Copy number variant detection in exome sequencing data using exomeCopy

Michael Love
michaelisaiahlove@gmail.com

October 30, 2017

Abstract

exomeCopy is an R package implementing a hidden Markov model for predicting copy number variants (CNVs) from exome sequencing experiments without paired control experiments as in tumor/normal sequencing. It models read counts in genomic ranges using negative binomial emission distributions depending on a hidden state of the copy number and on positional covariates such as GC-content and background read depth. Normalization and segmentation are performed simultaneously, eliminating the need for preprocessing of the raw read counts.

Contents

1 Introduction

The exomeCopy package was designed to address the following situation:

- Target enrichment, such as exome enrichment, leads to non-uniform read depth, which is often correlated across samples.
- CNVs overlapping enriched regions can be detected as increases or decreases in read counts relative to "background" read depth, generated by averaging over a control set.
- Individual samples can be more or less correlated with background read depth and have different dependencies on GC-content.

While exome sequencing is not designed for CNV genotyping, it can nevertheless be used for finding CNVs which overlap exons and are not common in the control set. It can provide an independent data source to be used in combination with higher resolution array-based methods. In this vignette we show how to import experiment data, generate background read depth, and recover CNVs using exomeCopy . The model implemented in this package is described in [Love et al., 2011]. On CRAN, the R package ReadDepth can be used for CNV detection in whole genome. The methods in this package are not appropriate for cancer exome sequencing, where amplifications can be high and not necessarily integer values. The ExomeCNV package on CRAN is designed for tumor/normal paired exome sequencing data.

`exomeCopy` is designed to run on read counts from consecutive genomic ranges on a single chromosome, as it tries to identify higher or lower read depth relative to a baseline. We will use a wrapper function to loop the main function over multiple chromosomes and samples. Then these results can be compiled across chromosomes and samples.

2 Quick start

Here is an abbreviated example of the code required for importing data, running the model (note: on autosomal regions) and compiling results. For more details, please read the rest of the vignette and try the help for various functions, e.g. `?compileCopyCountSegments`.

```
> library(exomeCopy)
> target.file <- "targets.bed"
> bam.files <- c("/path/to/file1.bam", "/path/to/file2.bam",
+     "/path/to/file3.bam")
> sample.names <- c("sample1", "sample2", "sample3")
> reference.file <- "/path/to/reference_genome.fa"
> target.df <- read.delim(target.file, header = FALSE)
> target <- GRanges(seqname = target.df[, 1], IRanges(start = target.df[,
+     2] + 1, end = target.df[, 3]))
> counts <- target
> for (i in 1:length(bam.files)) {
+     mcols(counts)[[sample.names[i]]] <- countBamInGRanges(bam.files[i],
+         target)
+ }
> counts$GC <- getGCcontent(target, reference.file)
> counts$GC.sq <- counts$GC^2
> counts$bg <- generateBackground(sample.names, counts,
+     median)
> counts$log.bg <- log(counts$bg + 0.1)
> counts$width <- width(counts)
> fit.list <- lapply(sample.names, function(sample.name) {
+     lapply(seqlevels(target), function(seq.name) {
+         exomeCopy(counts[seqnames(counts) == seq.name],
```

```
+             sample.name, X.names = c("log.bg", "GC",
+                 "GC.sq", "width"), S = 0:4, d = 2)
+     })
+ })
> compiled.segments <- compileCopyCountSegments(fit.list)
```

3 Importing experiment data

The necessary genomic range information, sample read counts and positional covariates (background read depth and GC-content) should be stored in a *GRanges* object. The user must provide genomic ranges of targeted enrichment. For exome sequencing, one can use exon annotations. The exomeCopy package provides the following convenience functions: `subdivideGRanges` for subdividing the target ranges into nearly-equal sized ranges, `countBamInGRanges` for counting reads from a BAM file in the target ranges, `getGCcontent` for calculating GC content in the target ranges, and `generateBackground` for calculating the median read depth over a set of background samples.

To minimize memory size, we first demonstrate with toy data how to construct a *GRanges* object `toy.counts` from a BED file describing the targeted region, BAM files for the read mapping and a FASTA file for the reference sequence. For demonstration of running the main functions, we will use a prepackaged *GRanges* object `exomecounts` containing real exome sequencing read counts. We will then simulate CNVs in a *GRanges* object, `example.counts` and run `exomeCopy`.

3.1 Subdividing targeted regions

It is possible, but not necessary to subdivide large input ranges into multiple ranges of comparable width to the average input range. It is not necessary, because exomeCopy can use range width as a covariate for modeling read counts. `subdivideGRanges` divides the targeted genomic ranges into a set of ranges of nearly equal width, which exactly cover the original ranges. The ranges will be sorted and reduced if they are not already so. `subdivideGRanges` requires an input *GRanges* object and returns a *GRanges* object.

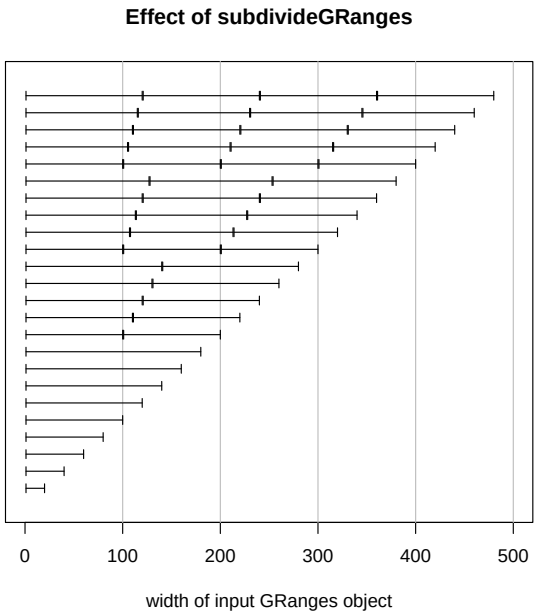
```
> library(exomeCopy)
> gr <- GRanges(seqname = "seq1", IRanges(start = 1, end = 345))
> subdivideGRanges(gr)
```

```
GRanges object with 3 ranges and 0 metadata columns:
      seqnames     ranges strand
         <Rle>  <IRanges>  <Rle>
  [1]     seq1 [  1, 115]      *
  [2]     seq1 [116, 230]      *
  [3]     seq1 [231, 345]      *
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

The default setting of `subdivideGRanges` is to divide an input range into ranges around $s = 100$bp, which is slightly less than the average exon width. Specifically, an input range of width w will be divided evenly into $\max(1, \lfloor w/s \rfloor)$ regions. We can visualize the effect of `subdivideGRanges` on ranges of increasing width.

```
> plot(0, 0, xlim = c(0, 500), ylim = c(0, 25), type = "n",
+     yaxt = "n", ylab = "", xlab = "width of input GRanges object",
+     main = "Effect of subdivideGRanges")
> abline(v = 1:5 * 100, col = "grey")
> for (i in 1:24) {
+     gr <- GRanges(seqname = "chr1", IRanges(start = 1,
+         width = (i * 20)))
```

```
+     sbd.gr <- subdivideGRanges(gr)
+     arrows(start(sbd.gr), rep(i, length(sbd.gr)), end(sbd.gr),
+         rep(i, length(sbd.gr)), length = 0.04, angle = 90,
+         code = 3)
+ }
```



Here we demonstrate reading in a targeted region BED file, converting to a *GRanges* object and the result from calling `subdivideGRanges`. Note that if the targeted region is read in from a BED file, one should add 1 to the starting position for representation as a *GRanges* object. It is important that the input ranges are sorted and non-overlapping, so we call `reduce` on the target *GRanges* object.

```
> target.file <- system.file("extdata", "targets.bed",
+     package = "exomeCopy")
> target.df <- read.delim(target.file, header = FALSE,
+     col.names = c("seqname", "start", "end"))
> target <- GRanges(seqname = target.df$seqname, IRanges(start = target.df$start +
+     1, end = target.df$end))
> target

GRanges object with 5 ranges and 0 metadata columns:
      seqnames     ranges strand
         <Rle>  <IRanges>  <Rle>
  [1]     seq1 [101, 250]      *
  [2]     seq1 [301, 650]      *
  [3]     seq2 [  1, 150]      *
  [4]     seq2 [401, 550]      *
  [5]     seq2 [701, 750]      *
  -------
  seqinfo: 2 sequences from an unspecified genome; no seqlengths

> target <- reduce(target, min.gapwidth = 0)
> target.sub <- subdivideGRanges(target)
> target.sub
```

```
GRanges object with 7 ranges and 0 metadata columns:
      seqnames     ranges strand
         <Rle>  <IRanges>  <Rle>
  [1]     seq1 [101, 250]      *
  [2]     seq1 [301, 417]      *
  [3]     seq1 [418, 533]      *
  [4]     seq1 [534, 650]      *
  [5]     seq2 [  1, 150]      *
  [6]     seq2 [401, 550]      *
  [7]     seq2 [701, 750]      *
  -------
  seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

3.2 Counting reads in genomic ranges

`countBamInGRanges` allows the user to count reads from a BAM read mapping file in genomic ranges covering the targeted region. The function takes as input the BAM filename and a *GRanges* object. It returns a vector of counts, representing the number of sequenced read starts (leftmost position regardless of strand) with mapping quality above a minimum threshold (default of 1) for each genomic range. It is also possible to count stranded read starts, all overlapping reads and only reads mapping to unique positions (see `?countBamInGRanges`).

Users should make sure the sequence names in the *GRanges* object are the same as the sequence names in the BAM file (which can be listed using `scanBamHeader` in the Rsamtools package). The BAM file requires a associated index file (see the man page for `indexBam` in the Rsamtools package). We will count reads using the subdivided genomic ranges in `target.sub` and store the counts as a new value column, `sample1`.

```
> bam.file <- system.file("extdata", "mapping.bam", package = "exomeCopy")
> scanBamHeader(bam.file)[[1]]$targets

seq1 seq2
 800  800

> seqlevels(target.sub)

[1] "seq1" "seq2"

> toy.counts <- target.sub
> sample.df <- data.frame(samples = "sample1", bam.files = bam.file,
+     stringsAsFactors = FALSE)
> for (i in 1:nrow(sample.df)) {
+     mcols(toy.counts)[[sample.df$samples[i]]] <- countBamInGRanges(sample.df$bam.files[i],
+         target.sub)
+ }
> toy.counts

GRanges object with 7 ranges and 1 metadata column:
      seqnames     ranges strand |   sample1
         <Rle>  <IRanges>  <Rle> | <integer>
  [1]     seq1 [101, 250]      * |        73
  [2]     seq1 [301, 417]      * |        59
  [3]     seq1 [418, 533]      * |        61
  [4]     seq1 [534, 650]      * |        54
  [5]     seq2 [  1, 150]      * |        80
  [6]     seq2 [401, 550]      * |        69
  [7]     seq2 [701, 750]      * |        31
  -------
  seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

3.3 Calculating GC-content

exomeCopy can model read counts from samples which are not perfectly correlated with background read depth using GC-content (ratio of G and C bases to total number of bases). The GC-content of DNA fragments is known to be a factor in the efficiency of high-throughput sequencing. To obtain the GC-content information, we only need the targeted ranges and a reference FASTA file.

```
> reference.file <- system.file("extdata", "reference.fa",
+     package = "exomeCopy")
> toy.counts$GC <- getGCcontent(target.sub, reference.file)
> toy.counts

GRanges object with 7 ranges and 2 metadata columns:
      seqnames     ranges strand |   sample1                GC
         <Rle>  <IRanges>  <Rle> | <integer>         <numeric>
  [1]     seq1 [101, 250]      * |        73 0.453333333333333
  [2]     seq1 [301, 417]      * |        59 0.427350427350427
  [3]     seq1 [418, 533]      * |        61 0.568965517241379
  [4]     seq1 [534, 650]      * |        54 0.521367521367521
  [5]     seq2 [  1, 150]      * |        80 0.573333333333333
  [6]     seq2 [401, 550]      * |        69              0.48
  [7]     seq2 [701, 750]      * |        31              0.36
  -------
  seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

3.4 Exome sequencing data from 1000 Genomes Project

For demonstrating the use of exomeCopy , we have provided a sample dataset, `exomecounts`, of exome sequencing read counts. The genomic ranges used in this dataset are generated from a small subset of the CCDS regions on chromosome 1 [Pruitt et al., 2009]. The regions are downloaded from the hg19 tables of the UCSC Genome Browser (`http://genome.ucsc.edu/cgi-bin/hgGateway`). Alternatively, one could use the GenomicFeatures package to download the CCDS regions. The original CCDS regions are subdivided using `subdivideGRanges` with default settings as in the example code above. The CCDS regions are convenient to use as genomic ranges for CNV detection in exome sequencing data, as they are often in the center of the targeted region in exome enrichment protocols and tend to have less variable coverage than the flanking regions.

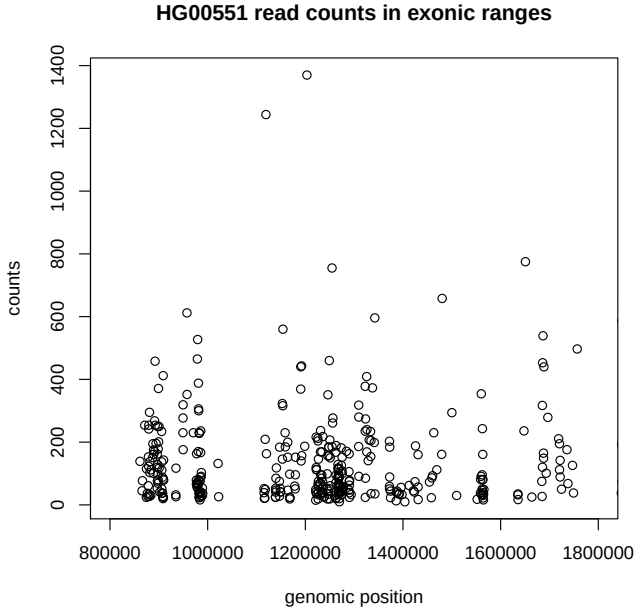
The read counts are taken from exome enriched, paired-end sequencing data of the 1000 Genomes Project for 16 samples of the PUR population [1000 Genomes Project Consortium, 2010]. The BAM read mapping files and descriptions of the experiments are available at the 1000 Genomes Project website (`http://www.1000genomes.org/data`). The ftp addresses used are listed in the file `1000Genomes_files.txt` in the `extdata` directory. The sample names are included as column names in the provided datatset.

```
> data(exomecounts)
> length(exomecounts)

[1] 1000

> head(exomecounts)

GRanges object with 6 ranges and 17 metadata columns:
      seqnames           ranges strand |        GC   HG00551
         <Rle>        <IRanges>  <Rle> | <numeric> <numeric>
  [1]     chr1 [861322, 861393]      * |    0.6389       139
  [2]     chr1 [865535, 865716]      * |    0.6484        45
```

```
  [3]     chr1 [866419, 866469]      * |    0.5882        77
  [4]     chr1 [871152, 871276]      * |     0.648       254
  [5]     chr1 [874420, 874509]      * |    0.6111        24
  [6]     chr1 [874655, 874840]      * |    0.6935       115
       HG00641   HG00731   HG00732   HG00734   HG00736   HG00739
     <numeric> <numeric> <numeric> <numeric> <numeric> <numeric>
  [1]       223       218       198       196        89       163
  [2]        90       135        93        74        47        62
  [3]       123       137       107        95        44        66
  [4]       285       348       391       226       219       217
  [5]        40        31        37        42        30        16
  [6]       169       225       171       124        96       131
       HG00740   HG01047   HG01048   HG01049   HG01054   HG01055
     <numeric> <numeric> <numeric> <numeric> <numeric> <numeric>
  [1]        44        53       148       208       178       161
  [2]        17        46        64        86       101        61
  [3]        28        37        70        91        96        82
  [4]        81       111       268       297       275       274
  [5]        10        12        34        35        35        16
  [6]        47        59       125       173       137       147
       HG01066   HG01067   HG01075
     <numeric> <numeric> <numeric>
  [1]       245       118        85
  [2]        80        51        35
  [3]       141        58        41
  [4]       260       206       133
  [5]        39        21        14
  [6]       201       108        68
  -------
  seqinfo: 1 sequence from hg19 genome; no seqlengths
```

The genomic ranges in `exomecounts` have been filtered such that only ranges with nonzero read count over the 16 samples are retained. The range of the 1000 genomic ranges is from 0.8 to 7.8 Mb on chromosome 1. Plotting the counts for one sample in a region of 1 Mb, one can observe both the irregular spacing of the ranges as well as the non-uniformities in read counts per range.

```
> plot(start(exomecounts), exomecounts$HG00551, xlim = c(8e+05,
+     1800000), xlab = "genomic position", ylab = "counts",
+     main = "HG00551 read counts in exonic ranges")
```



3.5 Generating background read depth

In order to run exomeCopy , we first generate background read depth. For demonstration of working with multiple chromosomes, we will copy the chr1 data as a new chromosome "chr1a".

```
> chr1a <- exomecounts
> seqlevels(chr1a) <- "chr1a"
> example.counts <- c(exomecounts, chr1a)
```

The background is generated by a simple function which performs 3 simple steps: 1) given a vector of names of samples to be used as background, extract the read counts data frame from the *GRanges* object, 2) divide each sample by its mean read count (column means), 3) calculate the median of these normalized read counts (row medians). The function can also be used to calculate any function across the rows of normalized read counts. For example, we also show how to calculate the variance of the normalized read counts, though we will not use covariate in this vignette.

Note that if the control set is a mix of males and females, the allosome portion of chrX will be a mixture of copy counts 1 and 2. In this case, one should created separate *GRanges* objects for chrX and chrY and run males and females separately with the appropriate expected copy count.

```
> exome.samples <- grep("HG.+", colnames(mcols(example.counts)),
+     value = TRUE)
> example.counts$bg <- generateBackground(exome.samples,
+     example.counts, median)
> example.counts$log.bg <- log(example.counts$bg + 0.1)
> example.counts$bg.var <- generateBackground(exome.samples,
+     example.counts, var)
```

We have already removed ranges have zero background coverage, but if we had not we could use the following lines. If there is zero coverage in the control set, this is most likely a region which is difficult to enrich, and we should not try to infer the copy number here.

```
> summary(example.counts$bg)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0915  0.3052  0.7549  0.9945  1.3187  8.5176
```

```
> example.counts <- example.counts[example.counts$bg >
+     0, ]
```

The relationship between read counts and GC-content over the ranges varies across protocols and samples. It can be roughly approximated per sample using second-order polynomial terms of GC-content. We store the square of GC-content as a new value column. Other functions of GC-content could be used as well. We also store the width of the ranges as a value column.

```
> example.counts$GC.sq <- example.counts$GC^2
> example.counts$width <- width(example.counts)
```

4 Introduction of the model

exomeCopy models the sample read counts on one chromosome, O, as emitted observations of a hidden Markov model (HMM), where the hidden state is the copy number of the sample. The emission distributions are modeled with negative binomial distributions, as the read counts from high-throughput sequencing are often overdispersed for the Poisson distribution.

$$O_t \sim \mathrm{NB}(\mu_{ti}, \phi)$$

$$\mu_{ti} = \frac{S_i}{d} e^{(x_{t*}\beta)}$$

The mean parameter, μ_{ti}, for genomic range t and hidden state i is a product of the possible copy number state S_i over the expected copy number d and an estimate of the positional effects. The positional effect modeling comes from an exponentiated $(x_{t*}\beta)$, where x_{t*} is the t-th row of X and β is a column vector of coefficients. The estimated positional effect is a combination of log background read depth, GC-content, range width, and any other useful covariates which are stored in the matrix X, with a row for each range and a column for each covariate. We use the log of background read depth so that the counts and read depth come out on the same scale.

The coefficients β are fit by the model, using the forward equations to assess the likelihood of the HMM over all hidden state paths. In this way, the normalization and segmentation steps are combined into one step of maximizing the likelihood of the parameters given the data. The Viterbi algorithm is then applied to provide the most likely path.

Table 1: Summary of notation

O_t	observed count of reads in the t-th genomic range
μ_{ti}	the mean parameter for f at range t in copy state i
ϕ	the dispersion parameter for f
S_i	the copy number for state i ($S_i \in \{0, 1, 2, \ldots\}$)
d	the expected background copy number (2 for diploid, 1 for haploid)
X	the matrix of covariates for estimating μ
Y	the matrix of covariates for estimating ϕ
β	the fitted coefficients for estimating μ
γ	the fitted coefficients for estimating ϕ

The base model uses a scalar estimate for the dispersion parameter ϕ (equivalent to $1/$`size` in the function `dnbinom`). An extension of this model also tries to fit the variance using positional information such as the log variance of the background read depth stored in a matrix Y.

$$O_t \sim \mathrm{NB}(\mu_{ti}, \phi_t)$$

$$\log(\phi_t) = y_{t*}\gamma$$

5 Simulating CNVs

Next we simulate CNVs in 8 samples on both chromosomes representing regions with a copy number of 0,1,3 and 4, relative to a background copy number of 2. This is accomplished by selecting a fraction of the reads contained in the CNV bounds and removing or doubling them.

```
> simulateCNV <- function(x, indices, multiply, prob) {
+     x[indices] <- x[indices] + multiply * rbinom(length(indices),
+         prob = prob, size = x[indices])
+     return(x)
+ }
> set.seed(2)
> cnv.probs <- rep(c(0.99, 0.5, 0.5, 0.95), each = 2)
> cnv.mult <- rep(c(-1, 1), each = 4)
> cnv.starts <- rep(c(1, 301, 601, 901), each = 2)
> for (i in 1:8) {
+     mcols(example.counts)[[exome.samples[i]]] <- simulateCNV(mcols(example.counts)[[exome.sampl
+         cnv.starts[i]:(cnv.starts[i] + 99), multiply = cnv.mult[i],
+         prob = cnv.probs[i])
+     mcols(example.counts)[[exome.samples[i]]] <- simulateCNV(mcols(example.counts)[[exome.sampl
+         1000 + cnv.starts[i]:(cnv.starts[i] + 99), multiply = cnv.mult[i],
+         prob = cnv.probs[i])
+ }
```

6 Running exomeCopy

6.1 Running exomeCopy on one sample, one chromosome

We can now run exomeCopy using the read counts for one of the simulated CNV samples. Later we will show how to write a simple wrapper function to loop the `exomeCopy` function over multiple chromosomes and samples. We specify the possible copy number values with `S` from 0 to 6 and the expected copy number state with `d` of 2.

```
> fit <- exomeCopy(example.counts[seqnames(example.counts) ==
+     "chr1"], sample.name = exome.samples[3], X.names = c("log.bg",
+     "GC", "GC.sq", "width"), S = 0:6, d = 2)
> show(fit)
```

```
ExomeCopy object
sample name: HG00731
percent normal state: 90%
```

After fitting, we call the function `copyCountSegments` on the *ExomeCopy* object, which provides the segmentation with the predicted copy number, the log odds of read counts being emitted from predicted copy count over normal copy count, the number of input genomic ranges contained within each segment, the number of targeted basepairs contained in the segments, and the name of the sample to help compile the segments across samples. The columns `log.odds` and `nranges` can be useful for filtering, see `?copyCountSegments` for details.

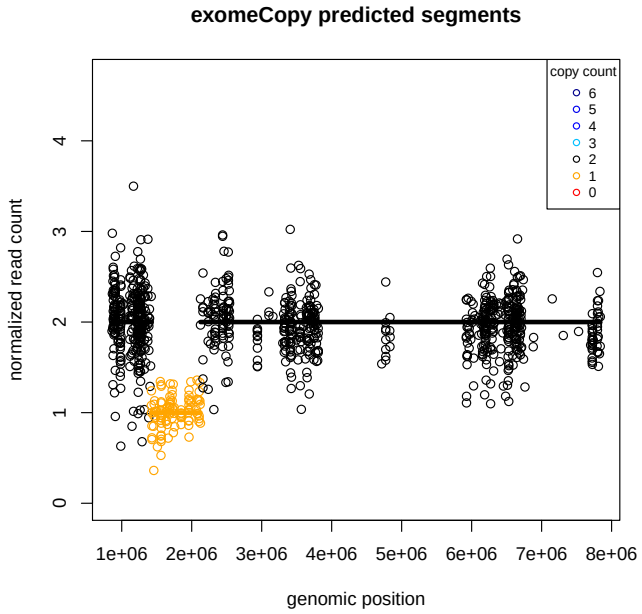
```
> copyCountSegments(fit)
```

```
GRanges object with 3 ranges and 5 metadata columns:
      seqnames             ranges strand | copy.count  log.odds
         <Rle>          <IRanges>  <Rle> |  <integer> <numeric>
  [1]     chr1 [ 861322, 1421615]      * |          2      0.00
  [2]     chr1 [1423243, 2125213]      * |          1    728.96
```

```
  [3]     chr1 [2125214, 7838229]      * |          2      0.00
       nranges targeted.bp sample.name
     <numeric>   <integer> <character>
  [1]       300        <NA>     HG00731
  [2]       100        <NA>     HG00731
  [3]       600        <NA>     HG00731
  -------
  seqinfo: 2 sequences from hg19 genome; no seqlengths
```

Calling `plot` on the *ExomeCopy* object draws segments of constant predicted copy number as colored horizontal lines and normalized read counts as points. If no colors are provided, the function will use red for copy counts less than `d` and blue for copy counts higher than `d`.

```
> cnv.cols <- c("red", "orange", "black", "deepskyblue",
+     "blue", "blue2", "blue4")
> plot(fit, col = cnv.cols)
```



6.2 Looping exomeCopy over multiple samples and chromosomes

In order to apply exomeCopy to a full dataset of multiple chromosomes and samples, we write a wrapper function `runExomeCopy`. We will only model duplications up to 4 copies. The wrapper construction allows for distribution of fitting each samples across workstations. Keep in mind that the non-PAR region of chrX, in male samples should be fit separately, with `d = 1`.

```
> runExomeCopy <- function(sample.name, seqs) {
+     lapply(seqs, function(seq.name) exomeCopy(example.counts[seqnames(example.counts) ==
+         seq.name], sample.name, X.names = c("log.bg",
+         "GC", "GC.sq", "width"), S = 0:4, d = 2))
+ }
```

We can now run exomeCopy using either `lapply` or using a function like `clusterApplyLB` from the snow package. We apply `runExomeCopy` for 8 samples over 2 chromosomes and the result is stored as a named list of named lists.

```
> seqs <- c("chr1", "chr1a")
> names(seqs) <- seqs
> samples <- exome.samples[1:8]
> names(samples) <- samples
> t0 <- as.numeric(proc.time()[3])
> fit.list <- lapply(samples, runExomeCopy, seqs)
> t1 <- as.numeric(proc.time()[3])
> time.elapsed <- as.numeric(t1 - t0)
> paste(round(time.elapsed), "seconds for", length(samples),
+     "samples,", round(sum(width(example.counts))/1000),
+     "kb of target")

[1] "13 seconds for 8 samples, 231 kb of target"

> paste("~", round(time.elapsed/60/(8 * sum(width(example.counts))) *
+     3.2e+07, 1), " minutes for 1 sample, 32 Mb of target",
+     sep = "")

[1] "~3.7 minutes for 1 sample, 32 Mb of target"
```

6.3 Compiling results, filtering and plotting CNVs

Using `compileCopyCountSegments` on the list of lists, we can compile the segments across samples into a *GRanges* object. We remove the segments with copy count 2, leaving the predicted CNVs for autosomal ranges.

```
> compiled.segments <- compileCopyCountSegments(fit.list)
> CNV.segments <- compiled.segments[compiled.segments$copy.count !=
+     2]
```

Some non-CNV ranges with too high or low read counts might be predicted to be CNV, despite the use of positional covariates and overdispersion in the emission distributions of the HMM. Often the majority of these cases can be removed by requiring a minimum number of ranges (for example requiring the `nranges` column returned by `copyCountSegments` to be greater than 5). The filtered CNV segments can be exported to tab-delimited files using `as.data.frame` and then `write.table`.

`copyCountSegments` also provides a column of log odds for each segment. This is the log ratio of the probability density function for the fitted predicted distribution over the fitted normal state distribution, then summed over all ranges in a segment (details of these are given in the help for `copyCountSegments`.) By definition these will be zero for the segments predicted normal and positive for the CNV segments.

```
> CNV.segments[1:6]

GRanges object with 6 ranges and 5 metadata columns:
      seqnames             ranges strand | copy.count  log.odds
         <Rle>          <IRanges>  <Rle> |  <integer> <numeric>
  [1]     chr1 [ 861322,  985971]      * |          0   1190.80
  [2]     chr1 [4715486, 4772259]      * |          4     17.61
  [3]    chr1a [ 861322,  985971]      * |          0   1192.64
  [4]     chr1 [ 861322,  985971]      * |          0   1069.61
  [5]     chr1 [1635263, 1635783]      * |          3     13.01
  [6]    chr1a [ 861322,  985971]      * |          0   1125.75
       nranges targeted.bp sample.name
     <numeric>   <integer> <character>
  [1]       100        <NA>     HG00551
  [2]         4        <NA>     HG00551
  [3]       100        <NA>     HG00551
  [4]       100        <NA>     HG00641
```

```
  [5]         3        <NA>     HG00641
  [6]       100        <NA>     HG00641
  -------
  seqinfo: 2 sequences from hg19 genome; no seqlengths
```

```
> table(CNV.segments$nranges)

  1   3   4   5  24  75  99 100
  2   1   1   1   1   1   2  13
```

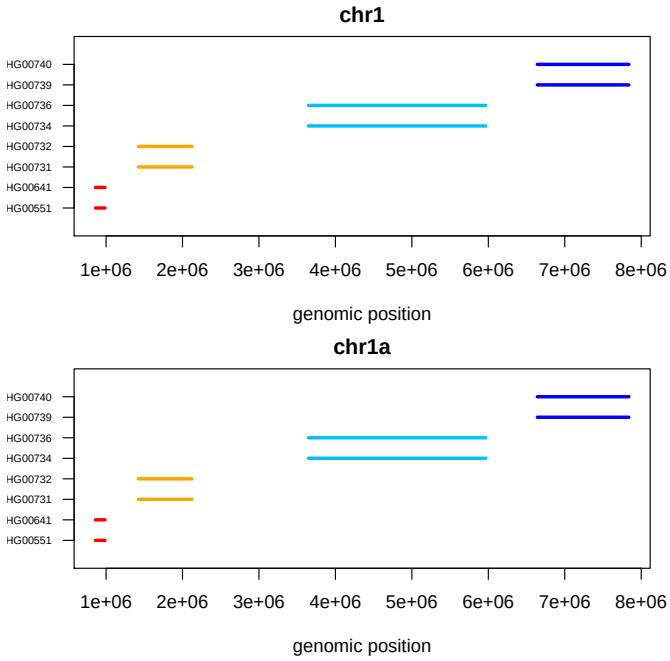
```
> CNV.segments <- CNV.segments[CNV.segments$nranges > 5]
```

To find overlaps within our calls across patients, we can use `findOverlaps` to create a matrix of overlaps between our predicted CNVs in different samples.

```
> CNV.overlaps.matrix <- as.matrix(findOverlaps(CNV.segments,
+     drop.self = TRUE))
> head(CNV.overlaps.matrix)

     queryHits subjectHits
[1,]         1           3
[2,]         2           4
[3,]         3           1
[4,]         4           2
[5,]         5           7
[6,]         6           8
```

`plotCompiledCNV` will plot the CNVs for a given sequence across samples if given an object returned by compileCopyCountSegments and the sequence name.

```
> par(mfrow = c(2, 1), mar = c(4, 3, 2, 1))
> cnv.cols <- c("red", "orange", "black", "deepskyblue",
+     "blue")
> plotCompiledCNV(CNV.segments = CNV.segments, seq.name = "chr1",
+     col = cnv.cols)
> plotCompiledCNV(CNV.segments = CNV.segments, seq.name = "chr1a",
+     col = cnv.cols)
```



6.4 Inspecting model parameters

Finally, we can inspect a number of fitted parameters in the *ExomeCopy* object, such as the β vector which is adjusted in optimizing the likelihood of the HMM. The β vector is initialized using a linear regression of counts on the covariates, and this and other initial parameter values can be accessed in the `init.par` slot of the *ExomeCopy* object. These coefficients are adjusted while optimizing the likelihood of the HMM. The final β vector and other final parameter values are accessible in the `final.par` slot.

```
> fit.list[[1]][[1]]@init.par$beta.hat
```

```
   intercept       log.bg           GC        GC.sq        width
 4.257472098  0.872083736  0.758468897 -1.065806294 -0.005971696
```

```
> fit.list[[1]][[1]]@final.par$beta
```

```
  intercept      log.bg          GC       GC.sq       width
 4.79500051  0.86625909  1.24278690 -1.24764491  0.01973945
```

7 Session info

```
> sessionInfo()
```

```
R version 3.4.2 (2017-09-28)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 16.04.3 LTS

Matrix products: default
BLAS: /home/biocbuild/bbs-3.6-bioc/R/lib/libRblas.so
LAPACK: /home/biocbuild/bbs-3.6-bioc/R/lib/libRlapack.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats4    parallel  stats     graphics  grDevices utils
[7] datasets  methods   base

other attached packages:
[1] exomeCopy_1.24.0     Rsamtools_1.30.0     Biostrings_2.46.0
[4] XVector_0.18.0       GenomicRanges_1.30.0 GenomeInfoDb_1.14.0
[7] IRanges_2.12.0       S4Vectors_0.16.0     BiocGenerics_0.24.0

loaded via a namespace (and not attached):
[1] bitops_1.0-6            zlibbioc_1.24.0
[3] BiocParallel_1.12.0     tools_3.4.2
[5] RCurl_1.95-4.8          compiler_3.4.2
[7] GenomeInfoDbData_0.99.1
```

References

1000 Genomes Project Consortium. A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061–1073, October 2010. ISSN 1476-4687. doi: 10.1038/nature09534. URL http://dx.doi.org/10.1038/nature09534.

Michael I. Love, Alena Myšičková, Ruping Sun, Vera Kalscheuer, Martin Vingron, and Stefan A. Haas. Modeling Read Counts for CNV Detection in Exome Sequencing Data. *Statistical Applications in Genetics and Molecular Biology*, 10(1), November 2011. ISSN 1544-6115. doi: 10.2202/1544-6115.1732. URL http://cmb.molgen.mpg.de/publications/Love_2011_exomeCopy.pdf.

Kim D. Pruitt, Jennifer Harrow, Rachel A. Harte, Craig Wallin, Mark Diekhans, Donna R. Maglott, Steve Searle, Catherine M. Farrell, Jane E. Loveland, Barbara J. Ruef, Elizabeth Hart, Marie-Marthe M. Suner, Melissa J. Landrum, Bronwen Aken, Sarah Ayling, Robert Baertsch, Julio Fernandez-Banet, Joshua L. Cherry, Val Curwen, Michael Dicuccio, Manolis Kellis, Jennifer Lee, Michael F. Lin, Michael Schuster, Andrew Shkeda, Clara Amid, Garth Brown, Oksana Dukhanina, Adam Frankish, Jennifer Hart, Bonnie L. Maidak, Jonathan Mudge, Michael R. Murphy, Terence Murphy, Jeena Rajan, Bhanu Rajput, Lillian D. Riddick, Catherine Snow, Charles Steward, David Webb, Janet A. Weber, Laurens Wilming, Wenyu Wu, Ewan Birney, David Haussler, Tim Hubbard, James Ostell, Richard Durbin, and David Lipman. The consensus coding sequence (CCDS) project: Identifying a common protein-coding gene set for the human and mouse genomes. *Genome research*, 19(7):1316–1323, July 2009. ISSN 1088-9051. doi: 10.1101/gr.080531.108. URL http://dx.doi.org/10.1101/gr.080531.108.