# Preparing Affymetrix Data

GENEVA Coordinating Center
Department of Biostatistics
University of Washington

February 27, 2014

## 1 Overview

This vignette describes how to prepare Affymetrix data for use in GWASTools. After following the examples here for prepapring the data, the analysis procedure is the same as presented in the GWAS Data Cleaning vignette.

## 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.*

Note that for Affymetrix data, the rs ID is often not unique, as there may be multiple probes for a given SNP. The probe ID is usually the unique SNP identifier. Also, Affymetrix annotation generally has a separate column or columns to indicate pseudoautosomal regions; some manipulation is usually required to include this information within the chromosome column itself.

```
> 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"

> # data
> head(pData(snpAnnot))

   snpID chromosome position      rsID       probeID
1 869828         21 13733610 rs3132407 SNP_A-8340403
2 869844         21 13852569 rs2775671 SNP_A-8340413
3 869864         21 14038583 rs2775018 SNP_A-8340427
4 869889         21 14136579 rs3115511 SNP_A-8340440
5 869922         21 14396024 rs2822404 SNP_A-8340775
6 869925         21 14404476 rs1556276 SNP_A-1968967

> # 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
```

# 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"

> # data
> head(pData(scanAnnot))

  scanID subjectID family father mother CoriellID race sex status
1      3 200150062     28      0      0   NA18912  YRI   F      0
2      5 200122600   1341      0      0   NA07034  CEU   M      1
3     14 200122151     58      0      0   NA19222  YRI   F      0
4     15 200033736      9      0      0   NA18508  YRI   F      1
5     17 200116780   1344      0      0   NA12056  CEU   M      1
6     28 200003216     28      0      0   NA18913  YRI   M      0
                                          genoRunID              plate
1 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A03_31250 GAINmixHapMapAffy2
2 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A05_31282 GAINmixHapMapAffy2
3 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B02_31236 GAINmixHapMapAffy2
4 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B03_31252 GAINmixHapMapAffy2
5 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B05_31284 GAINmixHapMapAffy2
6 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_C04_31270 GAINmixHapMapAffy2
                                                                 alleleFile
1 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A03_31250.BIRDSEED.ALLELE_SUMMARY.TXT
2 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A05_31282.BIRDSEED.ALLELE_SUMMARY.TXT
3 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B02_31236.BIRDSEED.ALLELE_SUMMARY.TXT
4 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B03_31252.BIRDSEED.ALLELE_SUMMARY.TXT
5 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B05_31284.BIRDSEED.ALLELE_SUMMARY.TXT
6 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_C04_31270.BIRDSEED.ALLELE_SUMMARY.TXT
                                                              chpFile
1 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A03_31250.BIRDSEED.CHP.TXT
2 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A05_31282.BIRDSEED.CHP.TXT
3 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B02_31236.BIRDSEED.CHP.TXT
4 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B03_31252.BIRDSEED.CHP.TXT
5 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_B05_31284.BIRDSEED.CHP.TXT
6 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_C04_31270.BIRDSEED.CHP.TXT

> # 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
```

# 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 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. 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

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")
```

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))
> names(col.nums) <- c("snp", "geno")
> # 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.

```
> 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)
```

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.

### Creating the Intensity NetCDF file

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")
```

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.

```
> 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)
```

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 by the user. For a thorough explanation and presentation of an application of these values, please refer to Peiffer, Daniel A., et al. (2006).[1]

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

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

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

$\theta$ 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). 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 $\theta$ 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 $\theta$ "cluster" value is calculated for each probe, for each of the three genotype clusters, resulting in $\theta_{AA}, \theta_{AB}$ and $\theta_{BB}$ for every probe. Then the $\theta$ value for each sample, call it $\theta_n$, is compared to $\theta_{AA}, \theta_{AB}$ and $\theta_{BB}$. The BAlleleFrequency is calculated

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

A $\theta_n$ value of 0 or 1 corresponds to a homozygote genotype for sample $n$ at that particular probe, and a $\theta_n$ 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.

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

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

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

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")
```

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")
> close(xyNC)
```

```
> close(genoNC)
> # 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)
```