Infrastructure for copy number analysis in crlmm

Rob Scharpf

May 30, 2012

Abstract

This vignette provides an overview of the *CNSet* class and a brief discussion of the underlying infrastructure for large data support with the ff package. This package instantiates an object of class *CNSet* using a trivial dataset with 3 files. As this sample size is too small for estimating copy number with the crlmm package, the final section of this vignette loads an object created by the analysis of 180 HapMap CEL files (Affymetrix 6.0 platform). In particular, this object was instantiated by running (1) the `AffymetrixPreprocessCN` vignette and (2) the `copynumber` vignette.

```
> library(ff)
> library(crlmm)
```

1 Supported platforms

The supported Affymetrix and Infinium platforms are those for which a corresponding annotation package is available. The annotation packages contain information on the markers, such as physical position and chromosome, as well as pre-computed parameters estimated from HapMap used during the preprocessing and genotyping steps. For Affymetrix, the 5.0 and 6.0 platforms are supported and the corresponding annotation packages are genomewidesnp5Crlmm and genomewidesnp6Crlmm. Supported Infinium platforms are listed in the following code chunk.

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

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

Large data: In order to reduce crlmm's memory footprint for copy number estimation, we require the ff. The ff package provides infrastructure for accessing and writing data to disk instead of keeping data in memory. As the functions for preprocessing, genotyping, and copy number estimation do not simultaneously require all samples and all probes in memory, memory-usage by crlmm can be fine-tuned by reading in and processing subsets of the markers and/or samples. The functions `ocSamples` and `ocProbesets` in the oligoClasses package can be used to declare how many markers and samples to read at once. In general, specifying smaller values should reduce the RAM required for a particular job. In general, smaller values will increase the run-time. In the following code-chunk, we declare that crlmm should process 150,000 markers at a time (when possible) and 500 samples at a time. If our dataset contained fewer than 500 samples, the `ocSamples` option would not have any effect. One can view the current settings for these commands, by typing the functions without an argument.

```
> ocProbesets(50e3)
> ocSamples(200)
```

2 The *CNSet* container

2.1 Instantiating an object of class *CNSet*

An object of class *CNSet* can be instantiated by one of two methods:

Approach 1: during the preprocessing of the raw intensities for Illumina and Affymetrix arrays by the the functions `constructInf` and `genotype`, respectively. (The `genotype` calls the non-exported function `constructAffy` to initialize a *CNSet* object for Affymetrix platforms.)

Approach 2: by subsetting an existing `CNSet` object. As per usual, the '[' method can be used to extract a subset of markers i as in '[i,]', a subset of samples j as in '[, j]', or a subset of markers i and samples j as in '[i, j]'.

There are important differences in the underlying data representation depending on how the object was instantiated. In particular, objects generated by the functions `constructInf` and `genotype` store high-dimensional data on disk rather than in memory through protocols defined in the R package ff. For instance, the normalized intensities and genotype calls in a *CNSet*-instance from approach (1) are ff-derived objects. By contrast, when such an objected generated by approach (1) is subset by the '[' method, an object of the same class is returned but the ff-derived objects are coerced to ordinary matrices. Note, therefore, that both approaches (1) and (2) may involve substantial I/O.

2.1.1 Approach 1

To illustrate the first approach, we begin by specifying a local directory to store output files and setting the `ldPath` function with this path.

```
> outdir <- paste("/local_data/r00/crlmm/", getRversion(), "/infrastructure", sep="")
> ldPath(outdir)
> if(!file.exists(outdir)) dir.create(outdir)
```

Next we load the annotation package required, as well as the R package hapmapsnp6 that contains 3 example CEL files. We use the `system.file` function to find the path to the CEL files.

```
> require(genomewidesnp6Crlmm) & require(hapmapsnp6)

[1] TRUE

> path <- system.file("celFiles", package="hapmapsnp6")
> celfiles <- list.celfiles(path, full.names=TRUE)
```

Typically, an object of class *CNSet* is instantiated as part of the preprocessing and genotyping by calling the function `genotype`, as illustrated in the `AffymtrixPreprocessCN` vignette.

```
> exampleSet <- genotype(celfiles, batch=rep("1", 3), cdfName="genomewidesnp6")
```

Several files with `.ff` extensions now appear in the directory indicated by the `ldPath` function.

```
> ldPath()

[1] "/local_data/r00/crlmm/2.15.0/infrastructure"
```

One could also instantiate an object of class *CNSet* without preprocessing/genotyping by calling the non-exported function `constructAffy` directly using the `:::` operator.

```
> tmp <- crlmm:::constructAffy(celfiles, batch=rep("1", 3), cdfName="genomewidesnp6")
```

The `show` method provides a concise summary of the `exampleSet` object. Note the class of the elements in the `batchStatistics` and `assayData` slots is indicated in the first line of the summary.

```
> invisible(open(exampleSet))
> exampleSet

CNSet (assayData/batchStatistics elements: ff_matrix)
CNSet (storageMode: lockedEnvironment)
assayData: 1852426 features, 3 samples
  element names: alleleA, alleleB, call, callProbability
protocolData
  rowNames: NA06985_GW6_C.CEL NA06991_GW6_C.CEL
    NA06993_GW6_C.CEL
  varLabels: ScanDate
  varMetadata: labelDescription
phenoData
  sampleNames: NA06985_GW6_C.CEL NA06991_GW6_C.CEL
    NA06993_GW6_C.CEL
  varLabels: SKW SNR gender
  varMetadata: labelDescription
featureData
  featureNames: SNP_A-2131660 SNP_A-1967418 ... CN_929945
    (1852426 total)
  fvarLabels: isSnp position chromosome
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation: genomewidesnp6
genome:  hg19
batch:    1:3
batchStatistics:  29  elements,  1852426  features,  1  batches
```

As the `assayData` elements of the `exampleSet` object are stored on disk rather than in memory, we inspect attributes of the elements by first opening the file connection using the `open`. For example, in the following code we extract the normalized intensities for the A allele by first opening the object returned by the `A` function. The name of the file where the data is stored on disk is provided by the `filename`. Finally, the normalized intensities can be pulled from disk to memory by the '[' method. It can be useful to wrap the '[' method by the `as.matrix` to ensure that the output is the desired class.

```
> invisible(open(exampleSet))
> class(A(exampleSet))

[1] "ff_matrix" "ff_array"  "ff"

> filename(A(exampleSet))

[1] "/local_data/r00/crlmm/2.15.0/infrastructure/crlmmA-22416285f0b8.ff"

> as.matrix(A(exampleSet)[1:5, ])

              NA06985_GW6_C.CEL NA06991_GW6_C.CEL NA06993_GW6_C.CEL
SNP_A-2131660              1381              1226               720
SNP_A-1967418               289               223               251
SNP_A-1969580               917              1207              1124
SNP_A-4263484              1502              2424              2319
SNP_A-1978185              1130              1243              1377
```

Moving `*.ff` files Ideally, the files with `.ff` extension should not be moved. However, if this is not possible, the safest way to move these files is to clone all of the ff objects using the `clone`, followed by the `delete` function to remove the original files on disk. An example of the `delete` function is included at the end of the `IlluminaPreprocessCN` vignette. See the documentation for the `clone` and `delete` functions in the ff package for additional details.

Order of operations: For *CNSet*-instances derived by approach (1), users should be mindful of the substantial I/O when using accessors to extract data from the class. For example, the following 2 methods would extract identical results, with the latter being much more efficient (extra parentheses are added to the second operation to emphasize the order of operations):

```
> ##inefficient
> ##invisible(open(cnSet))
> A(exampleSet[1:5, ])

             NA06985_GW6_C.CEL NA06991_GW6_C.CEL NA06993_GW6_C.CEL
SNP_A-2131660              1381              1226               720
SNP_A-1967418               289               223               251
SNP_A-1969580               917              1207              1124
SNP_A-4263484              1502              2424              2319
SNP_A-1978185              1130              1243              1377

> ## preferred
> (A(exampleSet))[1:5, ]

             NA06985_GW6_C.CEL NA06991_GW6_C.CEL NA06993_GW6_C.CEL
SNP_A-2131660              1381              1226               720
SNP_A-1967418               289               223               251
SNP_A-1969580               917              1207              1124
SNP_A-4263484              1502              2424              2319
SNP_A-1978185              1130              1243              1377
```

2.1.2 Approach 2: using '['

Here we instantiate a new object of class *CNSet* by applying the '[' method to an existing object of class *CNSet*.

```
> cnset.subset <- exampleSet[1:5, ]
```

Note the class of the `batchStatistics` and `assayData` elements of the `cnset.subset` object printed in the first line of the summary.

```
> show(cnset.subset)

CNSet (assayData/batchStatistics elements: matrix)
CNSet (storageMode: lockedEnvironment)
assayData: 5 features, 3 samples
  element names: alleleA, alleleB, call, callProbability
protocolData
  rowNames: NA06985_GW6_C.CEL NA06991_GW6_C.CEL
    NA06993_GW6_C.CEL
  varLabels: ScanDate
  varMetadata: labelDescription
phenoData
  sampleNames: NA06985_GW6_C.CEL NA06991_GW6_C.CEL
    NA06993_GW6_C.CEL
  varLabels: SKW SNR gender
  varMetadata: labelDescription
featureData
  featureNames: SNP_A-2131660 SNP_A-1967418 ... SNP_A-1978185
    (5 total)
  fvarLabels: isSnp position chromosome
```

```
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation: genomewidesnp6
genome:  hg19
batch:    1:3
batchStatistics:  29  elements,  5  features,  1  batches
```

2.2 Slots of class *CNSet*

2.2.1 `featureData`

Information on physical position, chromosome, and whether the marker is a SNP can be accessed through accessors defined for the `featureData`.

```
> library(Biobase)
> fvarLabels(exampleSet)

[1] "isSnp"      "position"   "chromosome"

> position(exampleSet)[1:10]

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

> chromosome(exampleSet)[1:10]

 [1] 1 1 1 1 1 1 1 1 1 1

> is.snp <- isSnp(exampleSet)
> table(is.snp)

is.snp
 FALSE   TRUE
945615 906811

> snp.index <- which(is.snp)
> np.index <- which(!is.snp)
> chr1.index <- which(chromosome(exampleSet) == 1)
```

2.2.2 `assayData`

The `assayData` elements are of the class *ff_matrix/ffdf* or *matrix*, depending on how the *CNSet* object was instantiated. Elements in the `assayData` environment can be listed using the `ls` function.

```
> ls(assayData(exampleSet))

[1] "alleleA"         "alleleB"         "call"
[4] "callProbability"
```

The normalized intensities for the A and B alleles have names `alleleA` and `alleleB` and can be accessed by the methods `A` and `B`, respectively. Genotype calls and confidence scores can be accessed by `snpCall` and `snpCallProbability`, respectively. Note that the confidence scores are represented as integers to reduce the filesize, but can be tranlated to the probability scale using the R function `i2p`. For example,

```
> scores <- as.matrix(snpCallProbability(exampleSet)[1:5, 1:2])
> i2p(scores)
```

```
              NA06985_GW6_C.CEL NA06991_GW6_C.CEL
SNP_A-2131660         0.9999964         0.9999996
SNP_A-1967418         0.9999969         0.9999997
SNP_A-1969580         0.9995187         0.9995139
SNP_A-4263484         0.9999999         1.0000000
SNP_A-1978185         1.0000000         1.0000000
```

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.

2.2.3 `batch` and `batchStatistics`

As defined in Leek *et al.* 2010, *Batch effects are sub-groups of measurements that have qualitatively different behaviour across conditions and are unrelated to the biological or scientific variables in a study.* The `batchStatistics` slot is an environment used to store SNP- and batch-specific summaries, such as the sufficient statistics for the genotype clusters and the linear model parameters used for copy number estimation. The `batch` slot is used to store the 'batch name' for each array. For small studies in which the samples were processed at similar times (e.g., within a month), all the samples can be considered to be in the same batch. For large studies in which the samples were processed over several months, users should the scan date of the array or the chemistry plate are useful surrogates. The only constraint on the `batch` variable is that it must be a character vector that is the same length as the number of samples to be processed. The `batch` is specified as an argument to the R functions `constructInf` and `genotype` that instantiate *CNSet* objects for the Illumina and Affymetrix platforms, respectively. The `batch` function can be used to access the `batch` information on the samples as in the following example.

```
> batch(exampleSet)
```

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

For the `batchStatistics` slot, the elements in the environment have the class *ff_matrix/ffdf* or *matrix*, depending on how the *CNSet* object was instantiated. The dimension of each element is the number of markers (SNPs + nonpolymorphic markers) $\times$ the number of batches. The names of the elements in the environment can be list using the R function `ls`.

```
> ls(batchStatistics(exampleSet))
```

```
 [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"
```

Currently, the batch-specific summaries are stored to allow some flexibility in the choice of downstream analyses of copy number and visual assessments of model fit. Documentation for such applications will be expanded in future versions of crlmm, and are currently not intended to be accessed directly by the user.

2.2.4 `phenoData`

Sample-level summaries obtained during the preprocessing/genotyping steps include skew (SKW), the signal to noise ratio (SNR), and gender (1=male, 2=female) are stored in the `phenoData` slot. As for other *eSet* extensions, the `$` method can be used to extract these summaries. For *CNSet* objects generated by approach (1), these elements are of the class *ff_vector*.

```
> varLabels(exampleSet)
```

```
[1] "SKW"    "SNR"    "gender"
```

```
> class(exampleSet$gender)
```

```
[1] "ff_vector" "ff"
```

```
> invisible(open(exampleSet$gender))
> exampleSet$gender
```

```
ff (open) integer length=3 (3)
[1] [2] [3]
  2   2   1
```

The '[' methods without arguments can be used to coerce to a vector.

```
> c("male", "female")[exampleSet$gender[]]
```

```
[1] "female" "female" "male"
```

```
> invisible(close(exampleSet$gender))
```

2.2.5 protocolData

The scan date of the arrays are stored in the `protocolData`.

```
> varLabels(protocolData(exampleSet))
```

```
[1] "ScanDate"
```

```
> protocolData(exampleSet)$ScanDate
```

```
[1] 2007-03-06 2007-03-06 2007-03-06
Levels: 2007-03-06
```

3 Trouble shooting with a HapMap example

This section uses an object of class *CNSet* instantiated by the `AffymetrixPreprocessCN` vignette and saved to a local path on our computing cluster indicated by the object `outdir` below. The `copynumber` vignette was used to fill out the `batchStatistics` slot of the `cnSet` object.

```
> if(getRversion() < "2.13.0"){
         rpath <- getRversion()
 } else rpath <- "trunk"
> outdir <- paste("/thumper/ctsa/snpmicroarray/rs/ProcessedData/crlmm/", rpath, "/copynumber_vignette",
```

Next, we load the `cnSet` object.

```
> if(!exists("cnSet")) load(file.path(outdir, "cnSet.rda"))
> invisible(open(cnSet))
```

3.1 Missing values

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 0.80 (the default). In the following code, we assess NA's appearing for the raw copy number estimates for the first 10 samples.

```
> GT.CONF.THR <- 0.80
> autosome.index <- which(isSnp(cnSet) & chromosome(cnSet) < 23)
> sample.index <- 1:10
> ct <- totalCopynumber(cnSet, i=autosome.index, j=sample.index)
> ca <- CA(cnSet, i=autosome.index, j=sample.index)
> cb <- CB(cnSet, i=autosome.index, j=sample.index)
> missing.ca <- is.na(ca)
> missing.cb <- is.na(cb)
> (nmissing.ca <- sum(missing.ca))

[1] 0

> (nmissing.cb <- sum(missing.cb))

[1] 0

> identical(nmissing.ca, nmissing.cb)

[1] TRUE
```

If `nmissing.ca` is nonzero, 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)))
        nmissingBecauseOfGtThr <- length(index.allbelow) * length(sample.index)
        stopifnot(identical(nmissingBecauseOfGtThr, nmissing.ca))
        ## or calculate the proportion of missing effected by low crlmm confidence
        length(index.allbelow) * length(sample.index)/nmissing.ca
 }
```

One could inspect the cluster plots for the 'low confidence' calls.

```
> ## TODO
```

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.

```
> ## start with first batch
> sample.index <- which(batch(cnSet)==batch(cnSet)[1])
> 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] 15781

> 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
```

From the above codechunk, we see that 367 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.

```
> invisible(open(cnSet$gender))
> F <- which(cnSet$gender[sample.index] == 2)
> M <- which(cnSet$gender[sample.index] == 1)
> gt.conf <- i2p(snpCallProbability(cnSet)[X.index, sample.index])
> below.thr <- gt.conf < GT.CONF.THR
> index.allbelowF <- as.integer(which(rowSums(below.thr[, F]) == length(F)))
> index.allbelowM <- as.integer(which(rowSums(below.thr[, M]) == length(M)))
> all.equal(index.allbelowF, index.allbelowM)

[1] TRUE

> all.equal(as.integer(index.allbelowF), as.integer(index))

[1] TRUE
```

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)
> ## NAs for one marker
> 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] 23

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

[1] 20
```

In total, there were 367 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.

```
> invisible(close(cnSet))
> invisible(close(cnSet$gender))
```

4 Session information

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

- R version 2.15.0 Patched (2012-04-25 r59178), 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=en_US.iso885915, LC_MESSAGES=en_US.iso885915, LC_PAPER=C, 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 2.16.0, BiocGenerics 0.3.0, BiocInstaller 1.5.7, bit 1.1-8, cacheSweave 0.6-1, crlmm 1.15.4, ff 2.2-7, filehash 2.2-1, genomewidesnp6Crlmm 1.0.6, hapmapsnp6 1.3.7, lattice 0.20-6, oligoClasses 1.19.15, stashR 0.3-5, VanillaICE 1.19.11
- Loaded via a namespace (and not attached): affyio 1.24.0, annotate 1.34.0, AnnotationDbi 1.18.0, Biostrings 2.24.1, codetools 0.2-8, compiler 2.15.0, DBI 0.2-5, digest 0.5.2, ellipse 0.3-7, foreach 1.4.0, genefilter 1.38.0, GenomicRanges 1.9.13, grid 2.15.0, IRanges 1.15.9, iterators 1.0.6, msm 1.1.1, mvtnorm 0.9-9992, preprocessCore 1.18.0, RSQLite 0.11.1, splines 2.15.0, stats4 2.15.0, survival 2.36-14, xtable 1.7-0, zlibbioc 1.2.0