Integration with the crlmm package for copy number inference

Robert Scharpf

May 21, 2014

```
> library(oligoClasses)
> library(VanillaICE)
> library(crlmm)
> library(SNPchip)
> library(IRanges)
> library(foreach)
```

We load a portion of chromosome 8 from 2 HapMap samples that were processed using the *crlmm* package.

```
> data(cnSetExample, package="crlmm")
```

In the following unevaluated code chunk, we could enable parallelization of the hidden Markov models using the package *snow*.

```
> ##registerDoSEQ()
> library(snow)
> library(doSNOW)
> cl <- makeCluster(2, type="SOCK")
> registerDoSNOW(cl)
> ocSamples(2)
```

The data `cnSetExample` is an object of class `CNSet`. We coerce the `CNSet` object to a list class that contains information on copy number (log R ratios), genotypes, genotype probabilities, and B allele frequencies. If the `CNSet` object contained assay data for multiple chromosomes, each element in the following list container would contain the assay data for one chromosome.

```
> oligoList <- BafLrrSetList(cnSetExample)
> oligoList[[1]]
```

```
BafLrrSet (storageMode: environment)
assayData: 15000 features, 2 samples
  element names: baf, lrr
protocolData: none
phenoData
  sampleNames: NA19007 NA19003
  varLabels: SKW SNR gender celFiles
  varMetadata: labelDescription
featureData
  featureNames: CN_1252892 SNP_A-8325516 ... SNP_A-2094900
    (15000 total)
  fvarLabels: isSnp position chromosome
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation: genomewidesnp6
genome:  hg19
```

Wave correction

To correct for genomic waves that correlate with GC content [refs], we use the R package *ArrayTV*. Currently, the *ArrayTV* is available from github (`https://github.com/rscharpf/ArrayTV`). As this package is not yet available from Bioconductor, the code chunks in this section are not evaluated.

In the following code-chunk, we first select a subset of the samples in the study to evaluate the genomic window for wave correction. See the ArrayTV vignette for details. For large datasets, one could randomly select 20 or 25 samples to compute the window, and then use a pre-selected window for wave correction on the remaining samples. We illustrate this process, even though our dataset only contains two samples. We process the dataset 20 samples at a time by setting `ocSamples()` to 20, thereby controlling the amount of RAM required (use smaller batch sizes for systems with less RAM).

```
> library(ArrayTV)
> i <- seq_len(ncol(oligoList))
> ocSamples(20)
> increms <- c(10,1000,100e3)
> wins <- c(100,10e3,1e6)
> tvScores <- gcCorrect(oligoList[, 1],
+                       increms=increms,
+                       maxwins=wins,
+                       returnOnlyTV=TRUE,
+                       verbose=TRUE)
> order(tvScores[[1]][,2], decreasing=TRUE)
```

Next, we select a small window of 10 bp and a larger window of 10,000 bp and pre-compute the gc composition:

```
> gc.matrix <- computeGC(oligoList, c(10, 10e3), c(10,10e3))
```

We use the gc content for the two windows in `gc.matrix` to correct the log R ratios for all samples in the dataset. If the assay data elements in the `oligoList` object were `ff` objects, NULL is returned and the log R ratios are updated on disk. If the assay data elements are matrices, a `BafLrrSetList` object is returned with the correct log R ratios.

```
> oligoList2 <- gcCorrect(oligoList, increms=c(10,10e3), maxwins=c(10,10e3),
+                         providedGC=gc.matrix)
```

HMM

To identify CNVs, we fit a 6-state hidden markov model from estimates of the B allele frequency and log R ratios. A `hmm` method is defined for the `BafLrrSetList` class, and we apply the method directly with a few parameters that change the arguments from their default values. For example, the `TAUP` parameter scales the transition probability matrix. Larger values of `TAUP` makes it more expensive to transition from the normal copy number state to states with altered copy number.

```
> res <- hmm(oligoList, p.hom=0.1, nupdates=5, TAUP=1e10)
```

Commonly used accessors for the BafLrrSetList class

The `[[` method can be used to extract a `BafLrrSet` object for one chromosome. This object is similar to an `ExpressionSet` in the Biobase package, but with assay data elements for the log R ratios and B allele frequencies.

```
> oligoSet <- oligoList[[1]]
```

Accessors for the log R ratios and B allele frequencies are given by `lrr` and `baf`, respectively.

```
> r <- lrr(oligoSet)
> b <- baf(oligoSet)
```

Note that the above objects are stored as integers.

```
> range(r, na.rm=TRUE)
```

```
[1] -278  272
```

```
> range(b, na.rm=TRUE)
```

```
[1]    0 1000
```

The log R ratios and B allele frequencies can be transformed back to the original scale by dividing by 100 and 1000, respectively.

```
> r <- r/100
> b <- b/1000
```

Visualization of genomic intervals (e.g., CNVs) with log R ratios and B allele frequencies

In this section, we use lattice-style plots to visualize the genomic intevals of the altered copy number states. We begin by subsetting the `GRanges` object from the HMM to contain only the altered copy number states.

```
> gr <- unlist(res)
> gr <- gr[state(gr) %in% c(1,2,5,6) & sampleNames(gr) == "NA19007",]
```

Next, we create a `SummarizedExperiment` object containing the log R ratios, B allele frequencies, and physical position (in Mb) that are within `maxgap` basepairs of the CNV intervals stored in the `gr` object. Because creation of a `SummarizedExperiment` will read in all of the relevant low level data from disk, it is useful to first select only the relevant chromosome and samples. While these steps are not really necessary in our toy example with only one chromosome and 2 samples, it can increase computational speed substantially in large studies. Finally, we create coerce the `brList` object to a `SummarizedExperiment`.

```
> chr <- paste("chr", chromosome(oligoList), sep="")
> brList <- oligoList[chr %in% chromosome(gr)]
> brList <- brList[, match(sampleNames(gr)[1], sampleNames(brList))]
> se <- as(brList, "SummarizedExperiment")
```

Next, we use the lattice function `xyplot` to plot the log R ratios and B allele frequencies for each genomic interval in the `gr` object. Lattice plots require a `data.frame`, and so we create a `data.frame` object using the method `dataFrame`. Additional arguments such as `maxgap` are passed to the `findOverlaps` function in the *GenomicRanges* package. Here, we frame each alteration by a genomic interval of 200kb by specifying `maxgap=200e3`.

```
> df <- dataFrame(gr, se, maxgap=500e3)
> head(df)
```

```
           x   lrr baf      id is.snp                     interval
8648 3193719  0.03   0 NA19007  FALSE chr8 interval 1, ID: NA19007
8649 3202945 -0.03   0 NA19007   TRUE chr8 interval 1, ID: NA19007
8650 3206554  0.07   0 NA19007  FALSE chr8 interval 1, ID: NA19007
8651 3209173  0.33   1 NA19007   TRUE chr8 interval 1, ID: NA19007
8652 3215911  0.30   0 NA19007  FALSE chr8 interval 1, ID: NA19007
8653 3223366  0.12   0 NA19007  FALSE chr8 interval 1, ID: NA19007
```

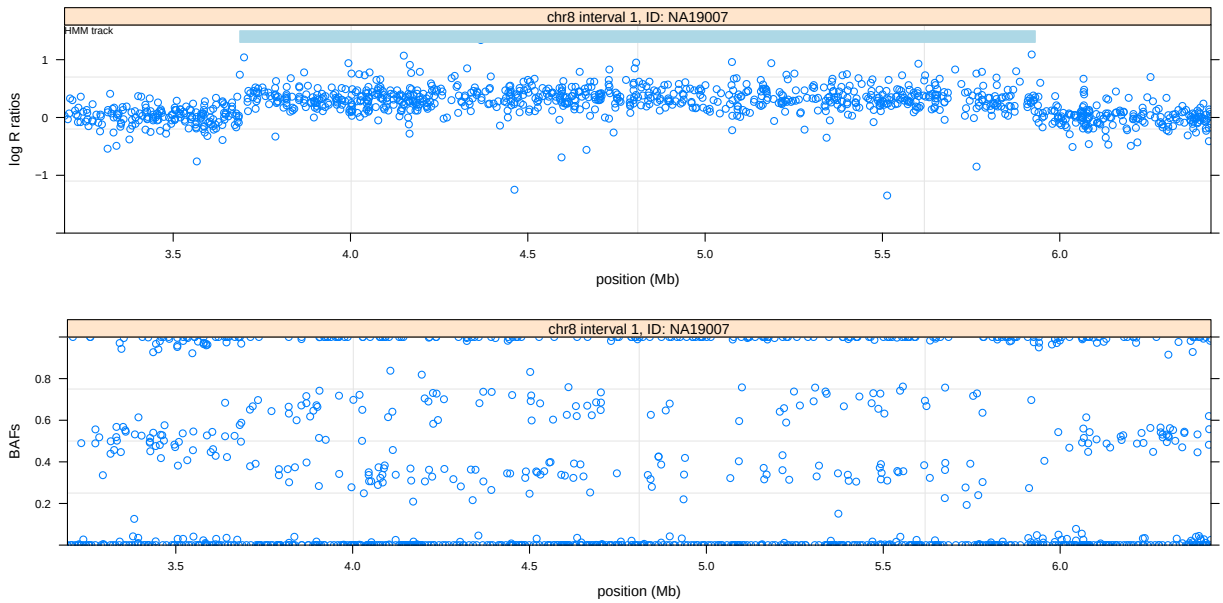

Figure 1: Plot of log R ratios for altered copy number states. Each panel displays one region with a copy number alteration predicted from the 6-state HMM with padding on each side. The light blue and orange shaded rectangles indicate duplications and hemizygous deletions, respectively, identified by the HMM.

Finally, we create separate `trellis` objects for the log R ratios and B allele frequencies using the wrapper function `latticeFigs`.

```
> colors <- c("red", "orange", "white", "white",
+             "lightblue", "blue")[state(gr)]
> figs <- latticeFigs(gr, df, colors=colors)
```

We can arrange the two trellis objects using the function `arrangeFigs` defined in the *SNPchip* package.

```
> library(grid)
> library(lattice)
> arrangeFigs(figs)
```

Parallelization

As the HMM is fit independently to each sample, parallelization is straightforward and will be performed automatically when multiple CPUs are available. For large datasets in which the assay data elements of the `BafLrrSetList` object are stored as `ff` objects on disk, only `ocSamples()` will be processed at a time. In the following unevaluated code chunk, we set up a parallel environment using the R packages *snow* and *foreach* and specify that we want to process the samples one at a time. Automatically, the two samples in the `oligoList` object are processed on separate CPUs.

```
> library(foreach)
> library(snow)
> library(doSNOW)
> cl <- makeCluster(2, type="SOCK")
> registerDoSNOW(cl)
> ocSamples(2)
```

```
> res2 <- hmm(oligoList, p.hom=0, TAUP=1e10, nupdates=5)

> stopCluster(cl)
```

Session Information

The version number of R and packages loaded for generating the vignette were:

- R version 3.1.0 (2014-04-10), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, grDevices, graphics, grid, methods, parallel, stats, utils
- Other packages: Biobase 2.24.0, BiocGenerics 0.10.0, Biostrings 2.32.0, DBI 0.2-7, GenomeInfoDb 1.0.2, GenomicRanges 1.16.3, IRanges 1.22.7, RColorBrewer 1.0-5, RSQLite 0.11.4, SNPchip 2.10.0, VanillaICE 1.26.1, XVector 0.4.0, crlmm 1.22.0, foreach 1.4.2, lattice 0.20-29, matrixStats 0.8.14, oligo 1.28.2, oligoClasses 1.26.0, pd.mapping50k.hind240 1.10.0, pd.mapping50k.xba240 1.10.0, preprocessCore 1.26.1
- Loaded via a namespace (and not attached): BiocInstaller 1.14.2, Matrix 1.1-3, R.methodsS3 1.6.1, Rcpp 0.11.1, RcppEigen 0.3.2.1.2, VGAM 0.9-3, affxparser 1.36.0, affyio 1.32.0, base64 1.1, bit 1.1-12, codetools 0.2-8, compiler 3.1.0, ellipse 0.3-8, expm 0.99-1.1, ff 2.2-13, illuminaio 0.6.0, iterators 1.0.7, msm 1.3, mvtnorm 0.9-99992, splines 3.1.0, stats4 3.1.0, survival 2.37-7, tools 3.1.0, zlibbioc 1.10.0