Segvis: A package for visualization of high throughout sequencing data along genomic segments

Rene Welch (welch@stat.wisc.edu) and Sündüz Keleş (keles@stat.wisc.edu)
Department of Statistics, University of Wisconsin - Madison
Madison, WI

October 2014

Contents

1 Overview

This vignette provides an introduction to the visualization of sequencing data by using the *Segvis* package. The minimum input to the package includes:

1. One or more bam files of aligned read data (e.g. from ChIP-seq experiments)
2. Coordinates for regions of interest

Segvis provides different tools to summarize and visualize these data, including but not limited to the following tasks:

- Extract read data of specified input regions
- Plot data from different files (conditions) accross the same set of regions, e.g. peak plots for ChIP-seq
- Plot central measures for a set of regions with the same width
- Subset this curves according to user defined annotations
- Plot the heatmap of signal curves accross regions separated by annotation

2 The basics

The package can be loaded with the command:

```
library(profile)
```

Different visualization of the data is done by the use of three following classes `profile`, `profileMatrix` and `profileMatrixList`. The first one is used to store the reads for a given bam file, the second is the one used to interact with the data and the third one is simply a list where all their elements are made from `profileMatrix` objects.

2.1 Creating a `profile` object

To create an `profile` object, there are needed two sources of data, the regions that are need to explore and a set of aligned reads. To define a `profile`, it is necessary to define some additional parameters:

```
name = "H3k27ac"
file ="../inst/extdata/encode_K562_H3k27ac_first3chr.sort.bam"
maxBw = 501
fl = 200
ourProfile = Profile(regionName = name,file = file,fileFormat= "bam",maxBandwidth = maxBw,fragLen = fl,r
ourProfile
```

```
## Profile for H3k27ac regions
## Fragment length: 200
## Max Bandwidth: 501
## Using reads files:
## ../inst/extdata/encode_K562_H3k27ac_first3chr.sort.bam
## **Not regions loaded**
```

Segvis is going to warn the user if there aren't any regions loaded. To load the regions a `GRanges` is necessary, to build one there are some instructions in A.

```
load(file = "../data/peaks_ctcf.RData")
windowExt = 1000
start(peaks_ctcf) = peaks_ctcf$summit - windowExt
end(peaks_ctcf) = peaks_ctcf$summit + windowExt
regions(ourProfile) = peaks_ctcf
ourProfile
```

```
## Profile for H3k27ac regions
## Fragment length: 200
## Max Bandwidth: 501
## Using reads files:
## ../inst/extdata/encode_K562_H3k27ac_first3chr.sort.bam
## Using regions for 3 chromosomes
## GRanges with 12461 ranges and 1 metadata column:
##           seqnames                ranges strand   |    summit
##              <Rle>             <IRanges>  <Rle>   | <integer>
##       [1]     chr1 [114888297, 114890297]      *   | 114889297
##       [2]     chr1 [225661805, 225663805]      *   | 225662805
##       [3]     chr1 [150951137, 150953137]      *   | 150952137
##       [4]     chr1 [ 17035453,  17037453]      *   |  17036453
##       [5]     chr1 [ 35317469,  35319469]      *   |  35318469
##       ...      ...                   ...    ... ...       ...
##   [12457]     chr3 [178822052, 178824052]      *   | 178823052
##   [12458]     chr3 [ 72083088,  72085088]      *   |  72084088
```

```
##   [12459]     chr3 [169290097, 169292097]      *   | 169291097
##   [12460]     chr3 [194826535, 194828535]      *   | 194827535
##   [12461]     chr3 [ 13432185,  13434185]      *   |  13433185
##   ---
##   seqlengths:
##    chr1 chr2 chr3
##      NA   NA   NA
```

2.2 Creating the coverage curves

To create a the coverage curves it is necessary to follow three steps `loadReads`, `matchReads` and `getCoverage`:

```
mc = 8
ourProfile = loadReads(ourProfile, mc)
ourProfile = matchReads(ourProfile, mc)
ourProfile = getCoverage(ourProfile, mc)
```

3 Some examples

3.1 Visualizing simple peak plots

...

3.2 Calculating the average coverage for a set of same width regions

Now it is possible to create the `profileMatrix`. When calculating the `profileMatrix`, the user should add a `bandwidth` parameter, which is used to smooth the coverage curves for each peak. To avoid innecessary repetitions of the previous 3 steps for different bandwidths, a parameter called `maxBandwidth` was added, thus it is possible to compare the coverage curve for different bandwidths:

```
ourProfileMatrix1 = ProfileMatrix(ourProfile, 1, mc)
ourProfileMatrix2 = ProfileMatrix(ourProfile, 51, mc)
ourProfileMatrix3 = ProfileMatrix(ourProfile, 301, mc)
ourProfileList = ProfileMatrixList(ourProfileMatrix1, ourProfileMatrix2, ourProfileMatrix3)
names(ourProfileList) = c("1", "51", "301")
```

To compare the different averaged profile curves we do, in the second case we are using a parameter coord to set the genomic coordinates used:

```
p1 = plot.profiles(ourProfileList)
p2 = plot.profiles(ourProfileList, coord = -windowExt:windowExt)
```

Lets notice that we changed the coordinates from the left to the right plot in figure 1. In the first case, we used the columns number in the matrix, while in the second we are using a relative distance to the middle of the regions.

3.3 Changing the `trim` parameter

However, this is a simple example. A lot of times it's necessary to compare the curves generated under different conditions. For this example, we are going to calculate the coverage curves using different read samples but the same set of regions. The reads corresponds in this case to the following histones:

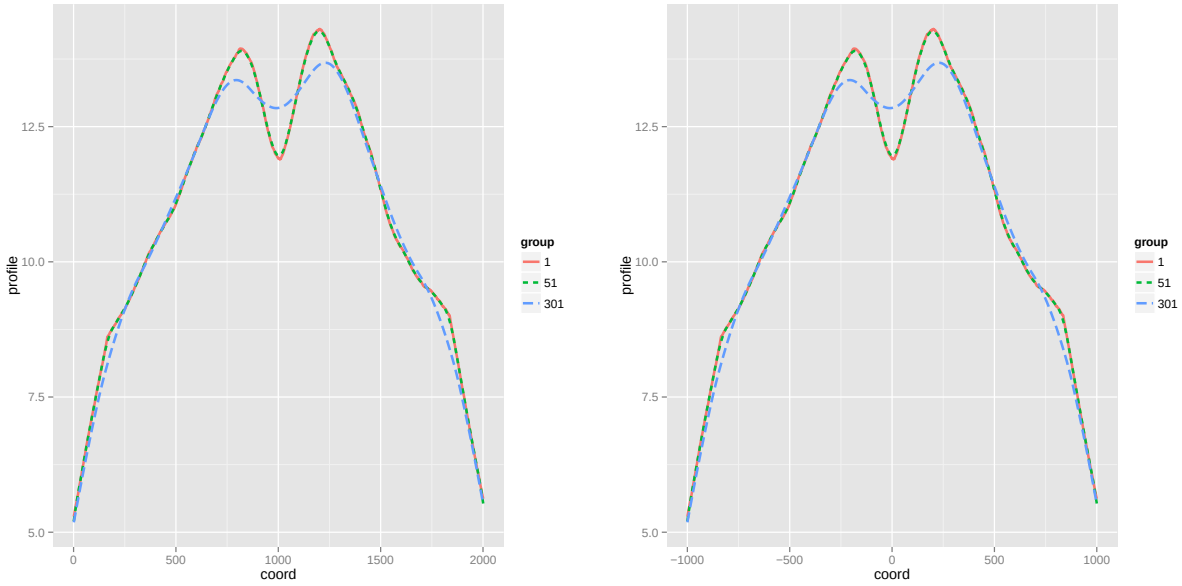

Figure 1: Profile plots smoothed using several bandwidths

- H3k27ac
- H3k4me1
- H3k4me3

Just as in subsection 2.2 we apply the same methods to build our `profile` objects:

```
ourProfiles = mapply(Profile, names, files, MoreArgs = list("bam", maxBw, fl, ""), SIMPLIFY = FALSE)
names(ourProfiles) = names
ourProfiles = lapply(ourProfiles, function(x, peaks_ctcf) {
    regions(x) = peaks_ctcf
    return(x)
}, peaks_ctcf)
ourProfiles = lapply(ourProfiles, loadReads, mc)
ourProfiles = lapply(ourProfiles, matchReads, mc)
ourProfiles = lapply(ourProfiles, getCoverage, mc)
```

Then, for each sample we build the `profileMatrix`, and we gather in a `profileMatrixList`:

```
ourMatrices = lapply(ourProfiles, ProfileMatrix, 251, mc)
names(ourMatrices) = names
ourList_notScaled = ProfileMatrixList(ourMatrices)
```

For the case when the depth of our samples may be very different among them we can also normalize our curves as if they were generated by sample with a million reads:

```
ourMatrices = lapply(ourMatrices, normalize.matrix)
ourList = ProfileMatrixList(ourMatrices)
```

And we plot them:

```
q1 = plot.profiles(ourList_notScaled, coord = -windowExt:windowExt)
q2 = plot.profiles(ourList, coord = -windowExt:windowExt)
```

Finally there maybe outlier curves for some of the regions. Therefore we can use the `trim` parameter:

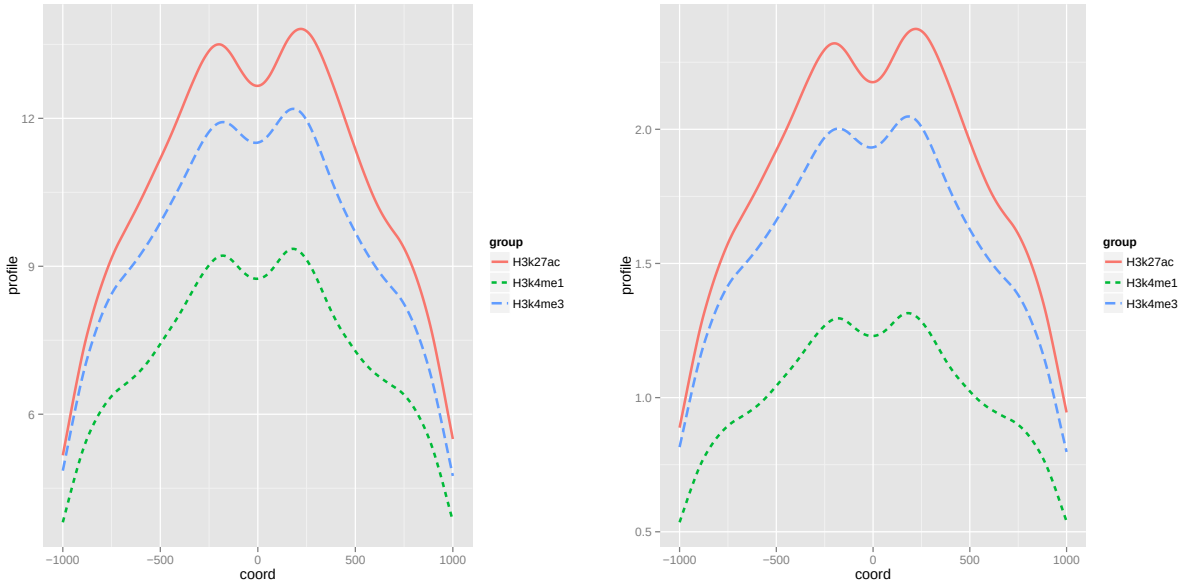

Figure 2: Profile plots of multiple samples with (right) and without (left) normalizing the curves

```
q3 = plot.profiles(ourList,coord= -windowExt:windowExt,trim = 1)
q4 = plot.profiles(ourList,coord= -windowExt:windowExt,trim = .25)
```

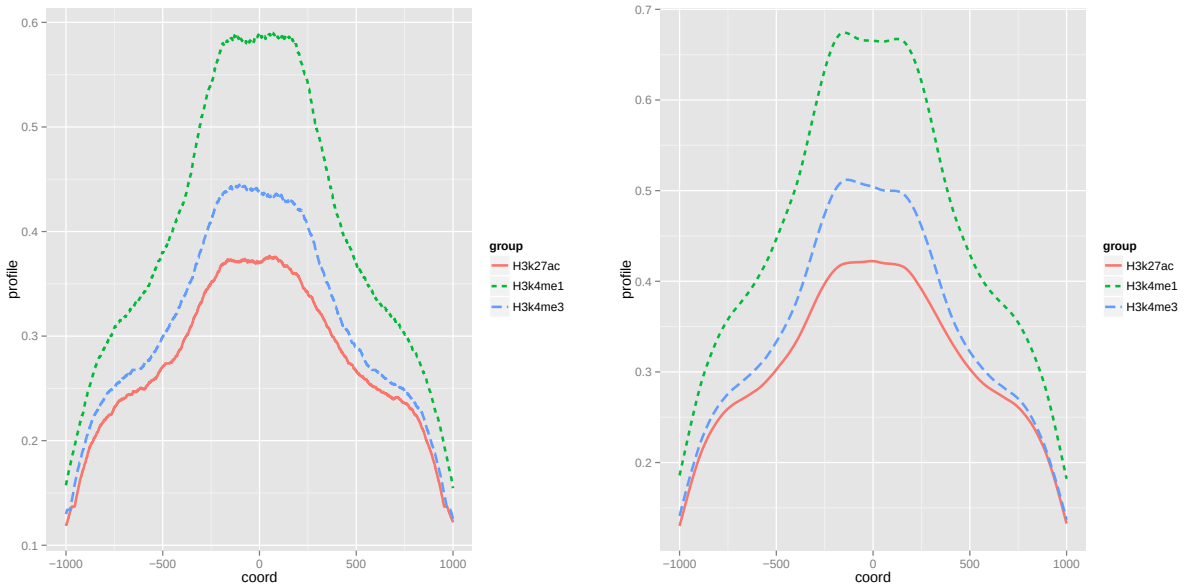

Figure 3: Averaged profile plots of multiple samples using the sample median (left) and a 75% trimmed mean (right)

4 Modifying the appearance of the plots

By applying the `plot.profiles` method, the *Segvis* packages generates a `ggplot` object, thus the user can modify the output by using some their commands.

For example we can change the colors of the curves, change the background, add a vertical line around the summit position or change the axis labels.

```
q2 + theme(legend.position = "bottom") + scale_color_brewer(palette = "Dark2")
q2 + theme_bw() + theme(legend.position = "bottom") + scale_color_brewer(palette = "Accent") +
    geom_vline(xintercept = 0, linetype = "dashed") + scale_linetype_manual(values = rep("solid",
    3)) + ylab("Nomalized signal") + xlab("Distance from summit")
```

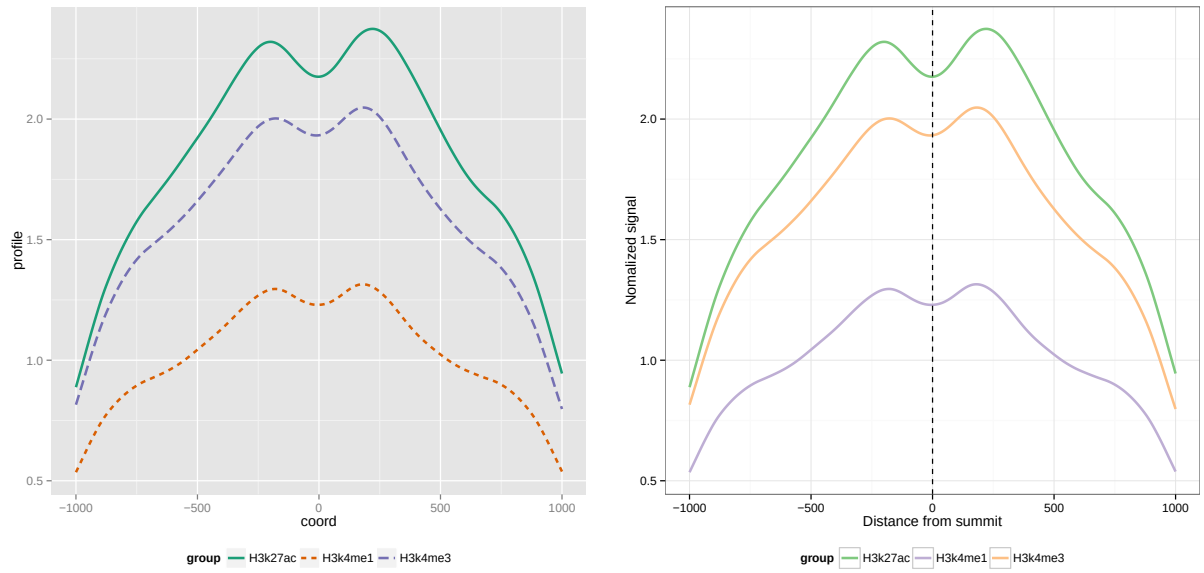

Figure 4: Format modification of average coverage plots

5 Adding columns to matrix and subsetting

In the previous examples, we explore the plotting possibilities of *Segvis*. However, *Segvis* have also methods designed to explore the data. Consider a normalized `profileMatrixList` generated as in 3.3, then for each `profileMatrix` we can add a column as follows:

```
  ourList = lapply(ourList,function(x,dnase_regions){
    regions(x)$dnase = countOverlaps(regions(x),dnase_regions)
    return(x)},dnase_regions)
```

where `countOverlaps` is a method from *GenomicRanges*. In this case, we are obtaining the number of overlaps of each peak in `regions(ourList[[i]])` with a `GRanges` object containing dnase sensitive regions in the first 3 chromosomes of the human genome. For this example, we added another annotation element as the number of overlaps between our peaks and Pol2b peaks. We can subset a `profileMatrix` using the method `subset.pm` or directly plot the average coverage curves as:

```
p = plot.profiles(ourMatrices, coord = -windowExt:windowExt)
p1 = plot.profiles(ourMatrices, coord = -windowExt:windowExt, condition = dnase > 0 & pol2b >
    0)
```

```
p2 = plot.profiles(ourMatrices, coord = -windowExt:windowExt, condition = dnase > 0 & pol2b ==
    0)
p3 = plot.profiles(ourMatrices, coord = -windowExt:windowExt, condition = dnase == 0 & pol2b >
    0)
```

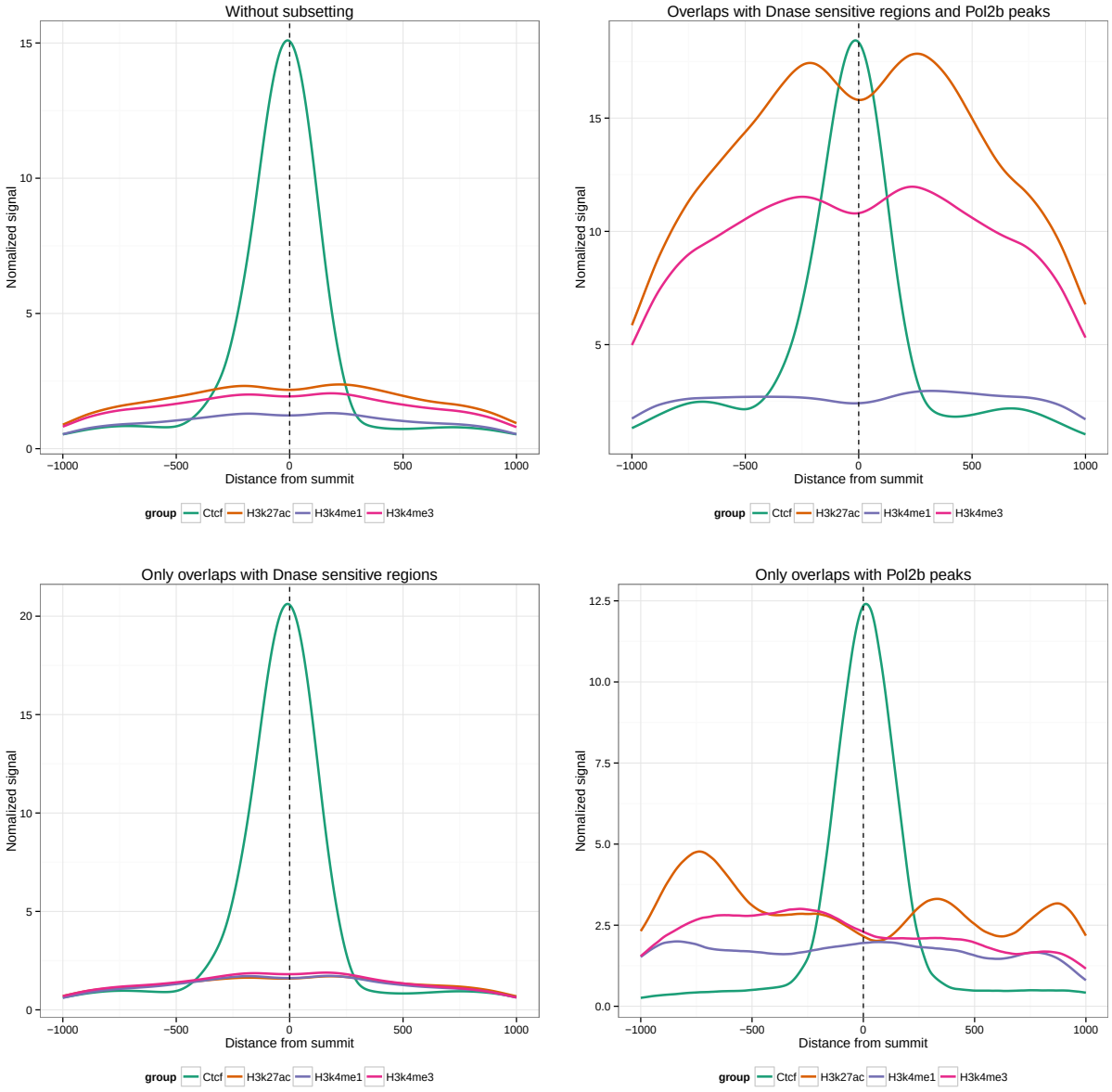

Figure 5: Coverage mean plot for several subsets

A Converting from extended bed format to a GRanges object

For this example we are going to read from a **bed** file, convert it to GRanges and save it as 'data/peaks.RData'.

```
bed_content = read.table(file = "../inst/extdata/encode_K562_Ctcf_peaks_first3chr.narrowPeak",
    stringsAsFactors = FALSE)
head(bed_content)
```

```
##     V1        V2        V3 V4 V5 V6    V7 V8    V9 V10
## 1 chr1 114889057 114889538  .  0  . 671.5 -1 4.572 240
## 2 chr1 225662556 225663044  .  0  . 632.5 -1 4.572 249
## 3 chr1 150951878 150952345  .  0  . 601.5 -1 4.572 259
## 4 chr1  17036198  17036690  .  0  . 585.6 -1 4.572 255
## 5 chr1  35318207  35318710  .  0  . 520.4 -1 4.572 262
## 6 chr1 204776085 204776784  .  0  . 519.1 -1 4.572 376
```

The **bed** files are going to be loaded as a table with 10 columns, where the first column is the chromosome, the second the start and the third column the end of the region. It may be of interest that in the case of **narrowPeak** format, the 10th column is the number of bp to the right of the start position where the summit of the peak is located.

```
peaks_ctcf = GRanges(seqnames = bed_content[, 1], ranges = IRanges(start = bed_content[, 2],
    end = bed_content[, 3]), strand = "*")
peaks_ctcf$summit = start(peaks_ctcf) + bed_content[, 10]
save(list = "peaks_ctcf", file = "../data/peaks_ctcf.RData")
peaks_ctcf
```

```
## GRanges with 12461 ranges and 1 metadata column:
##           seqnames                 ranges strand   |    summit
##              <Rle>              <IRanges>  <Rle>   | <integer>
##       [1]     chr1 [114889057, 114889538]      *   | 114889297
##       [2]     chr1 [225662556, 225663044]      *   | 225662805
##       [3]     chr1 [150951878, 150952345]      *   | 150952137
##       [4]     chr1 [ 17036198,  17036690]      *   |  17036453
##       [5]     chr1 [ 35318207,  35318710]      *   |  35318469
##       ...      ...                    ...    ... ...       ...
##   [12457]     chr3 [178822840, 178823264]      *   | 178823052
##   [12458]     chr3 [ 72083876,  72084300]      *   |  72084088
##   [12459]     chr3 [169290885, 169291309]      *   | 169291097
##   [12460]     chr3 [194827323, 194827747]      *   | 194827535
##   [12461]     chr3 [ 13432973,  13433397]      *   |  13433185
##   ---
##   seqlengths:
##    chr1 chr2 chr3
##      NA   NA   NA
```

B SessionInfo

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

- R version 3.1.1 (2014-07-10), x86_64-redhat-linux-gnu
- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_US.UTF-8, LC_COLLATE=en_US.UTF-8, 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, graphics, grDevices, methods, parallel, stats, utils

- Other packages: BiocGenerics 0.10.0, Biostrings 2.32.1, BSgenome 1.32.0, GenomeInfoDb 1.0.2, GenomicAlignments 1.0.6, GenomicRanges 1.16.4, ggplot2 1.0.0, IRanges 1.22.10, knitr 1.6, profile 1.2, rbamtools 2.10.0, Rsamtools 1.16.1, XVector 0.4.0
- Loaded via a namespace (and not attached): base64enc 0.1-2, BatchJobs 1.4, BBmisc 1.7, BiocParallel 0.6.1, BiocStyle 1.2.0, bitops 1.0-6, brew 1.0-6, checkmate 1.4, codetools 0.2-8, colorspace 1.2-4, DBI 0.2-5, digest 0.6.4, evaluate 0.5.5, fail 1.2, foreach 1.4.2, formatR 1.0, grid 3.1.1, gtable 0.1.2, highr 0.3, iterators 1.0.7, labeling 0.3, MASS 7.3-33, munsell 0.4.2, plyr 1.8.1, proto 0.3-10, RColorBrewer 1.0-5, Rcpp 0.11.3, reshape2 1.4, RSQLite 0.11.4, scales 0.2.4, sendmailR 1.2-1, stats4 3.1.1, stringr 0.6.2, tools 3.1.1, zlibbioc 1.10.0