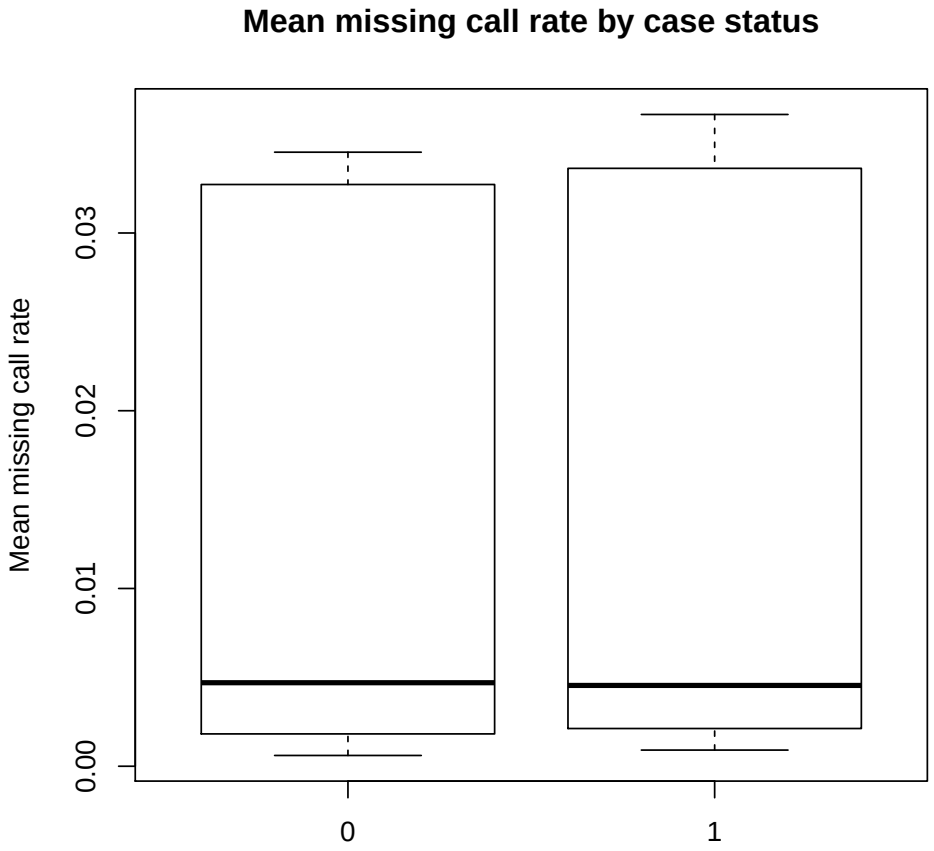


Figure 25: A boxplot showing the mean missing call rate stratified by case-control status, where 0 indicates a control and 1 indicates a case.

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.

```
> library(GWASTools)
> library(GWASdata)
> data(illumina_scan_annot)
> scanAnnot <- ScanAnnotationDataFrame(illumina_scan_annot)
> data(illumina_snp_annot)
> snpAnnot <- SnpAnnotationDataFrame(illumina_snp_annot)
> blfile <- system.file("extdata", "illumina_bl.nc", package="GWASdata")
> blnc <- NcdfIntensityReader(blfile)
> blData <-  IntensityData(blnc, scanAnnot=scanAnnot, snpAnnot=snpAnnot)
> genofile <- system.file("extdata", "illumina_geno.nc", package="GWASdata")
> genonc <- NcdfGenotypeReader(genofile)
> genoData <-  GenotypeData(genonc, scanAnnot=scanAnnot, snpAnnot=snpAnnot)
```

- 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 <- snpAnnot$chromosome
> pos <- snpAnnot$position
> build <- getAttribute(blnc, attname="genome_build")
> if (build == 36) {
+   data(HLA.hg18)
+   hla <- chrom == 6 & pos >= HLA.hg18$start.base & pos <= HLA.hg18$end.base
+   data(pseudoautosomal.hg18)
+   xtr <- chrom == 23 & pos >= pseudoautosomal.hg18["X.XTR", "start.base"] &
+     pos <= pseudoautosomal.hg18["X.XTR", "end.base"]
+   data(centromeres.hg18)
+   centromeres <- centromeres.hg18
+ } else if (build  ==  37) {
+   data(HLA.hg19)
+   hla <- chrom == 6 & pos >= HLA.hg19$start.base & pos <= HLA.hg19$end.base
+   data(pseudoautosomal.hg19)
+   xtr <- chrom == 23 & pos >= pseudoautosomal.hg19["X.XTR", "start.base"] &
+     pos <= pseudoautosomal.hg19["X.XTR", "end.base"]
+   data(centromeres.hg19)
```

```
+   centromeres <- centromeres.hg19
+ }
> ignore <- snpAnnot$missing.n1 == 1 #ignore includes intensity-only and failed snps
> snp.exclude <- ignore | hla | xtr
> snp.ok <- snpAnnot$snpID[!snp.exclude]
```

- We use circular binary segmentation to find change points in B Allele Frequency (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 LogRRatio. Change points in LogRRatio 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

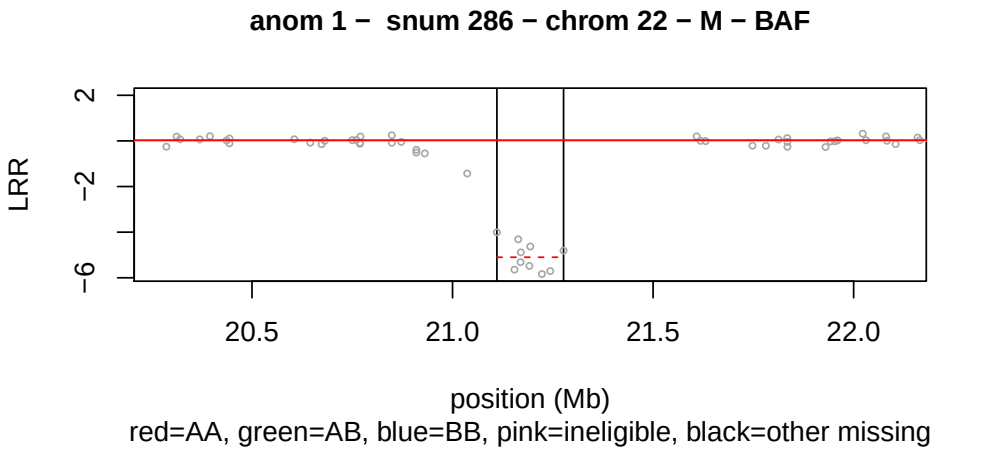
- Calculate statistics for the anomalous segments found with the BAF and LOH methods.

```
> # 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)
```



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")]
```

- 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)

[[1]]
[1] 65536

> close(genoData)

[[1]]
[1] 131072
```

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 with regard to 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.

```
> library(GWASTools)
> library(GWASdata)
> data(affy_scan_annot)
> scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
> 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.

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

[1] 0

> genofile <- system.file("extdata", "affy_geno.nc",
+   package="GWASdata")
> genoNC <- NcdfGenotypeReader(genofile)
> genoData <- GenotypeData(genoNC, 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 869828          0     4           0            0
2 869844          0     4           0            0
3 869864          0     4           0            0
4 869889          0     4           0            0
5 869922          0     4           0            0
6 869925          0     4           0            0

> length(dupdisc$discordance.by.subject)

[1] 4

> dupdisc$discordance.by.subject[[1]]

            106         107
106 0.000000000 0.001258653
107 0.001258653 0.000000000

> # 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.
0.0003150 0.0005348 0.0007605 0.0007737 0.0009994 0.0012590

> # Assign colors for the duplicate samples based on population group.
> dat$col <- NA
> dat$col[dat$pop == "CEU"] <- "black"
> dat$col[dat$pop == "YRI"] <- "red"
> table(dat$col, exclude=NULL)

black   red  <NA>
    3     1     0
```

```
     > dat <- dat[order(dat$disc),]
     > dat$rank <- 1:npair

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

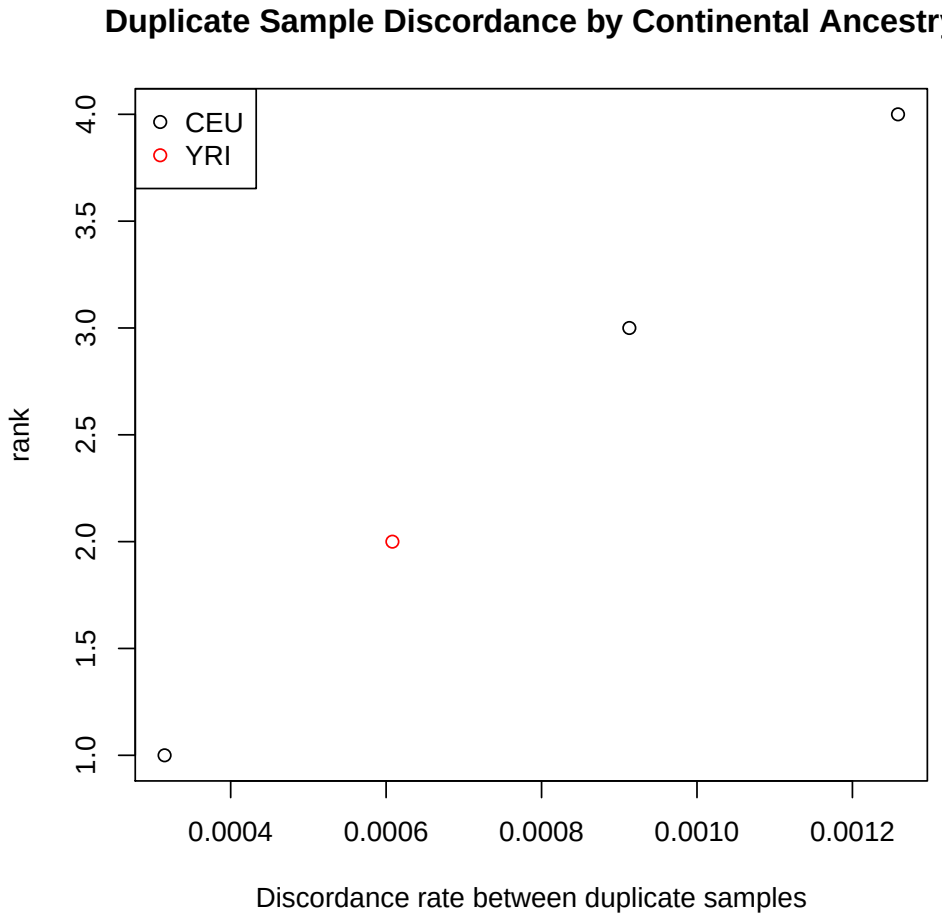


Figure 26: The discordance for duplicate samples plotted in increasing values, color coded by population group.

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

  offspring father mother
1       138    127    101
2       151    160    103
3       214     50     15
4       272    237    191
5       272    263    191
6       239     28      3

> dim(res)

[1] 18  3

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

[1] 50

> 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           0    3147         0       100    0    0    0    0
2      5 200102386           7    3132         0        99    0    0    0    0
3      9 200066330           3    3146         1        99    0    0    0    0
```

```
4     12 200013233           1    3139         0       100    0    0    0    0
5     28 200034659           5    3136         0       100    0    0    0    0
6   1334 200016815           2    3145         0       100    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
2     0     0     0     3     2    2     0    0
3     0     0     0     2     1    0     0    0
4     0     0     0     1     0    0     0    0
5     0     0     0     2     2    1     0    0
6     0     0     0     0     2    0     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. We expect to get some `NA` values in this calculation for those probes for which there were no valid trios to check for errors, causing the rate to have a denominator of zero.

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

FALSE  TRUE  <NA>
   31  3219     0
```

- We will plot the error rate per SNP as calculated above, sorting the values by increasing rate.
- 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 that the maximum number of errors per SNP. Next, we can look at subsets of SNPs with greater than 0, 1 and 2 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
> fam[is.na(err)] <- NA
> table(is.na(fam), exclude=NULL)
```

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

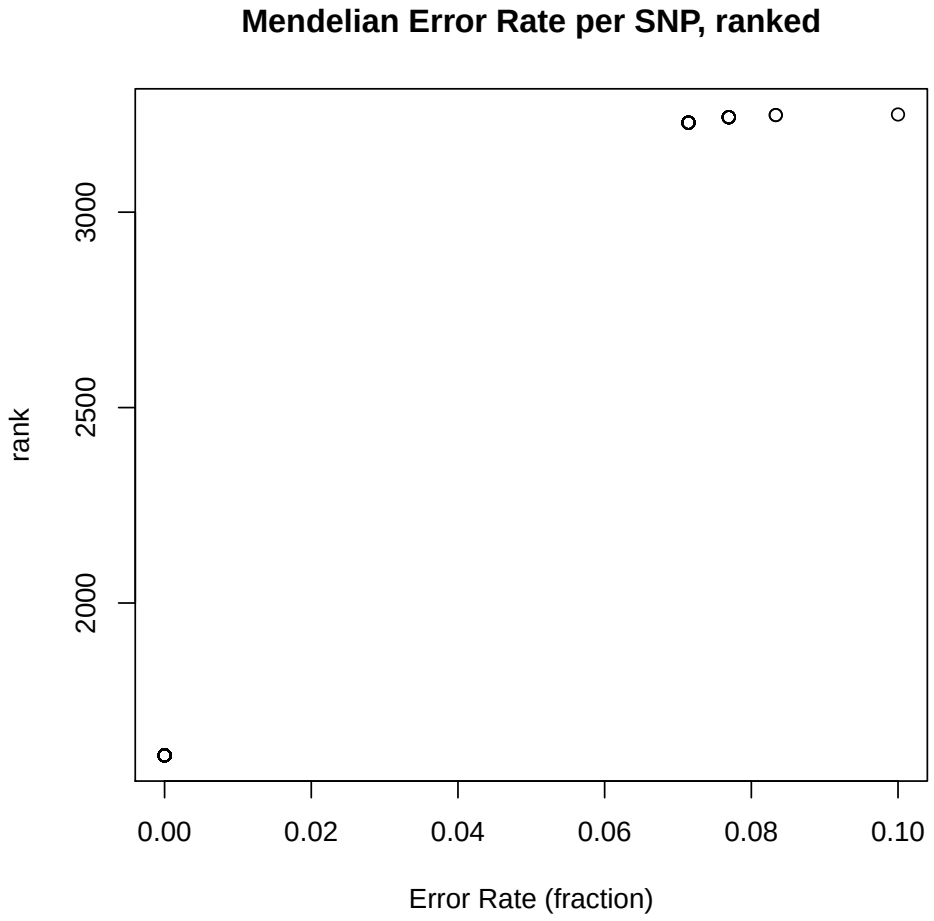


Figure 27: A plot of the Mendelian error rate per SNP, ranked in increasing order.

```
FALSE  <NA>
 3250     0

> summary(fam)

    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.000000 0.000000 0.000000 0.009538 0.000000 1.000000

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

[1] 31
```

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

[1] 0

> length(fam[n > 0 & fam > 2])

[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] 0.9904615
```

- A new version of the SNP annotation table will be saved with the Mendelian error values included. 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`.

```
> # Write the error rates to the SNP table
> 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] 50

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

[1] 50

> # Save the updated SNP annotation table with
> #   the new Mendelian error data
> 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 a cluster plot for 3 SNPs with the highest Mendelian error rate. We expect the plots to lack defined genotype clusters, leading to a poor call rate. For this example, we will plot the top 3 SNPs using the function `genoClusterPlotByBatch`.

```
> # 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),]
> snp <- snp[1:3,]
> xyfile <- system.file("extdata", "affy_qxy.nc",
+   package="GWASdata")
> xyNC <- NcdfIntensityReader(xyfile)
> xyData <- IntensityData(xyNC, 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))
> genoClusterPlotByBatch(xyData, genoData, snpID=snp$snpID, main.txt=mtxt,
+                        plot.type="XY", batchVar="plate", 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 0.0002026 0.0004820 0.0006509 0.0009536 0.0022350

> # 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
```

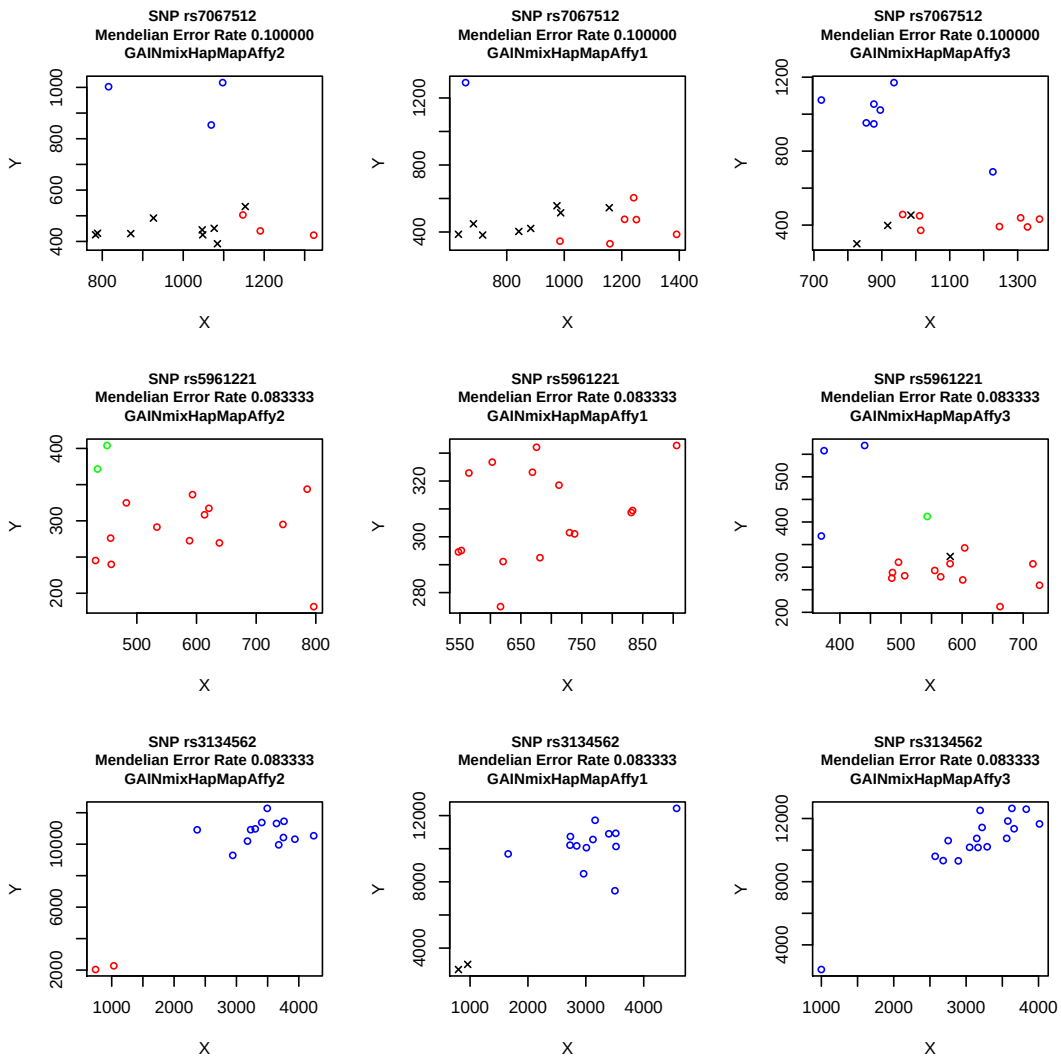


Figure 28: Cluster plots of the 3 SNPs with the highest Mendelian error rate. Blue indicates a sample with a "BB" genotype, green is an "AB" genotype and red is an "AA" genotype. The black X marks indicate a sample with a missing genotype for that SNP. All of these cluster plots are clearly problematic with regard to the genotype clusters and the called genotypes.

```
> 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$Mend.err.rate.fam[i]
+ }
> allequal(scanAnnot$scanID, sort(scanAnnot$scanID))

[1] TRUE
```

```
> head(scanAnnot$mend.err.rate.fam)

[1] 0.0015943878 0.0001639344           NA 0.0009535919 0.0009535919
[6] 0.0015943878

> 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,exclude=NULL)

 CEU  YRI <NA>
   7    5    0

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

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 `gwasExactHW` for the samples with European continental ancestry only, although the process is just the same for all population groups. This function filters out the males for the X chromosome and all samples for the Y and mitochondrial probes. Note that an optional argument to `gwasExactHW` is a a scan by chromosome by filter matrix, to exclude scan-chromosome combinations with aneuploidy.

```
> 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("black", "red"))
```

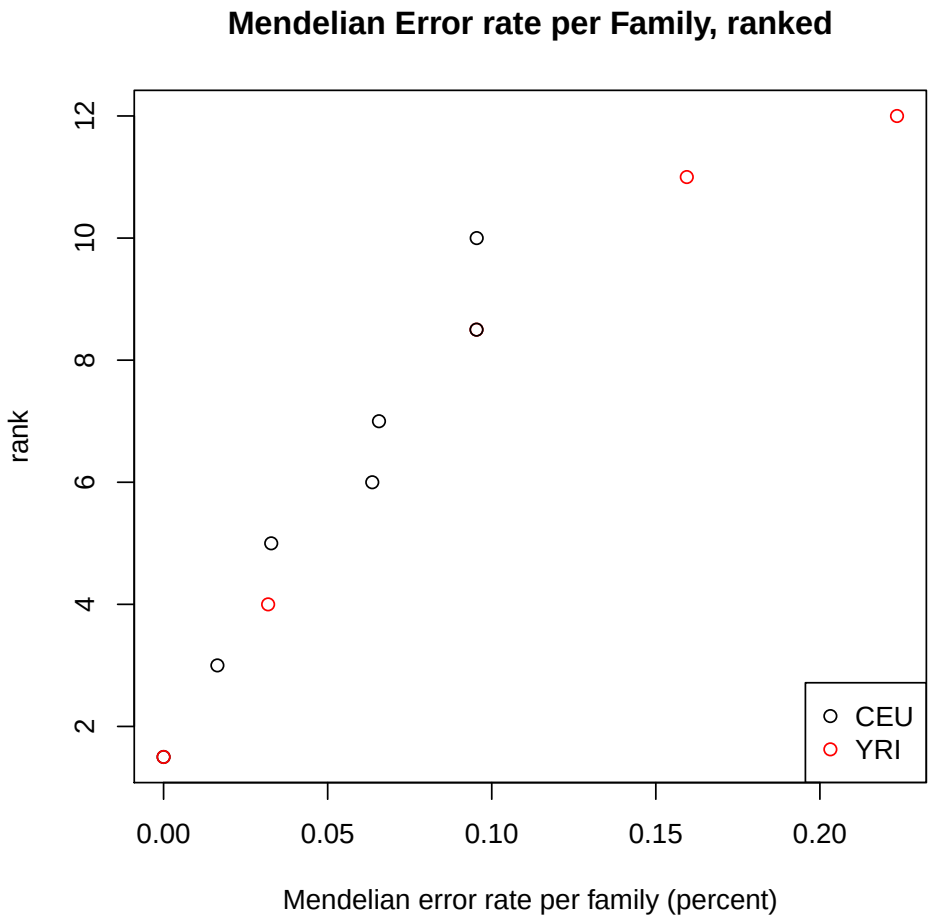


Figure 29: A plot of the Mendelian error rate for families, ranked in order of increasing rate. The black indicates a family with CEU continental ancestry and red indicates a family with YRI ancestry.

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

   father mother
3       0      0
5       0      0
14      0      0
```

```
15      0      0
17      0      0
28      0      0

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

nonfounders
FALSE  TRUE
   33    14

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

[1] 30

> hwe <- gwasExactHW(genoData, scan.exclude=scan.excl)
> close(genoData)

[[1]]
[1] 65536
```

- We will examine the CEU population data for the purposes of the tutorial. The remaining population groups and their results are analyzed in exactly the same manner.

  We will look at the values calculated from the function call to `gwasExactHW`, which include p-values, minor allele frequency, and genotype counts for each SNP on each of the chromosome types. All the p-values are missing for the Y and M chromosomes. There are minor allele frequencies on the same chromosome types; we look at the first 50 values. Finally, there are genotype counts for all chromosome types, which are stored as `nAA, nAB` and `nBB`.

```
> names(hwe)

 [1] "snpID"        "chromosome"   "position"     "nAA"          "nAB"
 [6] "nBB"          "MAF"          "minor.allele" "f"            "p.value"

> dim(hwe)

[1] 3300   10

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

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  13.00   17.00   17.00   16.95   17.00   17.00
```

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

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.000   8.000   8.000   7.963   8.000   8.000

> hwe$p.value[1:10]

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

> sum(is.na(hwe$p.value[hwe$chromosome == 24])) # XY

[1] 4

> sum(is.na(hwe$p.value[hwe$chromosome == 23])) # X

[1] 264

> hwe$MAF[1:10]

 [1] 0.05882353 0.00000000 0.11764706 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
869828   0   2  15
869844  17   0   0
869864  13   4   0
```

- 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
 -0.7778  -0.2000  -0.0667  -0.0412   0.1005   1.0000 682.0000


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

[1] 12 11

> summary(chkf$MAF)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.05882 0.05882 0.06066 0.10750 0.12500 0.25000
```

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

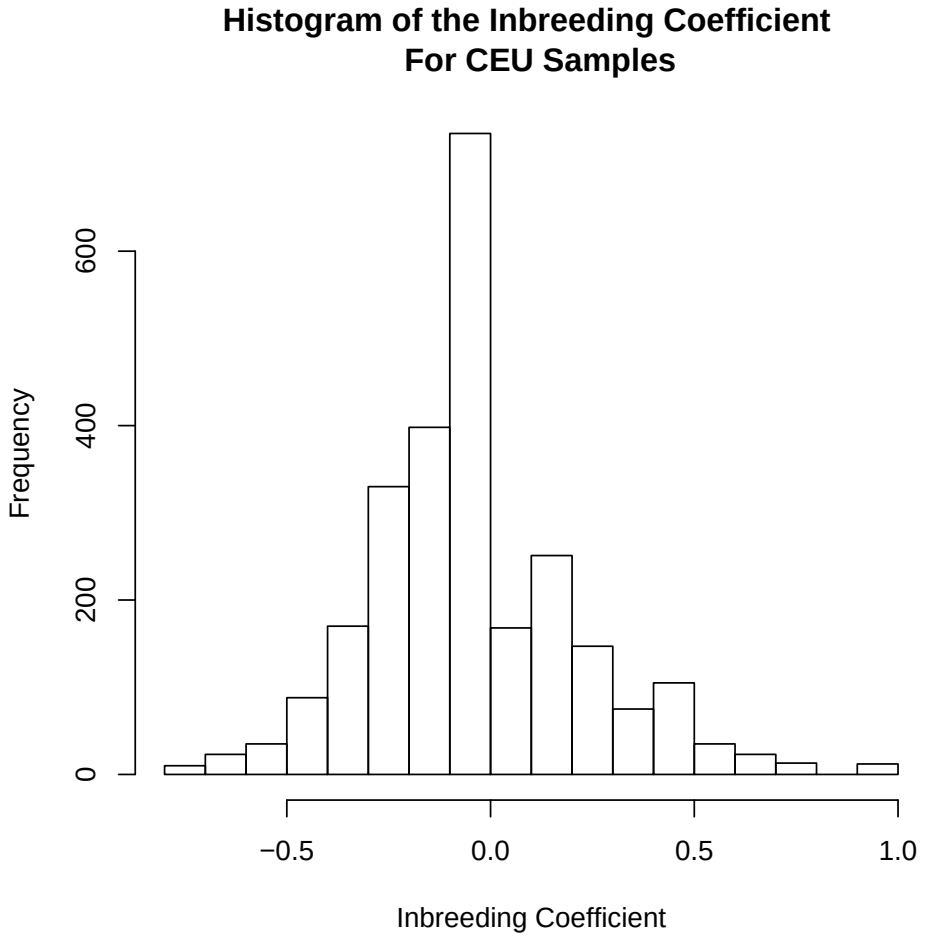


Figure 30: A histogram of the inbreeding coefficient values from running the Hardy-Weinberg analysis for the unrelated CEU samples.

- 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. Make a plot of the observed p-values vs. the expected p-values for the autosomes and X chromosome separately by calling the function `qqPlot`.

```
> idx <- which(hwe$MAF == 0); length(idx)

[1] 481

> hwe.0 <- hwe[-idx,]; dim(hwe.0)

[1] 2819   11
```

```
> # Check to makes sure that the correct number of SNPs were removed
> length(hwe$p.value) - length(hwe.0$p.value)

[1] 481

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

[1] 1786

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

[1] 1786

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

[1] 737

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

[1] 736

> 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()
```

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

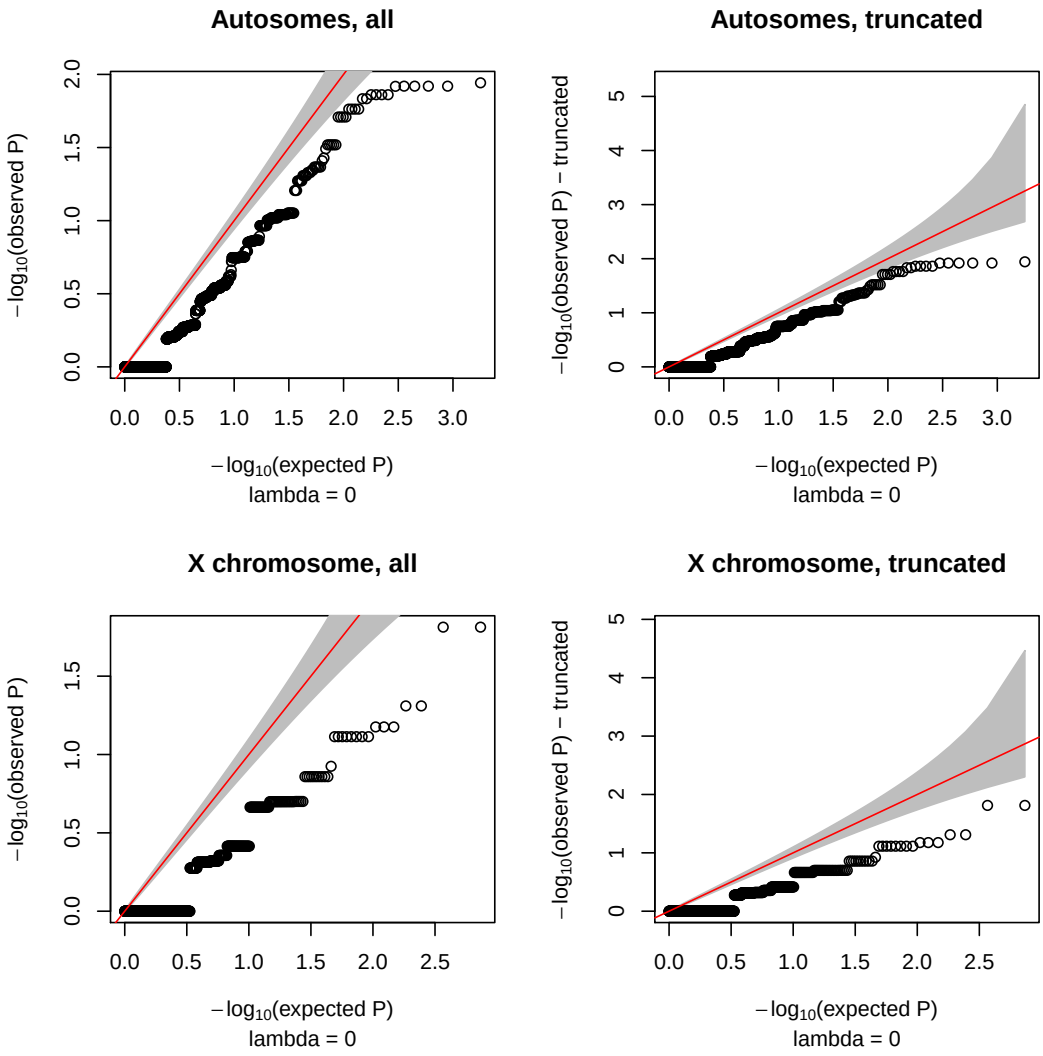


Figure 31: Four QQ-plots showing the deviation of Hardy-Weinberg p-values for each SNP for the samples from the CEU population group.

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

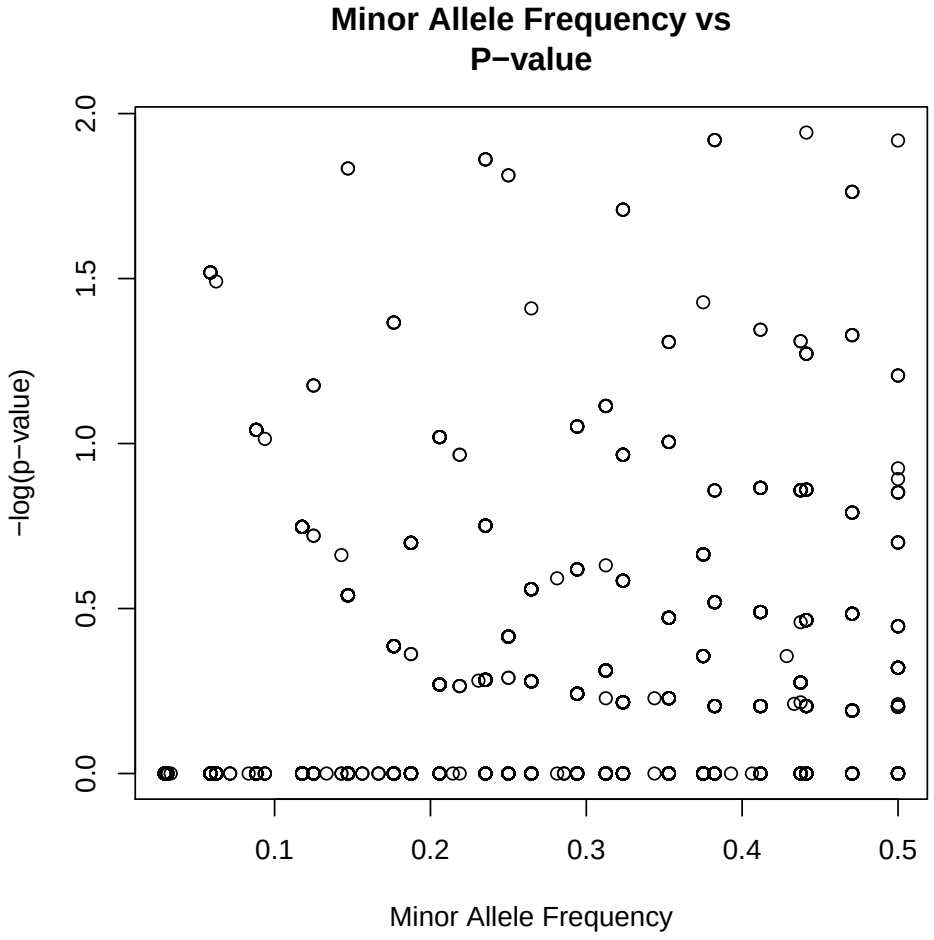


Figure 32: A plot of the p-value as a function of the minor allele frequency for SNPs for the CEU samples, excluding the SNPs with a minor allele frequency equal to 0.

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, regional association 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 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 `assocTestRegression`. For the tutorial, all samples will be kept, but we would use the argument `scan.exclude` for those samples we wish to filter out for the association test. 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.

```
> library(GWASTools)
> library(GWASdata)
> data(affy_scan_annot)
> scanAnnot <- ScanAnnotationDataFrame(affy_scan_annot)
> genofile <- system.file("extdata", "affy_geno.nc",
+   package="GWASdata")
> genoNC <- NcdfGenotypeReader(genofile)
> genoData <- GenotypeData(genoNC, scanAnnot=scanAnnot)
> assoc.file <- "assoc"
> assocTestRegression(genoData, outcome="status", covar.list=c(""),
+   ivar.list=c(""), model.type="logistic", robust=TRUE,
+   chromosome.set=c(24:26), outfile=assoc.file,)
```

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

- To create QQ plots of the ordered p-values from the association tests versus the ordered expected p-values, we first load the saved p-values data. Then, we will call `qqPlot` to plot the Wald test p-values. Given that the samples were split randomly between cases and controls, not surprisingly there are no outliers visible in the QQ plot in Figure 33.

```
> assoc <- getobj(paste(assoc.file,
+   ".model.1.additive.chr.24_26.RData", sep=""))
> names(assoc)

 [1] "snpID"                              "MAF"
 [3] "minor.allele"                       "model.1.additive.n"
 [5] "model.1.additive.warningOrError" "model.1.additive.Est.G"
 [7] "model.1.additive.SE.G"              "model.1.additive.OR.G"
 [9] "model.1.additive.OR_L95.G"          "model.1.additive.OR_U95.G"
[11] "model.1.additive.Stat.G"            "model.1.additive.pvalue.G"
[13] "model.1.nAA.cc0"                    "model.1.nAB.cc0"
[15] "model.1.nBB.cc0"                    "model.1.nAA.cc1"
[17] "model.1.nAB.cc1"                    "model.1.nBB.cc1"
```

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. See the resulting plot in Figure 34.

```
> data(affy_snp_annot)
> snpAnnot <- SnpAnnotationDataFrame(affy_snp_annot)
> chrom <- getChromosome(snpAnnot)
> chrom.sel <- chrom %in% 24:26
```

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 Likelihood Ratio test.

```
> # Identify SNPs with lowest p-values
> snp <- pData(snpAnnot)[chrom.sel, c("snpID", "rsID")]
> allequal(snp$snpID, assoc$snpID)
> snp$pval <- assoc$model.1.additive.pvalue.G
> snp <- snp[order(snp$pval),]
> snp <- snp[1:9,]
> xyfile <- system.file("extdata", "affy_qxy.nc",
+   package="GWASdata")
> xyNC <- NcdfIntensityReader(xyfile)
> xyData <- IntensityData(xyNC, snpAnnot=snpAnnot, scanAnnot=scanAnnot)
> pdf(file="DataCleaning-cluster.pdf")
> par(mfrow = c(3,3))
> mtxt <- paste("SNP", snp$rsID, "\np =", format(snp$pval, digits=4))
> genoClusterPlot(xyData, genoData, snpID=snp$snpID, main.txt=mtxt,
```

```
> qqPlot(pval=assoc$model.1.additive.pvalue.G,
+   truncate=TRUE, main="QQ Plot of Wald Test p-values")
```

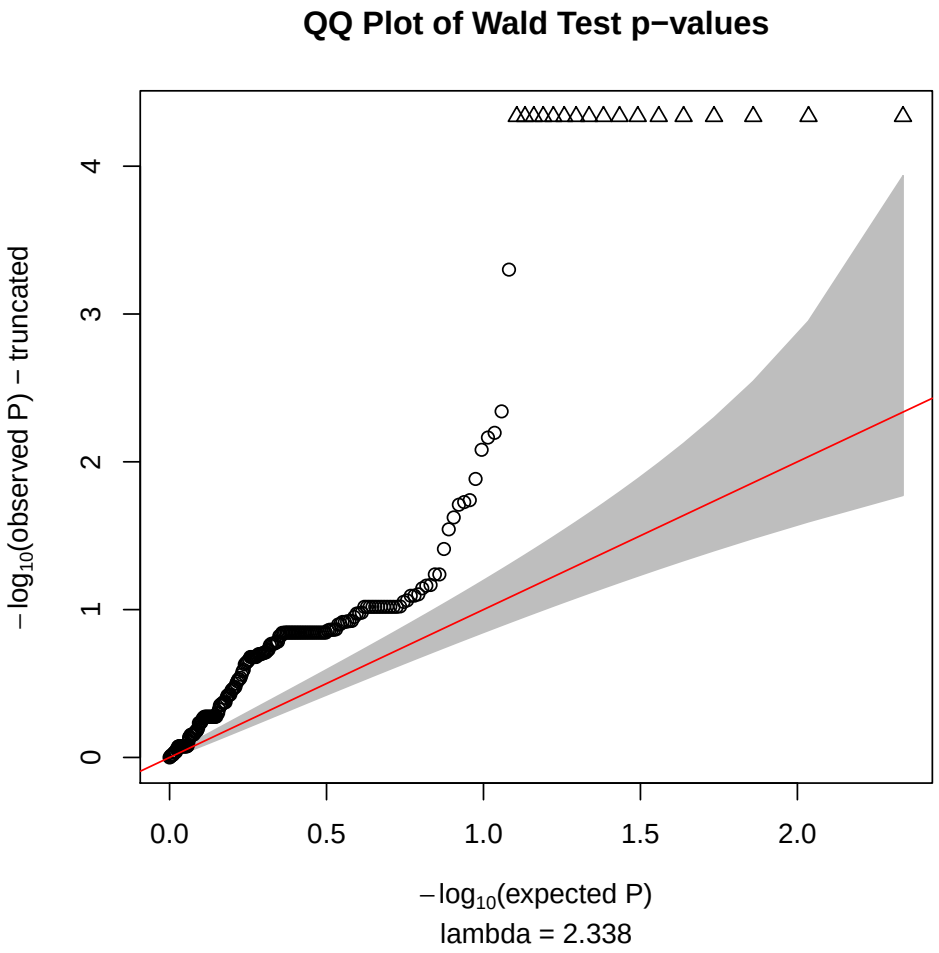


Figure 33: QQ plot of Wald statistic p-values for association test.

```
+                plot.type="XY")
> dev.off()
> close(xyData)
> close(genoData)
```

The cluster plots in Figure 35 show that some of the SNPs have poor cluster plots, indcating genotyping artifact. In a real association study, the evidence from such SNPs would be discounted.

```
> manhattanPlot(assoc$model.1.additive.pvalue.G,
+   chromosome=chrom[chrom.sel],
+   chrom.labels=c("XY","Y","M"))
```

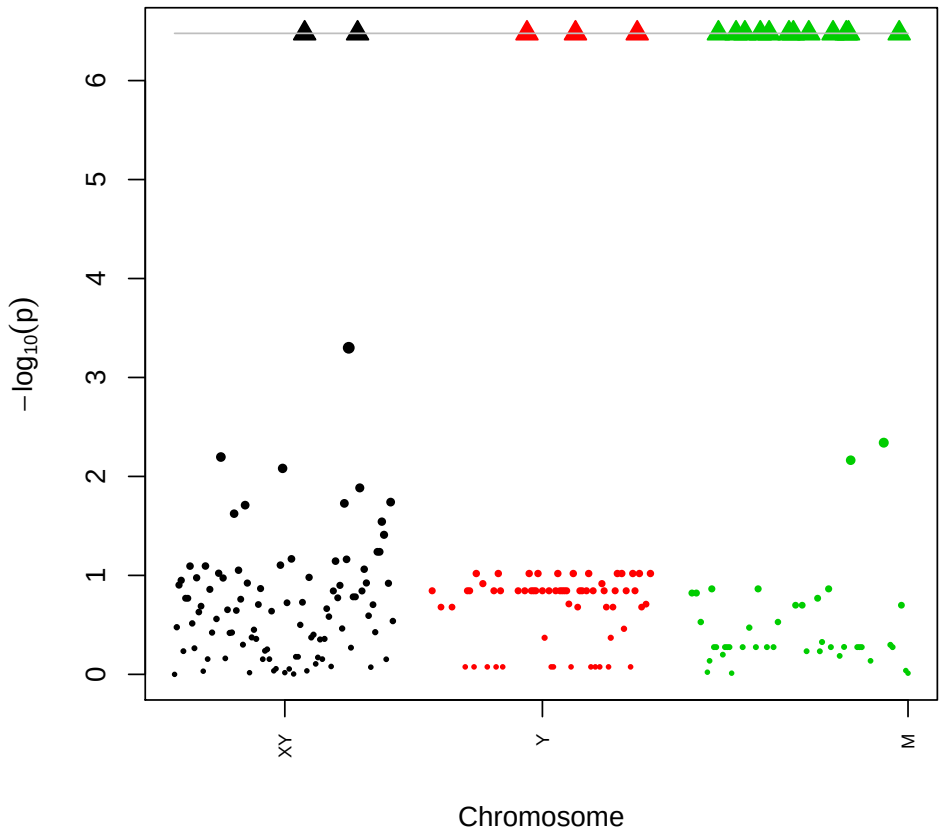


Figure 34: "Manhattan" plot of Wald statistic p-values for association test.

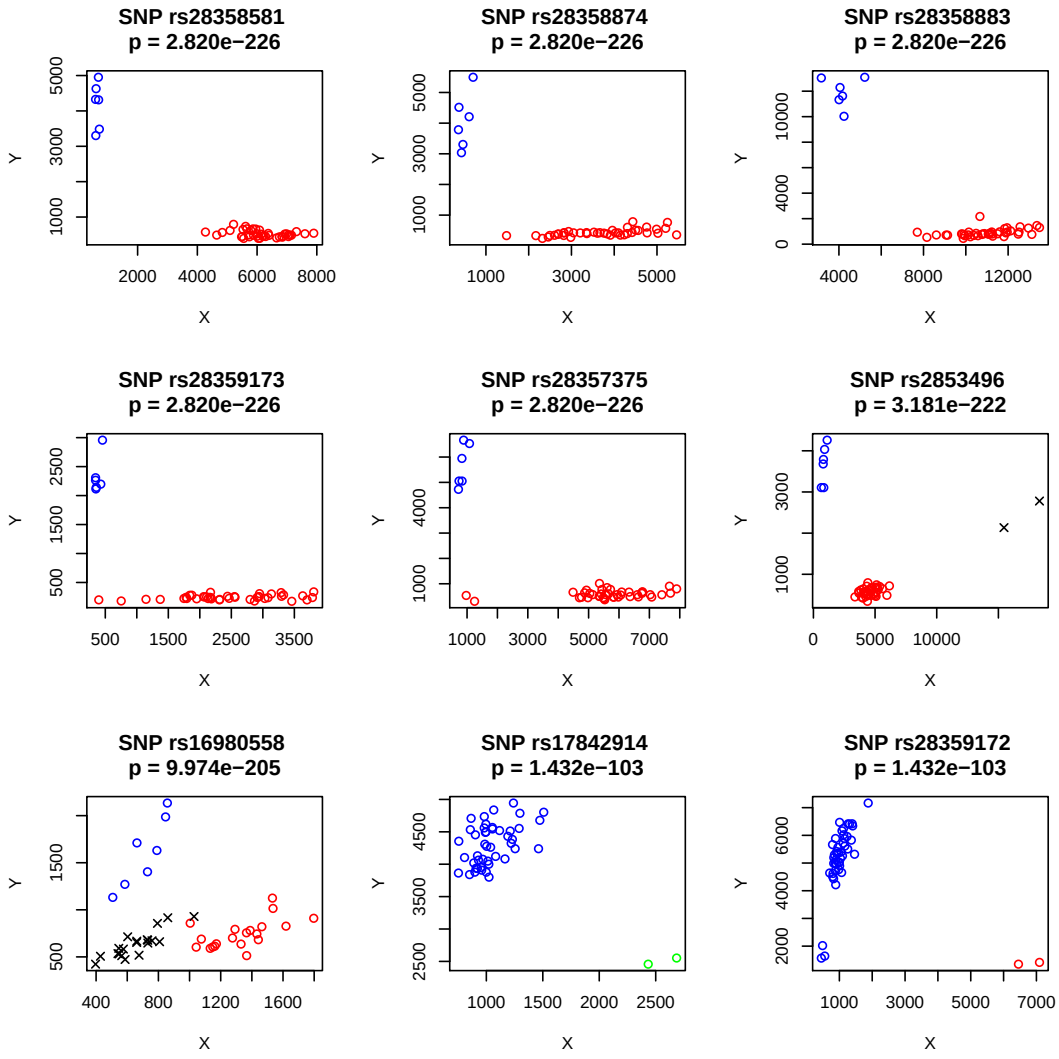


Figure 35: Cluster plots of the 9 SNPs with smallest p-values.

10 Acknowledgements

This manual reflects the work of many people. In the first place the methods described were developed and implemented by a team headed by Cathy Laurie. The team included David Crosslin, Stephanie Gogarten, David Levine, Caitlin McHugh, Sarah Nelson, Jess Shen, Bruce Weir, Qi Zhang and Xiuwen Zheng. Before any the work started, valuable advice was provided by Thomas Lumley and Ken Rice. Preparation of the manual began with a team headed by Ian Painter and Stephanie Gogarten. The team included Marshall Brown, Matthew Conomos, Patrick Danaher, Kevin Rubenstein, Emily Weed and Leila Zelnick.

The data cleaning activities of the GENEVA Coordinating Center have been greatly helped by the experience and advice from other participants in the GENEVA program: the genotyping centers at CIDR and the Broad; the dbGaP group at the National Center for Biotechnology Information (NCBI); and the many study investigators. Particular thanks to Kim Doheny and Elizabeth Pugh at CIDR and Stacey Gabriel and Daniel Mirel at the Broad and Justin Paschall at NCBI.

Funding for the GENEVA project includes HG 004446 (PI: Bruce Weir) for the Coordinating Center, U01 HG 004438 (PI: David Vallee) for CIDR, HG 004424 (PI: Stacey Gabriel) for the Broad.

The continuing guidance of Dr. Teri Manolio of NHGRI is deeply appreciated.