Overview of svcnvs package

Robert Scharpf, Daniel Bruhm

September 13, 2017

Introduction

The svcnvs package is used to identify deletions and amplicons from Whole Genome Sequencing (WGS) data through a combination of changes in normalized coverage and read pairs with aberrant spacing or orientation.

Additional packages required for this vignette are the `svfilters.hg19` package that contains various sequence filters for structural variant analyses and the `svalignments` package that contains wrappers for extracting properly- and improperly-paired reads from a BAM file. We also employ functions from `RSamtools` and `GenomicRanges`.

This package requires that normalized coverage is calculated for a set of bins in the package `svfilters.hg19` (or `svfilters.hg18`). See the `svpreprocess` vignette for comprehensive documentaion on generating normalized coverage. This process can take some time on large bam files so for the purpose of this tutorial we've distributed sample preprocessed coverage data from the `svpreprocess` package.

Preprocessing

Loading packages required for this analysis.

```
library(GenomicRanges)
library(Rsamtools)
library(svcnvs)
library(svfilters.hg19)
library(svalignments)
```

We first load an object from the `svpreprocess` package containing normalized and $\log_2$-transformed coverage estimates ($\log_2$ ratios) in non-overlapping 1kb bins along the genome (see `svpreprocess` to generate $\log_2$ ratios). The $\log_2$ ratios were multipled by 1000, rounded to the nearest integer, and saved as integers in a serialized R object to reduce the memory footpring. We append the $\log_2$ ratios to the `GRanges` object bins1kb from the `svfilters.hg19` package and remove chromosome Y from the analysis as this sample is of female origin. Note that in this instance we are using `svfilters.hg19` because we aligned our reads to the hg19 reference genome. There is an `svfilters.hg18` package for if your reads were aligned to hg18, and an `svfilters.hg38` package will be available in the future to support alignments to hg38.

```
ddir <- system.file("extdata", package="svpreprocess", mustWork=TRUE)
cov.file <- file.path(ddir, "preprocessed_coverage.rds")
log_ratio <- readRDS(cov.file)/1000
data(bins1kb, package="svfilters.hg19")
seqlevels(bins1kb, pruning.mode="coarse") <- paste0("chr", c(1:22, "X"))
## !requires column to be named log_ratio!
bins1kb$log_ratio <- log_ratio
```

Segmentation

Next, we segment the $\log_2$ ratios using the Circular Binary Segmentation algorithm to achieve estimated copy number states across the genome. So that our segmentation example runs quickly, we limit our analyses to two chromosomes and sample every 10th bin. Note, additional arguments can be passed to the `segment` function in the `DNAcopy` package.

```
bins_subset <- bins1kb
seqlevels(bins_subset, pruning.mode="coarse") <- c("chr1", "chr2")
bins_subset <- bins_subset[ seq(1, length(bins_subset), 50) ]
g <- segmentBins(bins_subset)
g
```

```
## GRanges object with 18 ranges and 1 metadata column:
##        seqnames                ranges strand |  seg.mean
##           <Rle>             <IRanges>  <Rle> | <numeric>
##    [1]     chr1   [  755001,  8726001]      * |    0.0961
##    [2]     chr1   [ 8776001,  9777001]      * |    0.7372
##    [3]     chr1   [ 9827001, 16799001]      * |     0.056
##    [4]     chr1   [16930001, 17233001]      * |    0.9933
##    [5]     chr1   [17288001, 72932001]      * |   -0.4317
##    ...      ...                   ...    ... .       ...
##   [14]     chr1 [245998001, 249201001]      * |    0.1331
##   [15]     chr2 [    36001,  11965001]      * |    0.2797
##   [16]     chr2 [ 12016001,  25228001]      * |   -0.4897
##   [17]     chr2 [ 25278001, 126334001]      * |    0.1806
##   [18]     chr2 [126384001, 243042001]      * |   -0.4086
##   -------
##   seqinfo: 2 sequences from hg19 genome
```

The result is a `GRanges` object with segment means of log-normalized coverage in the `seg.mean` column. Here, we load previously computed segments from the full dataset.

```
path <- system.file("extdata", package="svcnvs")
data(segments, package="svcnvs")
```

To generate this segments object on your own data use the `segmentBins` function.

```
segments <- segmentBins(bins1kb, param = SegmentParam())
```

We've created a `SegmentParam` class (see `?SegmentParam`) as a container for a subset of the parameters for the `segment` function in the `DNAcopy package`. Running `SegmentParam()` creates a SegmentParam obect with default values that we've found generally work well on most datasets for these following slots:

```
SegmentParam()
```

```
## An object of class "SegmentParam"
## Slot "alpha":
## [1] 0.001
##
## Slot "undo.splits":
## [1] "sdundo"
##
## Slot "undo.SD":
## [1] 2
##
## Slot "verbose":
```

```
## [1] 0
```

If you would like to alter the CBS parameters you can easily create your own `SegmentParam` object. For example, suppose you want to run CBS with alpha = 0.01 instead of 0.001:

```
x <- SegmentParam(alpha = 0.01, undo.splits = "sdundo", undo.SD = 2, verbose = 0)
segments <- segmentBins(bins1kb, param = x)
```

Any additional arguments to `DNAcopy::segment` can be incorporated into `segmentBins()` by using them as arguments to `segmentBins`. For example, a user may want to use min.with = 2 instead of the default value of 3:

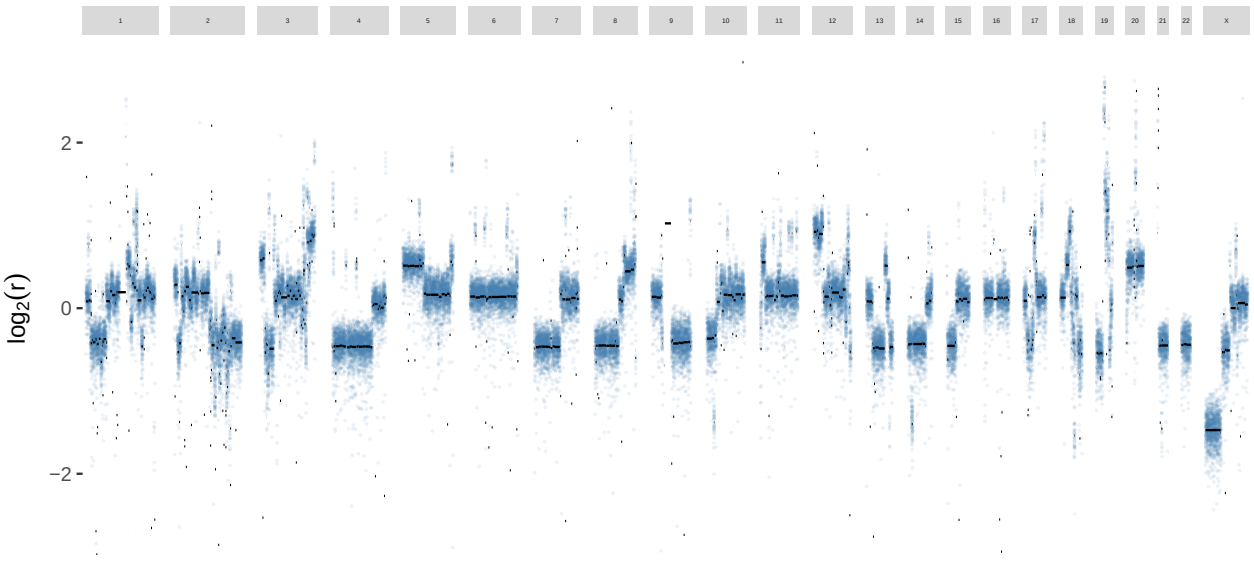
```
segments <- segmentBins(bins1kb, param = SegmentParam(), min.width = 2)
```

See `?DNAcopy::segment` for a complete list of parameters.

Plotting the genome

We routinely visualize the segmented $\log_2$ ratios of the entire genome to get an overview of the data. An example using the `ggplot2` library follows.

```
bins1kb <- sort(bins1kb)
lrr.df <- as.data.frame(bins1kb[seq(1, length(bins1kb), 50)])
seg.df <- as.data.frame(segments)
library(ggplot2)
chromlabels <- setNames(c(1:22, "X"), seqlevels(bins1kb))
ggplot(lrr.df, aes(start, log_ratio)) +
  geom_point(size=0.1, color="steelblue", alpha = 0.1) +
  geom_segment(data=seg.df,
               aes(x=start, xend=end, y=seg.mean, yend=seg.mean),
               color="black", inherit.aes=FALSE) +
  facet_grid(~seqnames, space="free", scales="free_x",
             labeller=as_labeller(chromlabels)) +
  theme(axis.text.x=element_blank(),
        panel.grid=element_blank(),
        panel.background=element_rect(fill="white"),
        axis.ticks.x=element_blank(),
        strip.text.x = element_text(size = 3)) +
  xlab("") +
  ylab(expression(log[2](r))) +
  ylim(c(-3, 3))
```



It is evident from visual inspection that this sample has numerous structural alterations. We will proceed to integrate read pair information with these segmented $\log_2$ ratios to strengthen CNV calls.

Deletions

In addtition to the segmented $\log_2$ ratios, the deletion analysis requires improperly paired reads. Below, we specify the complete file path to the BAM file used for this analysis that is provided by the `svbams` package. We extract improperly paired reads from the entire genome and an initial set of properly paired reads from a region on chromosome 15.

```
extdata <- system.file("extdata", package="svbams")
bam.file <- file.path(extdata, "cgov44t_revised.bam")
iparams <- improperAlignmentParams(what=c("flag", "mrnm", "mpos", "mapq"))
improper_rp <- getImproperAlignmentPairs(bam.file,
                                         param=iparams,
                                         build="hg19")
segs <- keepSeqlevels(segments, "chr15", pruning.mode="coarse")
del.gr <- reduce(segs[segs$seg.mean < hemizygousThr(DeletionParam())],
                 min.gapwidth=2000)
proper_rp <- properReadPairs(bam.file, gr=del.gr, DeletionParam())
read_pairs <- list(proper_del=proper_rp, improper=improper_rp)
```

We collect the bin-level summaries ($\log_2$ ratios), the segmentation data, and the read pair data in a single list object:

```
pdata <- preprocessData(bam.file=bam.file,
                        genome="hg19",
                        bins=bins1kb,
                        segments=segments,
                        read_pairs=read_pairs)
```

Below, we call segments as homozygous deletion (homozygous), homozygous deletion supported by improperly paired reads (homozygous+), hemizygous deletion (hemizygous), and hemizygous deletion supported by improperly paired reads (hemizygous). For the purpose of identifying somatic deletions without a matched normal, we exclude hemizygous deletions that are not supported by improperly paired reads. With this toy

dataset, we identify 16 possible homozygous deletions and the calls are both `homozygous` except onw which is `homozygous+`.

```
deletions <- sv_deletions(preprocess=pdata)
```

```
## Revising junctions...
```

```
## Refining homozygous boundaries by spanning hemizygous+
```

```
variant(deletions)
```

```
## GRanges object with 16 ranges and 2 metadata columns:
##        seqnames              ranges strand |  seg.mean      sample
##           <Rle>           <IRanges>  <Rle> | <numeric> <character>
##    sv1     chr2 [95731001, 95736001]      * |   -7.8007     CGOV44T
##    sv2     chr4 [22602001, 22606001]      * |   -7.8356     CGOV44T
##    sv3     chr4 [26291001, 26294001]      * |   -8.8232     CGOV44T
##    sv4     chr4 [40377001, 40439001]      * |   -8.2143     CGOV44T
##    sv5     chr4 [92858001, 92946001]      * |   -8.4734     CGOV44T
##    ...      ...                 ...    ... .       ...          ...
##   sv12    chr15 [63201015, 63209243]      * |   -8.6549     CGOV44T
##   sv13    chr17 [29448001, 29705001]      * |   -8.5644     CGOV44T
##   sv14    chr20 [37009001, 37014001]      * |   -8.0983     CGOV44T
##   sv15    chr22 [24874001, 24885001]      * |   -8.6562     CGOV44T
##   sv16     chrX [56812001, 56862001]      * |   -3.4914     CGOV44T
##   -------
##   seqinfo: 23 sequences from hg19 genome
```

```
calls(deletions)
```

```
##  [1] "homozygous"  "homozygous"  "homozygous"  "homozygous"  "homozygous"
##  [6] "homozygous"  "homozygous"  "homozygous"  "homozygous"  "homozygous"
## [11] "homozygous"  "homozygous+" "homozygous"  "homozygous"  "homozygous"
## [16] "homozygous"
```

The improperly-paired reads supporting the `homozygous+` call can be extracted as a `GAlignmentPairs` object from the twelth element of the `StructuralVariant` object.

```
improper(deletions[12])
```

```
## GAlignmentPairs object with 51 pairs, strandMode=1, and 0 metadata columns:
##       seqnames strand   :               ranges  --               ranges
##          <Rle>  <Rle>   :            <IRanges>  --            <IRanges>
##   i44    chr15      -   : [63209313, 63209412]  -- [63200507, 63200606]
##   i45    chr15      -   : [63209323, 63209422]  -- [63200525, 63200624]
##   i13    chr15      +   : [63200600, 63200699]  -- [63209331, 63209430]
##   i14    chr15      +   : [63200600, 63200699]  -- [63209363, 63209462]
##   i47    chr15      -   : [63209326, 63209425]  -- [63200607, 63200706]
##   ...      ...    ... ...                 ... ...                  ...
##   i32    chr15      +   : [63200745, 63200844]  -- [63209386, 63209485]
##   i33    chr15      +   : [63200749, 63200848]  -- [63209349, 63209448]
##   i60    chr15      -   : [63209455, 63209554]  -- [63200750, 63200849]
##   i34    chr15      +   : [63200757, 63200856]  -- [63209357, 63209456]
##   i63    chr15      -   : [63209477, 63209576]  -- [63200854, 63200953]
##   -------
##   seqinfo: 23 sequences from hg19 genome
```

Amplicons

Amplicons can be identified using the same list data structure for the preprocessed data we called `pdata` as was used in the *Deletions* section.

```
ag <- sv_amplicons2(pdata, params=ampliconParams())
```

Note the object returned by `sv_amplicons` is a graph where the nodes are the individual amplicons and the edges are links between amplicons given by improperly paired reads. By default, with 30x coverage we require at least 5 improperly paired reads to support an edge. See `?ampliconParams` for customing these settings.

Plotting deletions

We will use the *ggplot2* and *gridExtra* packages for plotting the deletions.

```
library(ggplot2)
suppressPackageStartupMessages(library(gridExtra))
library(scales)
```

In the following code chunk, we extract the genomic coordinates for a deletion stored in a *StructuralVariant* object distributed with this package. To view the deletion in the context of the surrounding region, we create a second *GRanges* object that includes 200kb of the flanking genome on each side of the deletion.

```
data(deletion)
roi <- variant(deletion) # region of interest
seqlevels(roi, pruning.mode="coarse") <- "chr15"
expand <- 200e3
roi2 <- GRanges(seqnames(roi), IRanges(start(roi)-expand,
                                       end(roi) + expand))
seqinfo(roi2) <- seqinfo(roi)
```

Next, we subset the views object to contain only the genomic bins in the region of interest defined above. In addition, we create a data.frame containing all the segments for this particular chromosome and sample and a data.frame containing the preprocessed coverage.

```
segs <- keepSeqlevels(segments, seqlevels(roi), pruning.mode="coarse")
segs.df <- as(segs, "data.frame")
hom.plus <- subsetByOverlaps(bins1kb, roi2)
df <- data.frame(logr=hom.plus$log_ratio,
                 start=start(hom.plus))
```

We restrict the y-axis limits to a suitable range for visualizing the log ratios, thresholding log ratios that are extreme. We highlight the region identified by the segmentation in the ggplot graphic (the boundaries for the deletion are subsequently revised by the improperly paired reads as described in the next section).

```
ylim <- c(-9, 2)
df$logr <- svpreprocess::threshold(df$logr, ylim)
brks <- pretty(df$start, n=8)
region <- subsetByOverlaps(segs, roi)
region <- region[region$seg.mean < -1]
region <- as.data.frame(region)
xlim <- c(start(roi2), end(roi2))

A <- ggplot(df, aes(start, logr)) +
  geom_point(size=1, color="gray50") +
```

```
  scale_x_continuous(expand=c(0,0), breaks=brks, labels=brks/1e6)+
  scale_y_continuous(expand=c(0,0)) +
  geom_segment(data=segs.df,
               aes(x=start, xend=end, y=seg.mean, yend=seg.mean),
               size=1) +
  coord_cartesian(xlim=xlim, ylim=ylim) +
  ylab(expression(log[2]~ratio)) +
  geom_rect(data=region,
            aes(xmin=start, xmax=end, ymin=-Inf, ymax=+Inf),
            fill="steelblue", color="transparent", alpha=0.3,
            inherit.aes=FALSE) +
  theme(axis.text=element_text(size=10),
        axis.text.x=element_blank()) + xlab("") +
  annotate("text", x=xlim[1] + 15e3, y=-8, label="chr15", size=3)
A1 <- ggplotGrob(A)
```

Plotting rearranged read pairs from a deletion object

In addition to the log ratios, we would like to visualize the rearranged read pairs (read pairs with aberrant spacing or orientation with respect to the reference genome) that support the deletion. The rearranged read pairs supporting the deletion are encapsulated in the *deletion* object that we already loaded. First, we pull read pairs flanking the candidate deletion that have normal spacing and orientation. Because there are typically a large number of the normal read pairs, we thin these using the function *thinReadPairs*. Next, we melt these reads into a *data.frame* useful for plotting.

```
rps <- thinReadPairs(deletion)
rps <- svcnvs:::meltReadPairs(rps)
```

We again use *ggplot* to plot the data. Note the vertical dashed lines depict the more precise boundaries of the deletion made possible by the improperly paired (rearranged) reads.

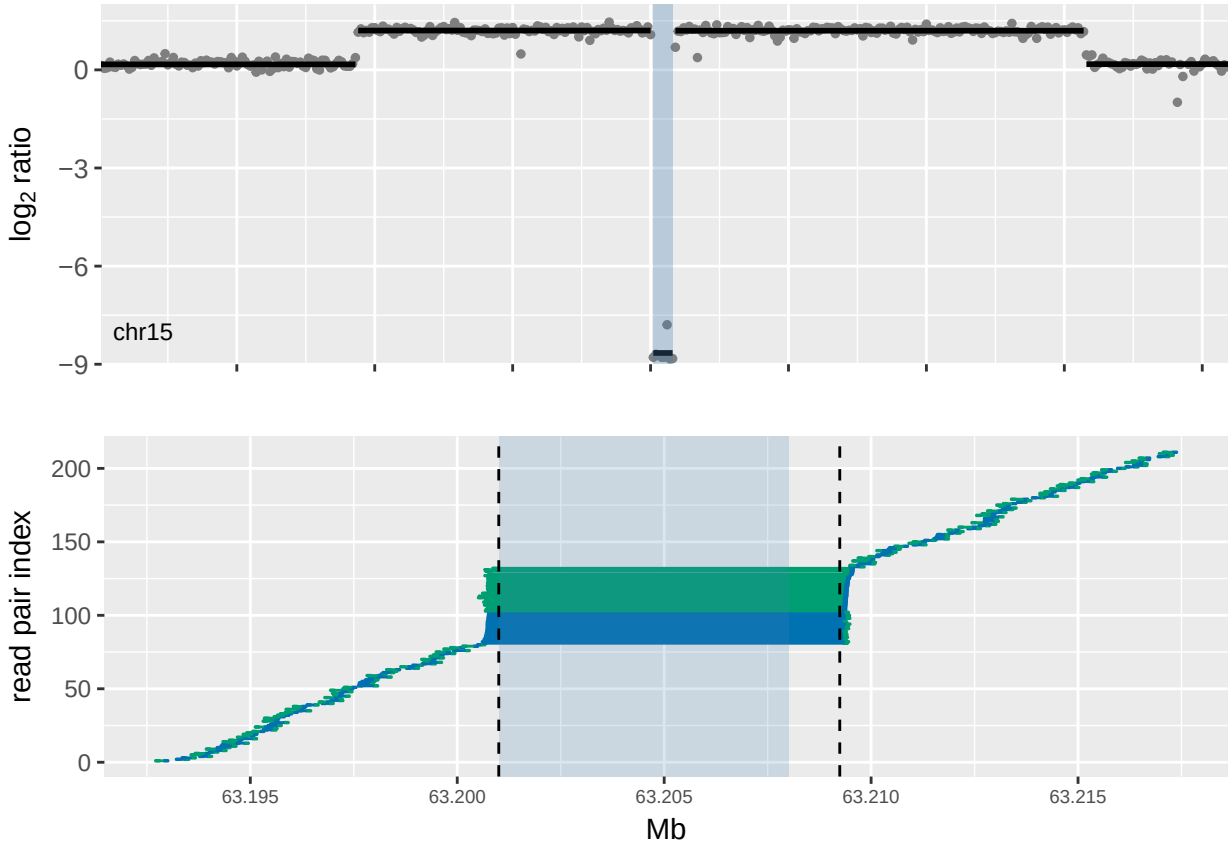
```
colors <- c("#0072B2", "#009E73")
p <- ggplot(rps, aes(ymin=readpair-0.2, ymax=readpair+0.2,
                xmin=start/1e6, xmax=end/1e6, color=read,
                fill=read, group=readpair)) +
  geom_rect() +
  xlim(c(min(rps$start), max(rps$end))/1e6) +
  geom_line(aes(x=start/1e6, y=readpair)) +
  ylab("read pair index") +
  scale_x_continuous(breaks=pretty_breaks(5)) +
  geom_rect(data=region,
            aes(xmin=start/1e6, xmax=end/1e6, ymin=-Inf, ymax=+Inf),
            fill="steelblue", color="transparent", alpha=0.2,
            inherit.aes=FALSE) +
  scale_color_manual(values=colors) +
  scale_fill_manual(values=colors) +
  xlab("Mb") +
  theme(axis.text.x=element_text(size=7)) +
  guides(fill=FALSE, color=FALSE) +
  geom_vline(xintercept=c(start(roi)/1e6, end(roi)/1e6), linetype="dashed")
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which
## will replace the existing scale.
```

```
B <- ggplotGrob(p)
```

Finally, we make a composite graphic of the log ratios and rearranged reads. Note, the vertical dashed lines show the revised deletion boundaries using the improperly-paired reads that flank the new sequence junction formed as a result of the deletion.

```
grid.arrange(A1, B, ncol=1)
```



Plotting amplicon graphs

See the amplicons vignette.