Copy number estimation and genotype calling with crlmm

Rob Scharpf

November 14, 2010

Abstract

This vignette estimates copy number for HapMap samples on the Affymetrix 6.0 platform. See [1] for additional details.

1 Copy number estimation

1.1 Set up

```
> library(crlmm)
```

Several genotyping platforms are currently supported. Supported platforms must have a corresponding annotation package (installed separately) that are listed below.

```
> pkgs <- annotationPackages()
> pkgs <- pkgs[grep("Crlmm", pkgs)]
> pkgs
```

```
 [1] "genomewidesnp6Crlmm"    "genomewidesnp5Crlmm"
 [3] "human370v1cCrlmm"       "human370quadv3cCrlmm"
 [5] "human550v3bCrlmm"       "human650v3aCrlmm"
 [7] "human610quadv1bCrlmm"   "human660quadv1aCrlmm"
 [9] "human1mduov3bCrlmm"     "humanomni1quadv1bCrlmm"
```

Note that 'pd.genomewidesnp6' and 'genomewidesnp6Crlmm' are both annotation packages for the Affymetrix 6.0 platform available on Bioconductor, but the 'genomewidesnp6Crlmm' annotation package must be used for copy number estimation. The annotation package is specified through the 'cdfName' – the identifier without the 'Crlmm' postfix. In the following code, we specify the cdf name for Affymetrix 6.0, provide the complete path to the CEL files, and indicate where intermediate files from the copy number estimation are to be saved.

```
> cdfName <- "genomewidesnp6"
> pathToCels <- "/thumper/ctsa/snpmicroarray/hapmap/raw/affy/1m"
> if (getRversion() < "2.13.0") {
     rpath <- getRversion()
 } else rpath <- "trunk"
> outdir <- paste("/thumper/ctsa/snpmicroarray/rs/ProcessedData/crlmm/",
     rpath, "/copynumber_vignette", sep = "")
> dir.create(outdir, recursive = TRUE, showWarnings = FALSE)
```

All long computations are saved in the output directory `outdir`. Users should change these variables as appropriate. The following code chunk should fail unless the above arguments have been set appropriately.

```
> if (!file.exists(outdir)) stop("Please specify a valid directory for storing output")
> if (!file.exists(pathToCels)) stop("Please specify the correct path to the CEL files")
```

Processed data from codechunks requiring long computations are saved to disk by wrapping function calls in the `checkExists` function. After running this vignette as a batch job, subsequent calls to `Sweave` will load the saved computations from disk. See the `checkExists` help file for additional details.

1.2 Preprocessing and genotyping.

In the following code chunk, we provide the complete path to the Affymetrix CEL files and define a 'batch' variable. The `batch` variable will be used to initialize a container for storing the normalized intensities, the genotype calls, and the parameter estimates for copy number. Often the chemistry plate or the scan date of the array is a useful surrogate for batch. For the HapMap CEL files in our analysis, the CEPH (C) and Yoruban (Y) samples were prepared on separate chemistry plates. In the following code chunk, we extract the population identifier from the CEL file names and assign these identifiers to the variable `batch`.

```
> celFiles <- list.celfiles(pathToCels, full.names = TRUE,
      pattern = ".CEL")
> celFiles <- celFiles[substr(basename(celFiles), 13, 13) %in%
      c("C", "Y")]
> batch <- as.factor(substr(basename(celFiles), 13, 13))
```

The preprocessing steps for copy number estimation includes quantile normalization of the raw intensities for each probe and a step that summarizes the intensities of multiple probes at a single locus. For example, the Affymetrix 6.0 platform has 3 or 4 identical probes at each polymorphic locus and the normalized intensities are summarized by a median. For the nonpolymorphic markers on Affymetrix 6.0, only one probe per locus is available and the summarization step is not needed.

After preprocessing the arrays, the crlmm package estimates the genotype and provides a confidence score at each polymorphic locus. Unless the dataset is small (e.g., fewer than 50 samples), we suggest installing and loading the R package ff to reduce the RAM required for preprocessing and genotyping. Loading the ff package at this point will automatically enable large data support (LDS).

The function `genotype` checks to see whether the ff is loaded. If loaded, the normalized intensities and genotype are stored as `ff` objects on disk. Otherwise, the genotypes and normalized intensities are stored in matrices. A word of caution: the `genotype` function without ff requires a potentially large amount of RAM. A more RAM-friendly approach to preprocessing and genotyping requires the ff package. In particular, the functions `ocProbesets` and `ocSamples` can be used to manage how many probesets and samples are to processed at a time and can therefore be used to fine tune the needed RAM for a particular job. The function `ldPath` indicates that `ff` objects will be stored in the directory `outdir`.

```
> library(ff)
> ldPath(outdir)
> ocProbesets(1e+05)
> ocSamples(200)
```

With LDS enabled, we preprocess and genotype 180 samples from the CEPH and Yoruban populations. Users interested only in the genotypes should instead use the R function `crlmm` or `crlmm2`. We wrap the call to `genotype` in `checkExists` so that subsequent calls to `Sweave` can be run interactively.

```
> if (!file.exists(file.path(outdir, "cnSet.rda"))) {
      gtSet <- checkExists("gtSet", .path = outdir, .FUN = genotype,
          filenames = celFiles, cdfName = cdfName, batch = batch)
 }
```

The value returned by genotype is an instance of the class `CNSet`. In addition to the normalization and genotyping, the `genotype` function initializes a container that will store summary statistics for the batches and parameters needed for copy number estimation. At this point, the batch summaries and parameters for copy number are all NA's.

1.3 Copy number estimation.

The `crlmmCopynumber` performs the following steps:

- computes summary statistics for each batch

- imputes unobserved genotype centers (for each batch)
- shrinks the within-genotype variances
- estimates parameters for allele-specific copy number

With `verbose=TRUE`, the above steps for CN estimation are displayed during the processing.

```
> GT.CONF.THR <- 0.9
> cnSet <- checkExists("cnSet", .path = outdir, .FUN = crlmmCopynumber,
     object = gtSet, GT.CONF.THR = GT.CONF.THR)
> invisible(open(cnSet))
```

In an effort to reduce I/O, the crlmmCopynumber function no longer stores the allele-specific estimates of copy number as part of the object. Rather, several functions are available that will compute relatively quickly the allele-specific estimates from the stored normalized intensities and the linear model parameters. At allele k, marker i, sample j, and batch p, the estimate of allele-specific copy number is computed by subtracting the estimated background from the observed intensity and scaling by the slope coefficient.

$$\hat{c}_{k,ijp} \quad = \quad \max\left\{\frac{1}{\hat{\phi}_{k,ip}}\left(I_{k,ijp} - \hat{\nu}_{k,ip}\right),\ 0\right\} \text{ for } k \in \{A, B\}. \tag{1}$$

See [1] for details.

For large datasets, the above computation will not be instantaneous as the I/O can be substantial. The functions `CA`, `CB`, and `totalCopynumber` should be used to extract CN estimates from the `CNSet` container. The following code chunks illustrate several examples, as well as some of the useful accessors for extracting which markers are SNPs, which are chromosomes, etc.

```
> snp.index <- which(isSnp(cnSet) & !is.na(chromosome(cnSet)))
> ca <- CA(cnSet, i = snp.index, j = 1:5)
> cb <- CB(cnSet, i = snp.index, j = 1:5)
> ct <- ca + cb
```

Alternatively, total copy number can be obtained by

```
> ct2 <- totalCopynumber(cnSet, i = snp.index, j = 1:5)
> stopifnot(all.equal(ct, ct2))
```

At nonpolymorphic loci, either the `CA` or `totalCopynumber` functions can be used to obtain estimates of total copy number.

```
> marker.index <- which(!isSnp(cnSet))
> ct <- CA(cnSet, i = marker.index, j = 1:5)
> stopifnot(all(CB(cnSet, i = marker.index, j = 1:5) ==
     0))
> ct2 <- totalCopynumber(cnSet, i = marker.index, j = 1:5)
> stopifnot(all.equal(ct, ct2))
```

Nonpolymorphic markers on chromosome X:

```
> require(RColorBrewer)
> bp.cols <- brewer.pal(8, "Paired")[c(3, 4)]
> npx.index <- which(chromosome(cnSet) == 23 & !isSnp(cnSet))
> M <- sample(which(cnSet$gender == 1), 5)
> F <- sample(which(cnSet$gender == 2), 5)
> cols <- bp.cols[cnSet$gender[c(M, F)]]
> cn.M <- CA(cnSet, i = npx.index, j = M)
```

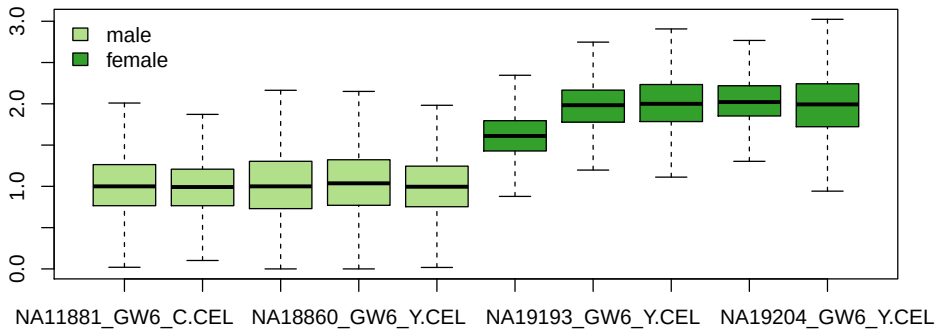

Figure 1: Copy number estimates for nonpolymorphic loci on chromosome X (5 men, 5 women). crlmm assumes that the median copy number across samples at a given marker on X is 1 for men and 2 for women.

```
> cn.F <- CA(cnSet, i = npx.index, j = F)
> boxplot(data.frame(cbind(cn.M, cn.F)), pch = ".", col = cols,
    outline = FALSE)
> legend("topleft", bty = "n", fill = bp.cols, legend = c("male",
    "female"))
```

Polymorphic markers on chromosome X:

```
> X.markers <- which(isSnp(cnSet) & chromosome(cnSet) ==
    23)
> ca.M <- CA(cnSet, i = X.markers, j = M)
> cb.M <- CB(cnSet, i = X.markers, j = M)
> ca.F <- CA(cnSet, i = X.markers, j = F)
> cb.F <- CB(cnSet, i = X.markers, j = F)
> cn.M <- ca.M + cb.M
> cn.F <- ca.F + cb.F
> boxplot(data.frame(cbind(cn.M, cn.F)), pch = ".", outline = FALSE,
    col = cols, xaxt = "n", ylim = c(0, 5))
> legend("topleft", bty = "n", fill = bp.cols, legend = c("male",
    "female"))
> cn2 <- totalCopynumber(cnSet, i = X.markers, j = c(M,
    F))
> stopifnot(all.equal(cbind(cn.M, cn.F), cn2))
```

Accessors for physical position and chromosome are also provided. In the following codechunk we extract the position and chromosome for the first 10 markers in the `cnSet` object.

```
> position(cnSet)[1:10]

 [1] 1156131 2234251 2329564 2553624 2936870 2951834 3095126 3165267
 [9] 3302871 3705226

> chromosome(cnSet)[1:10]

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

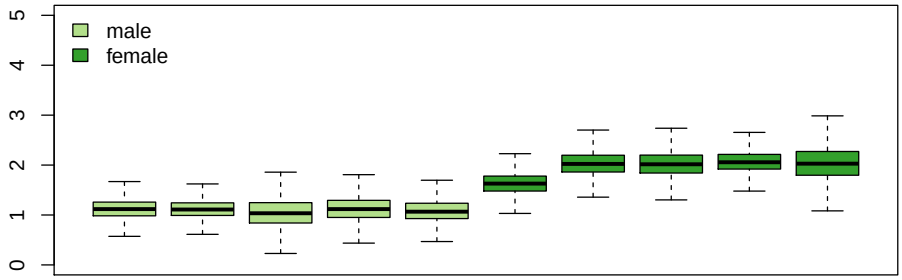

Figure 2: Copy number estimates for polymorphic markers on chromosome X. crlmm assumes that the median copy number across samples at a given marker on X is 1 for men and 2 for women.

2 The CNSet container

The objects returned by the `genotype` and `crlmmCopynumber` have assay data elements that are pointers to `ff` objects stored in the directory `outdir`. Had we not loaded the ff prior to running these functions, the `AssayData` elements would be ordinary matrices, though the RAM required for running the algorithm would be substantial. The functions `open` and `close` open and close the connections to the `assayData` elements. Subsetting an `ff` object pulls the data from disk into memory and should be used with caution. In particular, subsetting the `gtSet` would subset each element in the `assayData` slot, returning an object of the same class but with `assayData` elements that are matrices. Such an operation can be exceedingly slow when performed over a network and require subsantial RAM. The preferred approach is to extract only the assay data element that is needed. In the example below, we extract the genotype calls for the the first 50 samples.

```
> dims(cnSet)

          alleleA alleleB    call callProbability
Features 1852215 1852215 1852215         1852215
Samples      180     180     180             180

> print(object.size(cnSet), units = "Mb")

134.3 Mb

> invisible(open(calls(cnSet)))
> gt <- calls(cnSet)[, 1:50]
```

The `CNSet` class also contains the slot `batchStatistics` that contains batch-specific summaries needed for copy number estimation. In particular, each element is a matrix (or an ff object) with R rows and C columns, correspoinding to R markers and C batches. The summaries includes the within genotype cluster medians and median absolute deviations (mads), but also parameters estimated from the linear model. (For unobserved genotypes, the medians are imputed and the variance is obtained the median variance (across markers) within a batch.) The elements of the slot can be listed as follows.

```
> assayDataElementNames(batchStatistics(cnSet))
```

```
 [1] "corrAA"     "corrAB"     "corrBB"     "flags"      "madA.AA"
 [6] "madA.AB"    "madA.BB"    "madB.AA"    "madB.AB"    "madB.BB"
[11] "medianA.AA" "medianA.AB" "medianA.BB" "medianB.AA" "medianB.AB"
[16] "medianB.BB" "N.AA"       "N.AB"       "N.BB"       "nuA"
[21] "nuB"        "phiA"       "phiB"       "phiPrimeA"  "phiPrimeB"
[26] "tau2A.AA"   "tau2A.BB"   "tau2B.AA"   "tau2B.BB"
```

Note that for the Affymetrix 6.0 platform the assay data elements each have a row dimension corresponding to the total number of polymorphic and nonpolymorphic markers interrogated by the Affymetrix 6.0 platform. A consequence of keeping the rows of the assay data elements the same for all of the statistical summaries is that the matrix used to store genotype calls is larger than necessary. Also, note the additional overhead of some operations when using `ff` objects. For instance, the posterior probabilities for the CRLMM genotype calls are represented as integers. The accessor `snpCallProbability` can be used to access these confidence scores. When stored as matrices, converting the integer representation back to the probability scale is straightforward as shown below. However, for the `ff` objects we must first convert the ff object to a matrix. One could use the function `[,]` but this could be slow and require a lot of RAM depending on the size of the dataset. We suggest pulling only the needed rows and columns from memory. In the following example, we convert the integer scores to probabilities for the CEPH samples. As genotype confidence scores are not applicable to the nonpolymorphic markers, we extract only the polymorphic markers using the `isSnp` function.

```
> rows <- which(isSnp(cnSet))
> cols <- which(batch == "C")
> invisible(open(snpCallProbability(cnSet)))
> posterior.prob <- tryCatch(i2p(snpCallProbability(cnSet)),
     error = function(e) print("This will not work for an ff object."))
```

```
[1] "This will not work for an ff object."
```

Accessors for the quantile normalized intensities for the A allele at polymorphic loci:

```
> snp.index <- which(isSnp(cnSet))
> np.index <- which(!isSnp(cnSet))
> invisible(open(A(cnSet)))
> a <- (A(cnSet))[snp.index, ]
> dim(a)
```

```
[1] 906600    180
```

The extra set of parentheses surrounding `A(cnSet2)` above is added to emphasize the appropriate order of operations. Subsetting the entire `cnSet` object in the following, unevaluated codechunk should be avoided for large datasets.

```
> a <- A(cnSet[snp.index, ])
```

The quantile-normalized intensities for nonpolymorphic loci are obtained by:

```
> npIntensities <- (A(cnSet))[np.index, ]
> invisible(close(A(cnSet)))
```

Quantile normalized intensities for the B allele at polymorphic loci:

```
> invisible(open(B(cnSet)))
> b.snps <- (B(cnSet))[snp.index, ]
```

Note that NA's are stored in the slot for normalized 'B' allele intensities:

```
> all(is.na(B(cnSet)[np.index, ]))

[1] TRUE

            used   (Mb) gc trigger   (Mb)  max used   (Mb)
Ncells   3292241  175.9    4953636  264.6   4953636  264.6
Vcells 186546102 1423.3  530749310 4049.3 618255332 4717.0
```

2.1 Other accessors

Information on physical position and chromosome can be accessed as follows:

```
> xx <- position(cnSet)
> yy <- chromosome(cnSet)
```

Parameters from the linear model used to estimate copy number are stored in the slot `batchStatistics`. TODO: Describe accessors for batch-level summaries.

3 Suggested visualizations

SNR. A histogram of the signal to noise ratio for the HapMap samples:

```
> open(cnSet$SNR)

[1] FALSE

> hist(cnSet$SNR[, ], xlab = "SNR", main = "", breaks = 25)
```

One sample at a time: locus-level estimates Figure 4 plots physical position (horizontal axis) versus copy number (vertical axis) for the first sample. There is less information to estimate copy number at nonpolymorphic loci; improvements to the univariate prediction regions at nonpolymorphic loci are a future area of research. If the SNPchip is available, an idiogram can be added to the existing plotting coordinates as indicated in the following example.

```
> marker.index <- which(chromosome(cnSet) == 1)
> cn <- totalCopynumber(cnSet, i = marker.index, j = 1)
> x <- position(cnSet)[marker.index]
> par(las = 1, mar = c(4, 5, 4, 2))
> plot(x, cn, pch = ".", cex = 2, xaxt = "n", col = "grey60",
     ylim = c(0, 6), ylab = "copy number", xlab = "physical position (Mb)",
     main = paste(sampleNames(cnSet)[1], ", CHR: 1"))
> axis(1, at = pretty(x), labels = pretty(x)/1e+06)
> require(SNPchip)
> invisible(plotCytoband(1, new = FALSE, cytoband.ycoords = c(5.5,
     6), label.cytoband = FALSE))
```

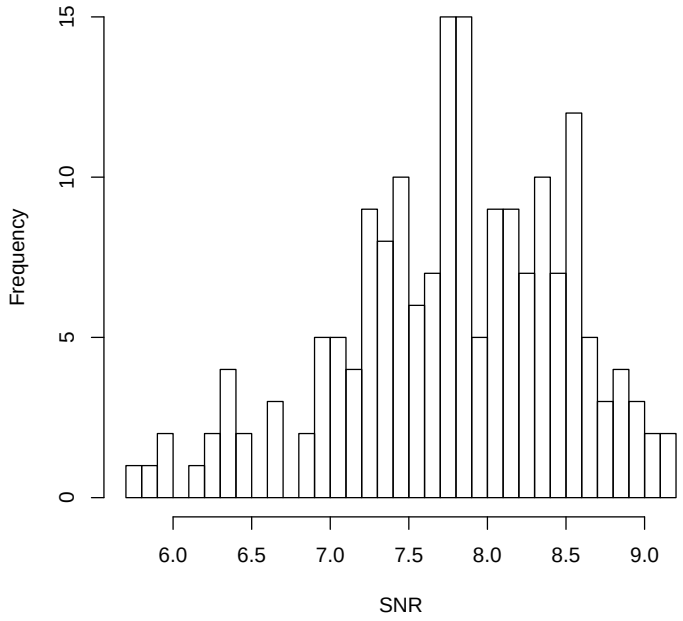

Figure 3: Signal to noise ratios for the HapMap samples. SNRs below 5 for the Affymetrix platform are often samples of lower quality. Such samples will tend to have much more variable estimates of copy number.

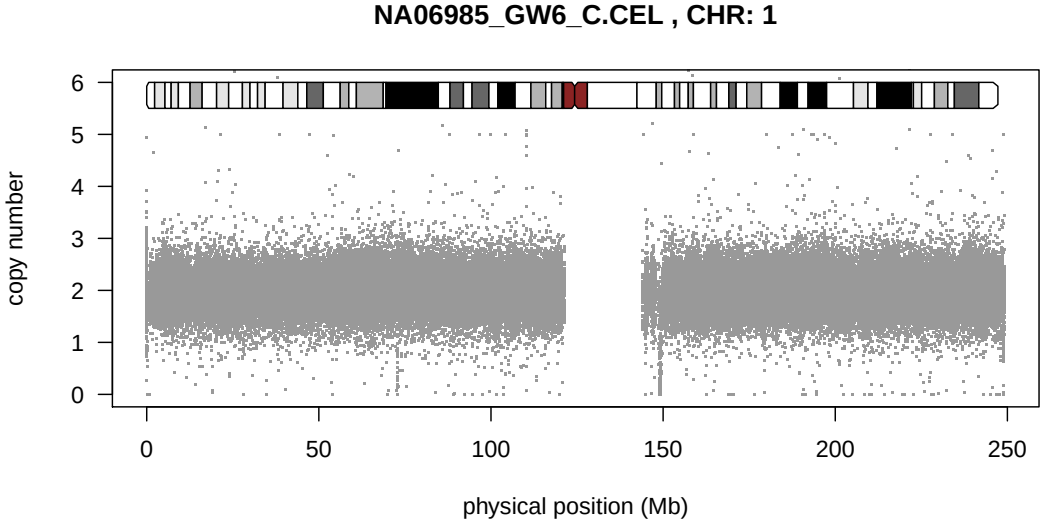

Figure 4: Total copy number (y-axis) for chromosome 1 plotted against physical position (x-axis) for one sample. Estimates at nonpolymorphic loci are plotted in light blue.

One SNP at a time Scatterplots of the A and B allele intensities (log-scale) can be useful for assessing the biallelic genotype calls. This section of the vignette is currently under development.

4 Reasons for missing values

There are several reasons for estimates of the allele-specific copy number to have missing values (`NA`'s). This section briefly elaborates on the source of missing values in the HapMap analysis and discusses possible alternatives to reduce the number of missing values. Note that allele-specific copy number, 'CA' and 'CB', is not saved in the `cnSet` object. Rather, the respective accessors calculate 'CA' and 'CB' on the fly from the normalized intensities and from the marker-specific parameter estimates in the linear model. In general, a missing value arises when the background or slope parameter was not estimated in the linear model. Most often, missing values occur when the genotype confidence scores for a SNP were below the threshold used by the `crlmmCopynumber` function. For the HapMap analysis, we used a confidence threshold of 0.9.

```
> autosome.index <- which(isSnp(cnSet) & chromosome(cnSet) <
      23)
> sample.index <- which(batch(cnSet) == "C")
> ca <- CA(cnSet, i = autosome.index, j = sample.index)
> missing.ca <- is.na(ca)
> (nmissing.ca <- sum(missing.ca))

[1] 0
```

If `nmissing.ca` is nonzero, then one could check the genotype confidence scores provided by the crlmm genotyping algorithm against the threshold specified in `crlmmCopynumber`.

```
> if (nmissing.ca > 0) {
      invisible(open(snpCallProbability(cnSet)))
      gt.conf <- i2p(snpCallProbability(cnSet)[autosome.index,
          sample.index])
      invisible(close(snpCallProbability(cnSet)))
      below.thr <- gt.conf < GT.CONF.THR
      index.allbelow <- as.integer(which(rowSums(below.thr) ==
          length(sample.index)))
      all.equal(index, index.allbelow)
      sum(index.allbelow %in% index)/length(index)
 }
```

We repeat the above check for missing values at polymorphic loci on chromosome X. In this case, we compare the `rowSums` of the missing values to the number of samples to check whether all of the estimates are missing for a given SNP.

```
> X.index <- which(isSnp(cnSet) & chromosome(cnSet) ==
      23)
> ca.X <- CA(cnSet, i = X.index, j = sample.index)
> missing.caX <- is.na(ca.X)
> (nmissing.caX <- sum(missing.caX))

[1] 20970

> missing.snp.index <- which(rowSums(missing.caX) == length(sample.index))
> index <- which(rowSums(missing.caX) == length(sample.index))
> length(index) * length(sample.index)/nmissing.caX

[1] 1
```

From the above codechunk, we see that 233 SNPs have NAs for all the samples. Next, we tally the number of NAs for polymorphic markers on chromosome X that are below the confidence threshold. For the HapMap analysis, all of the missing values arose from SNPs in which either the men or the women had confidence scores that were all below the threshold.

```
> if (nmissing.caX > 0) {
     invisible(open(snpCallProbability(cnSet)))
     gt.conf <- i2p(snpCallProbability(cnSet)[X.index,
         sample.index])
     invisible(close(snpCallProbability(cnSet)))
     below.thr <- gt.conf < GT.CONF.THR
     F <- which(cnSet$gender[sample.index] == 2)
     M <- which(cnSet$gender[sample.index] == 1)
     index.allbelowF <- as.integer(which(rowSums(below.thr[,
         F]) == length(F)))
     index.allbelowM <- as.integer(which(rowSums(below.thr[,
         M]) == length(M)))
     index.allbelow <- as.integer(unique(c(index.allbelowF,
         index.allbelowM)))
     all.equal(index, index.allbelow)
     sum(index.allbelow %in% index)/length(index)
 }
```

```
[1] 0.9957082
```

Next, we verify that the number of missing values is the same for the 'B' allele at autosomal polymorphic loci

```
> cb <- CB(cnSet, i = autosome.index, j = sample.index)
> missing.cb <- is.na(cb)
> sum(missing.cb)
```

```
[1] 0
```

For nonpolymorphic loci, the genotype confidence scores are irrelevant and estimates are available at most markers.

```
> np.index <- which(!isSnp(cnSet) & chromosome(cnSet) ==
     23)
> ca.F <- CA(cnSet, i = np.index, j = F)
> ca.M <- CA(cnSet, i = np.index, j = M)
> ca.F <- ca.F[-match("CN_974939", rownames(ca.F)), ]
> ca.M <- ca.M[-match("CN_974939", rownames(ca.M)), ]
> sum(is.na(ca.F))
```

```
[1] 0
```

```
> sum(is.na(ca.M))
```

```
[1] 0
```

TODO: marker CN_974939 has NAs for the normalized intensities. This is because CN_974939 is not in the `npProbesFid` file in genomewidesnp6Crlmm. The `npProbesFid` file should be updated in the next genomewidesnp6Crlmm release.

In total, there were 233 polymorphic markers on chromosome X for which copy number estimates are not available. Lowering the confidence threshold would permit estimation of copy number at most of these loci.

A confidence threshold is included as a parameter for the copy number estimation as an approach to reduce the sensitivity of genotype-specific summary statistics, such as the within-genotype median, to intensities from samples that do not clearly fall into one of the biallelic genotype clusters. There are drawbacks to this approach, including variance estimates that can be a bit optimistic at some loci. More direct approaches for outlier detection and removal may be explored in the future.

Copy number estimates for other chromosomes, such as mitochondrial and chromosome Y, are not currently available in crlmm.

5 Session information

```
> toLatex(sessionInfo())
```

- R version 2.13.0 Under development (unstable) (2010-11-14 r53587), x86_64-unknown-linux-gnu
- Locale: LC_CTYPE=en_US.iso885915, LC_NUMERIC=C, LC_TIME=en_US.iso885915, LC_COLLATE=en_US.iso885915, LC_MONETARY=C, LC_MESSAGES=en_US.iso885915, LC_PAPER=en_US.iso885915, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.iso885915, LC_IDENTIFICATION=C
- Base packages: base, datasets, graphics, grDevices, methods, stats, tools, utils
- Other packages: Biobase 2.11.6, bit 1.1-4, crlmm 1.9.6, ff 2.1-4, oligoClasses 1.13.5, RColorBrewer 1.0-2, SNPchip 1.13.0
- Loaded via a namespace (and not attached): affyio 1.19.2, annotate 1.29.2, AnnotationDbi 1.13.2, Biostrings 2.19.0, DBI 0.2-5, ellipse 0.3-5, genefilter 1.33.0, IRanges 1.7.37, mvtnorm 0.9-92, preprocessCore 1.13.1, RSQLite 0.9-3, splines 2.13.0, survival 2.36-1, xtable 1.5-6

References

References

[1] Robert B Scharpf, Ingo Ruczinski, Benilton Carvalho, Betty Doan, Aravinda Chakravarti, and Rafael Irizarry. A multilevel model to address batch effects in copy number estimation using snp arrays. *Biostatistics*, 2010.