# Segvis: A package for visualization of high throughput 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

April 2015

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

*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 (SET or PET) ChIP-seq.
- Calculate and plot statistics(e.g. mean, median, variace, etc.) over a window around biologically meaningful coordinates (TSS, TFBS, etc.)
- Subset this regions according to user defined annotations.
- Plot the heatmap of signal curves accross regions separated by annotation.

# 2 How to use Segvis?

The package can be loaded with the command:

```
library(Segvis)
```

Different visualization of the data is done by the use of three following classes `segvis`, `segvis_block` and `segvis_block_list`. 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 made exclusively of `segvis_block` objects.

## 2.1 Building a set of regions for Segvis

The minimum input for the package includes:

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

The coordinates may be obtained by several means: Visual exploration in the genome browser, calling peaks from a ChIP-seq experiment, etc. To use *Segvis* it is necessary to load the regions of interest into a `segvis` object by formatting them as a *GRanges* object.

For example, if the peaks are saved in a **narrowPeak** file format[1], then we can load it into *R* by using:

```
peaks_file <- "../inst/extdata/peaks/encode_K562_Ctcf_peaks_first3chr.narrowPeak"
ctcf_peaks <- read.table(peaks_file)
head(ctcf_peaks,15)
```

```
##      V1        V2        V3 V4 V5 V6       V7 V8      V9 V10
## 1  chr1 114889057 114889538  .  0  . 671.4930 -1 4.57207 240
## 2  chr1 225662556 225663044  .  0  . 632.4595 -1 4.57207 249
## 3  chr1 150951878 150952345  .  0  . 601.5148 -1 4.57207 259
## 4  chr1  17036198  17036690  .  0  . 585.6260 -1 4.57207 255
## 5  chr1  35318207  35318710  .  0  . 520.4329 -1 4.57207 262
## 6  chr1 204776085 204776784  .  0  . 519.1454 -1 4.57207 376
## 7  chr1  33177715  33178175  .  0  . 514.7651 -1 4.57207 239
## 8  chr1  19239498  19239983  .  0  . 501.9843 -1 4.57207 230
## 9  chr1  38455665  38456081  .  0  . 496.4812 -1 4.57207 185
## 10 chr1 154989953 154990397  .  0  . 495.8747 -1 4.57207 221
## 11 chr1   9686983   9687432  .  0  . 489.9299 -1 4.57207 216
## 12 chr1 186344435 186344962  .  0  . 485.4386 -1 4.57207 239
## 13 chr1  26221873  26222297  .  0  . 481.1880 -1 4.57207 187
## 14 chr1  47902101  47902527  .  0  . 476.5053 -1 4.57207 185
## 15 chr1 109806165 109806687  .  0  . 476.2783 -1 4.57207 236
```

Then to convert it into a *GRanges* object we can use:

```
K <- 2000
ctcf_gr <- GRanges(seqnames = ctcf_peaks$V1,
  ranges = IRanges(start = ctcf_peaks$V2,
    end = ctcf_peaks$V3),strand = "*")
ctcf_gr <- ctcf_gr[order(ctcf_peaks$V7,decreasing=TRUE)[1:K]]
ctcf_gr
```

```
## GRanges object with 2000 ranges and 0 metadata columns:
##          seqnames                ranges strand
##             <Rle>             <IRanges>  <Rle>
```

[1] A description of several common file formats is given in https://genome.ucsc.edu/FAQ/FAQformat.html

```
##      [1]     chr1 [114889057, 114889538]      *
##      [2]     chr1 [225662556, 225663044]      *
##      [3]     chr3 [195577700, 195578158]      *
##      [4]     chr1 [150951878, 150952345]      *
##      [5]     chr2 [ 91814978,  91815458]      *
##      ...      ...                   ...    ...
##   [1996]     chr1 [ 43823756,  43824034]      *
##   [1997]     chr1 [205323464, 205323745]      *
##   [1998]     chr2 [ 56410741,  56410988]      *
##   [1999]     chr1 [152430955, 152431197]      *
##   [2000]     chr1 [226033541, 226033846]      *
##   -------
##   seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

A complete description of the meaning of each column in the peaks file is given in https://genome.ucsc.edu/FAQ/FAQformat.html#format12. Using the signal values (on the 7th column), we are going to consider the top 2000 peaks.

## 2.2 Creating a `segvis` object

To create a *segvis* object, it is necessary to specify the following parameters:

- `name` - The name of the `segvis` object.
- `regions` - The regions to be loaded, in our case those are `ctcf_gr`.
- `file` - The file were the reads of the experiment are stored.
- `maxBandwidth` - The upper bound of all the possible bandwidths used to smooth the coverage plots when creating a `segvis_block` object.
- `fragLen` - The fragment length used to extend the fragment reads. If it is defined as zero, then it would use the original read widths.
- `chr` - The chromosomes for which the `segvis` object is defined. There are a couple of predefined cases as 'human' or 'mouse' to automatically consider all chromosomes in those genomes.
- `isPET` - A logical indicator if the reads of the experiment are paired-ended. In this case, the `fragLen` parameter is ignored.

```
ctcf <- buildSegvis(name = "ctcf_peaks",
  file = "../inst/extdata/reads/encode_K562_Ctcf_first3chr_Rep1.sort.bam",
  maxBandwidth = 101,fragLen = 200,isPET = FALSE,
  chr = c("chr1","chr2","chr3"))
regions(ctcf) <- ctcf_gr
ctcf
```

```
## Segvis for ctcf_peaks regions
## Paired-end Tags:  FALSE
## Fragment length: 200
## Max Bandwidth: 101
## Using reads file:
## ../inst/extdata/reads/encode_K562_Ctcf_first3chr_Rep1.sort.bam
## Using regions for 3 chromosomes
## GRanges object with 2000 ranges and 0 metadata columns:
##          seqnames               ranges strand
##             <Rle>            <IRanges>  <Rle>
##      [1]     chr1 [114889057, 114889538]      *
##      [2]     chr1 [225662556, 225663044]      *
##      [3]     chr1 [150951878, 150952345]      *
##      [4]     chr1 [ 17036198,  17036690]      *
```

```
##      [5]     chr1 [ 35318207,  35318710]      *
##      ...      ...                   ...    ...
##   [1996]     chr3 [171171060, 171171364]      *
##   [1997]     chr3 [150863855, 150864128]      *
##   [1998]     chr3 [196756655, 196756932]      *
##   [1999]     chr3 [118603610, 118603955]      *
##   [2000]     chr3 [  9768737,   9769156]      *
##   -------
##   seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

### 2.3 Creating `segvis_block` object

*Segvis* allows the use of several cores by using the parameter `mc`, which specifies the number of cores used by parallel processing. To create a `segvis_block` object it is necessary to follow a series of steps:

```
ctcf <- loadReads(ctcf, mc = 24)
ctcf <- matchReads(ctcf,mc = 24)
ctcf <- getCoverage(ctcf,mc = 24)
ctcf_block <- Segvis_block(ctcf,bw = 1,mc = 24)
```

To obtain the number of reads considered in an experiment:

```
ctcf_reads <- countReads(ctcf)
ctcf_reads
```

```
## [1] 6649370
```

In order to visually compare the enrichment level of more than one experiments it is necessary to normalize the experiment, which *Segvis* allows to do, by using:

```
normConst(ctcf_block) <- ctcf_reads
ctcf_block <- normalize(ctcf_block)
```

## 3 Some examples

### 3.1 Plotting different marks accross specific peaks

Using the same peaks as before. We are going to create a `segvis_block` for two additional marks, and plot all three marks the top 3 peaks, therefore we need to build two new `segvis_block` objects:

```
h3k27ac <- buildSegvis(name = "h3k27ac",
  file = "../inst/extdata/reads/encode_K562_H3k27ac_first3chr.sort.bam",
  maxBandwidth = 101,fragLen = 200,isPET = FALSE,
  chr = c("chr1","chr2","chr3"))
regions(h3k27ac) <- ctcf_gr

h3k27ac <- loadReads(h3k27ac, mc = 24)
h3k27ac <- matchReads(h3k27ac,mc = 24)
h3k27ac <- getCoverage(h3k27ac,mc = 24)
h3k27ac_block <- Segvis_block(h3k27ac,bw = 1,mc = 24)

h3k4me1 <- buildSegvis(name = "h3k4me1",
  file = "../inst/extdata/reads/encode_K562_H3k4me1_first3chr.sort.bam",
```

```
  maxBandwidth = 101,fragLen = 200,isPET = FALSE,
  chr = c("chr1","chr2","chr3"))
regions(h3k4me1) <- ctcf_gr

h3k4me1 <- loadReads(h3k4me1, mc = 24)
h3k4me1 <- matchReads(h3k4me1,mc = 24)
h3k4me1 <- getCoverage(h3k4me1,mc = 24)
h3k4me1_block <- Segvis_block(h3k4me1,bw = 1,mc = 24)
```

When sequencing data from different samples are considered together, they are often normalized to account for differences in the sequencing depths. The `normalize` function provides normalization functionalities. The default to to scale all the samples to 1M reads.

```
h3k27ac_reads <- countReads(h3k27ac)
normConst(h3k27ac_block) <- h3k27ac_reads
h3k27ac_block <- normalize(h3k27ac_block)

h3k4me1_reads <- countReads(h3k4me1)
normConst(h3k4me1_block) <- h3k4me1_reads
h3k4me1_block <- normalize(h3k4me1_block)

block_list <- Segvis_block_list(ctcf_block,h3k27ac_block,h3k4me1_block)
names(block_list) <- c("ctcf","h3k27ac","h3k4me1")
```

Then we can visualize all three marks over one regions by using the `plot_profiles` function:

```
rstart <- start(ctcf_gr)[1]
rend <- end(ctcf_gr)[1]
chr <- as.character(seqnames(ctcf_gr)[1])
iden <- function(x)x

p1 <- plot_profiles(block_list,condition = seqnames == chr & start == rstart,
  coord = rstart:rend,FUN = iden,mc=24)
```

The output of `plot_profiles` is a `ggplot` object, which we can modify it to obtains better looking plots.

```
p2 <- p1 + facet_grid(condition~.,scales = "free_y")
p3 <- p2 + scale_colour_brewer(palette = "Dark2")+theme(legend.position = "none")
```

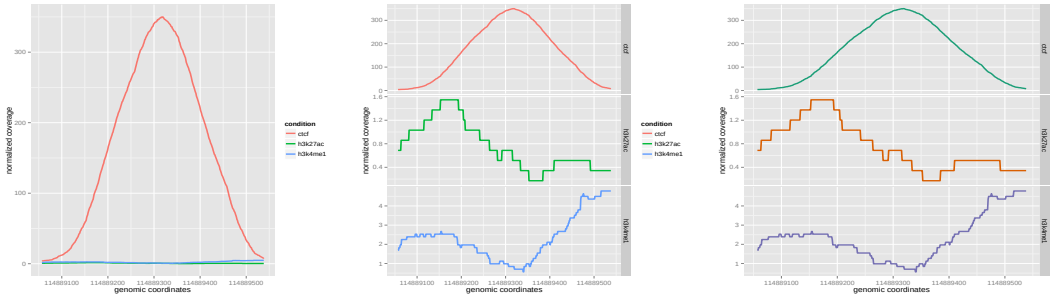


Figure 1: Initial use to plot the first Ctcf peak with all three marks (Ctcf, H3k27ac and H3k4me1). From left to right: p1, p2 and p3.

## 3.2 Finding the summit of a specific mark's peaks

One of the functions of *Segvis* is to calculate and plot statistics over a window around biologically meaningful coordinates like TSS, TFBS, etc. This coordinates are not always available, therefore *Segvis* allows to find the summits of peaks formed by a collection of genomic regions and fragment reads.

In subsection 2.3, we built the `ctcf` object which contains both regions and fragment reads, we can find the summit by using the `findSummit`, and then "add" it to the `segvis_block`'s regions:

```
  summits <- findSummit(ctcf,bw=1,mc=24)
  ctcf_block <- addColumn(ctcf_block,name="summit",col=summits)
  ctcf_block
```

```
## Segvis profile: ctcf_peaks
## Bandwidth: 1
## The profile matrix IS scaled
## GRanges object with 2000 ranges and 1 metadata column:
##          seqnames                ranges strand   |    summit
##             <Rle>             <IRanges>  <Rle>   | <numeric>
##      [1]     chr1 [114889057, 114889538]      *   | 114889318
##      [2]     chr1 [225662556, 225663044]      *   | 225662819
##      [3]     chr1 [150951878, 150952345]      *   | 150952129
##      [4]     chr1 [ 17036198,  17036690]      *   |  17036475
##      [5]     chr1 [ 35318207,  35318710]      *   |  35318418
##      ...      ...                   ...    ... ...       ...
##   [1996]     chr3 [171171060, 171171364]      *   | 171171197
##   [1997]     chr3 [150863855, 150864128]      *   | 150864001
##   [1998]     chr3 [196756655, 196756932]      *   | 196756778
##   [1999]     chr3 [118603610, 118603955]      *   | 118603783
##   [2000]     chr3 [  9768737,   9769156]      *   |   9768843
##   -------
##   seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

## 3.3 Summarizing the coverage for a set of same width regions

Another functionality of *Segvis* is to calculate and plot statistics(e.g. mean, median, variace, etc.) over a window around biologically meaningful coordinates (TSS, TFBS, etc.), for which we are going to consider the summits that we found as those coordinates:

```
  window_ext <- 500
  new_start <- summits - window_ext
  new_end <- summits + window_ext
  new_regions <- GRanges(seqnames = seqnames(ctcf_gr),
    ranges = IRanges(start = new_start,end = new_end),strand ="*")
  str(width(new_regions))
```

```
##  int [1:2000] 1001 1001 1001 1001 1001 1001 1001 1001 1001 1001 ...
```

```
  regions(ctcf) <- new_regions
  regions(h3k27ac) <- new_regions
  regions(h3k4me1) <- new_regions
  all_segvis <- list("ctcf"=ctcf,"h3k27ac"=h3k27ac,"h3k4me1"=h3k4me1)
```

We are going to create a new `segvis_block_list` based on the new regions, for which we can apply the same methods `loadReads`, `matchReads`, `getCoverage` and `Segvis_block` to build it. This time, we are going to use two functions to apply all methods together:

```
do_all <- function(segvis_obj,bw,mc)
{
  segvis_obj <- loadReads(segvis_obj,mc = mc)
  segvis_obj <- matchReads(segvis_obj,mc = mc)
  segvis_obj <- getCoverage(segvis_obj,mc = mc)
  out <- Segvis_block(segvis_obj,bw = bw , mc = mc)
  return(out)
}

all_segvis_blocks <- lapply(all_segvis,do_all,bw = 1, mc = 24)
nreads <- c(ctcf_reads,h3k27ac_reads,h3k4me1_reads)

assign_and_normalize <- function(segvis_bl_obj,nreads)
{
  normConst(segvis_bl_obj) <- nreads
  segvis_bl_obj <- normalize(segvis_bl_obj)
  return(segvis_bl_obj)
}

all_segvis_blocks <- mapply(assign_and_normalize,
  all_segvis_blocks,nreads,SIMPLIFY=FALSE)
all_segvis_blocks <- Segvis_block_list(all_segvis_blocks)
names(all_segvis_blocks) <- names(all_segvis)
```

In this case, we have again 2000 regions with the same width of $m = 1001$, i.e. we can think it as a matrix of $K \times m$ and to analize it we can apply the same function to all column vectors of this matrix:

```
q1 <- plot_profiles(all_segvis_blocks,FUN = mean,mc = 24,
  coord = -window_ext:window_ext)+xlab("distance to summit")+
  ylab("mean normalized counts")+
  scale_color_brewer(guide = guide_legend(title = "condition"),palette = "Dark2")+
  theme(legend.position = "top")+geom_vline(xintercept=0,linetype= 2)

q2 <- plot_profiles(all_segvis_blocks,FUN = median,mc = 24,
  coord = -window_ext:window_ext)+xlab("distance to summit")+
  ylab("median normalized counts")+
  scale_color_brewer(guide = guide_legend(title = "condition"),palette = "Dark2")+
  theme(legend.position = "top")+geom_vline(xintercept=0,linetype= 2)
```

We can even use functions created in the moment, the only need to take a vector argument and return a 1 - dimensional value.

```
varlog <- function(x)var(log(1 + x))

q3 <- plot_profiles(all_segvis_blocks,FUN = varlog,mc = 24,
  coord = -window_ext:window_ext)+xlab("distance to summit")+
  ylab("variance of log( 1 + normalized counts)")+
  scale_color_brewer(guide = guide_legend(title = "condition"),palette = "Dark2")+
  theme(legend.position = "top")+geom_vline(xintercept=0,linetype= 2)
```

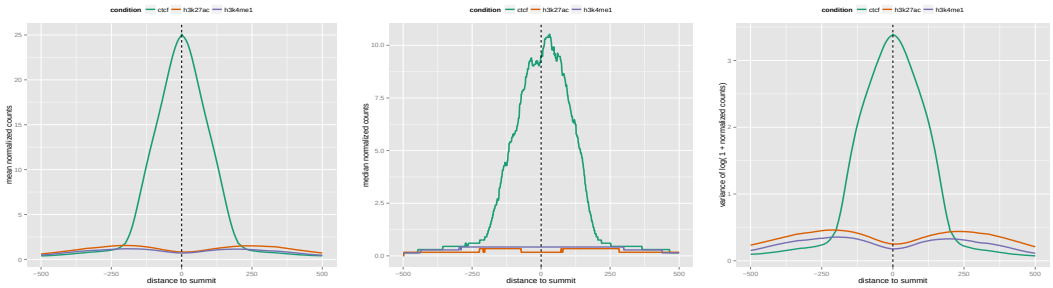


Figure 2: Mean, median and varlog profiles of all three marks (Ctcf,H3k27ac and H3k4me1) across the first 2000 Ctcf peaks

Another of *Segvis* functionalities is to calculate the data without generating the plot by the use of the `plot_data` function, which uses the same arguments as `plot_profiles`:

```
  # x is the genomic coordinates or distance to summit in thise case
  # y is the normalized counts
  # 1001 x 3 conditions = 3003 rows (in table below)
  new_data1 <- plot_data(all_segvis_blocks,FUN = mean,trim = .1,mc = 24,
    coord = -window_ext:window_ext)
  new_data2 <- plot_data(all_segvis_blocks,FUN = median,mc = 24,
    coord = -window_ext:window_ext)
  new_data1
```

```
##          x         y condition
##    1: -500 0.2004889      ctcf
##    2: -499 0.2003009      ctcf
##    3: -498 0.2017108      ctcf
##    4: -497 0.2031207      ctcf
##    5: -496 0.2045307      ctcf
##   ---
## 2999:  496 0.2799976   h3k4me1
## 3000:  497 0.2787677   h3k4me1
## 3001:  498 0.2769227   h3k4me1
## 3002:  499 0.2760441   h3k4me1
## 3003:  500 0.2755170   h3k4me1
```

And we can use this `data.table` into previously defined plot:

```
  p4 <- p3 %+% new_data1 + ylab("average normalized counts")
  p5 <- p3 %+% new_data2 + ylab("median normalized counts")
```

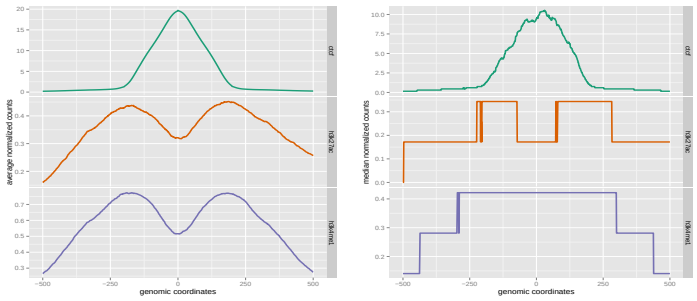


Figure 3: Mean and median coverage plot, by using a previously defined `ggplot` object

## 3.4 Subsetting and exploring the data with respect to user defined annotations

*Segvis* have methods to explore the data too. Lets consider as before that we want to explore the top 2000 Ctcf peaks, for which we already pre-processed a `segvis_block_list` made with Ctcf, H3k27ac and H3k4me1 fragment reads. Now, we want to explore this set of peaks respect to additional annotation information as to wheter this peaks overlap with Dnase accesible regions.

We read the Dnase hypersensitive sites from a file, and convert it to a `GRanges` object:

```
  dnase_file <- "../inst/extdata/peaks/encode_K562_dnase_openChrom_first3chr.narrowPeak"
  list.files("../inst/extdata/peaks/")
```

```
## [1] "encode_K562_Ctcf_peaks_first3chr.narrowPeak"
## [2] "encode_K562_dnase_openChrom_first3chr.narrowPeak"
## [3] "encode_K562_dnase_Uw_1_first3chr.narrowPeak"
## [4] "encode_K562_dnase_Uw_2_first3chr.narrowPeak"
## [5] "encode_K562_Pol2b_first3chr.narrowPeak"
```

```
  dnase_sites = read.table(dnase_file)
  dnase_gr <- GRanges(seqname = dnase_sites$V1,
    ranges = IRanges(start = dnase_sites$V2,end = dnase_sites$V3),
    strand = "*")
 dnase_gr
```

```
## GRanges object with 29343 ranges and 0 metadata columns:
##           seqnames                ranges strand
##              <Rle>             <IRanges>  <Rle>
##       [1]     chr1       [540885, 541038]      *
##       [2]     chr1       [713835, 714489]      *
##       [3]     chr1       [752734, 753083]      *
##       [4]     chr1       [762187, 763231]      *
##       [5]     chr1       [777812, 778569]      *
##       ...      ...                    ...    ...
##   [29339]     chr3 [197686834, 197687293]      *
##   [29340]     chr3 [197754035, 197754154]      *
##   [29341]     chr3 [197759379, 197759678]      *
##   [29342]     chr3 [197807020, 197808185]      *
##   [29343]     chr3 [197834936, 197835028]      *
##   -------
##   seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

We count the number of overlaps between the Dnase hypersensitive sites and the top 2000 Ctcf peaks and add those columns to the `segvis_block_list` we calculated in :

```
  nr_overlaps <- countOverlaps(regions(all_segvis_blocks[[1]]),dnase_gr)
  all_segvis_blocks <- lapply(all_segvis_blocks,
    addColumn,name = "dnase_overlaps",col =nr_overlaps)
  all_segvis_blocks[[1]]
```

```
## Segvis profile: ctcf_peaks
## Bandwidth: 1
## The profile matrix IS scaled
## GRanges object with 2000 ranges and 1 metadata column:
##          seqnames                ranges strand   | dnase_overlaps
##             <Rle>             <IRanges>  <Rle>   |      <integer>
##      [1]     chr1 [114888818, 114889818]      *   |              1
##      [2]     chr1 [225662319, 225663319]      *   |              1
##      [3]     chr1 [ 17035975,  17036975]      *   |              1
```

```
##      [4]     chr1 [204776035, 204777035]      *   |              1
##      [5]     chr1 [ 19239241,  19240241]      *   |              1
##      ...      ...                     ...    ... ...            ...
##   [1996]     chr3 [193498236, 193499236]      *   |              1
##   [1997]     chr3 [ 11100887,  11101887]      *   |              1
##   [1998]     chr3 [126470748, 126471748]      *   |              1
##   [1999]     chr3 [ 52177993,  52178993]      *   |              1
##   [2000]     chr3 [ 10024309,  10025309]      *   |              0
##   -------
##   seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

Then we can subset the data based on queries based on that annotation, for example:

```
  ctcf_subset <- subset(all_segvis_blocks[[1]],dnase_overlaps > 0)
  ctcf_subset
```

```
## Segvis profile: ctcf_peaks
## Bandwidth: 1
## The profile matrix IS scaled
## GRanges object with 1053 ranges and 1 metadata column:
##          seqnames               ranges strand   | dnase_overlaps
##             <Rle>            <IRanges>  <Rle>   |      <integer>
##      [1]     chr1 [114888818, 114889818]      *   |              1
##      [2]     chr1 [225662319, 225663319]      *   |              1
##      [3]     chr1 [ 17035975,  17036975]      *   |              1
##      [4]     chr1 [204776035, 204777035]      *   |              1
##      [5]     chr1 [ 19239241,  19240241]      *   |              1
##      ...      ...                     ...    ... ...            ...
##   [1049]     chr3 [ 38537250,  38538250]      *   |              1
##   [1050]     chr3 [193498236, 193499236]      *   |              1
##   [1051]     chr3 [ 11100887,  11101887]      *   |              1
##   [1052]     chr3 [126470748, 126471748]      *   |              1
##   [1053]     chr3 [ 52177993,  52178993]      *   |              1
##   -------
##   seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

```
  cover_table(ctcf_subset)
```

```
##           chr match     coord tagCounts
##       1: chr1     1 114888818 0.4511706
##       2: chr1     1 114888819 0.4511706
##       3: chr1     1 114888820 0.4511706
##       4: chr1     1 114888821 0.4511706
##       5: chr1     1 114888822 0.4511706
##      ---
## 1054049: chr3   343  52178989 0.1503902
## 1054050: chr3   343  52178990 0.1503902
## 1054051: chr3   343  52178991 0.1503902
## 1054052: chr3   343  52178992 0.1503902
## 1054053: chr3   343  52178993 0.1503902
```

Furthermore we can recreate the plots of figure 2 for the Ctcf peaks that overlap Dnse hypersensitive sites:

```
  s1 <- plot_profiles(all_segvis_blocks,FUN = mean,mc = 24,
    condition = dnase_overlaps > 0 ,
    coord = -window_ext:window_ext)+xlab("distance to summit")+
    ylab("mean normalized counts")+
```

```
  scale_color_brewer(guide = guide_legend(title = "condition"),palette = "Dark2")+
  theme(legend.position = "top")+geom_vline(xintercept=0,linetype= 2)

s2 <- plot_profiles(all_segvis_blocks,FUN = median,mc = 24,
  condition = dnase_overlaps > 0 ,
  coord = -window_ext:window_ext)+xlab("distance to summit")+
  ylab("median normalized counts")+
  scale_color_brewer(guide = guide_legend(title = "condition"),palette = "Dark2")+
  theme(legend.position = "top")+geom_vline(xintercept=0,linetype= 2)

s3 <- plot_profiles(all_segvis_blocks,FUN = varlog,mc = 24,
  condition = dnase_overlaps > 0 ,
  coord = -window_ext:window_ext)+xlab("distance to summit")+
  ylab("variance of log( 1 + normalized counts)")+
  scale_color_brewer(guide = guide_legend(title = "condition"),palette = "Dark2")+
  theme(legend.position = "top")+geom_vline(xintercept=0,linetype= 2)
```

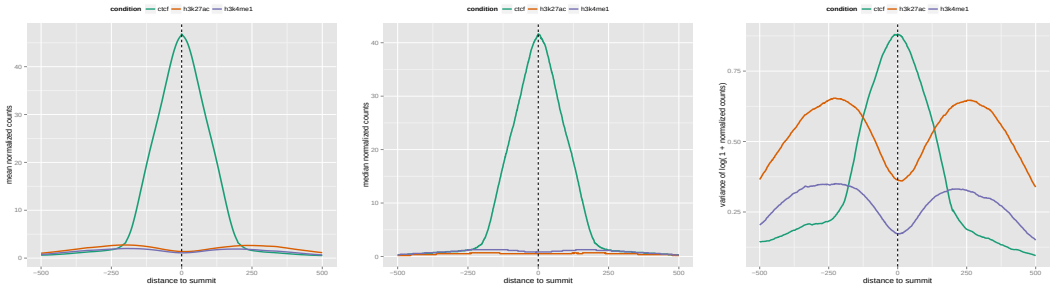


Figure 4: Mean, median and varlog profiles of all three marks (Ctcf,H3k27ac and H3k4me1) across the first 2000 Ctcf peaks that overlap with Dnse hypersentive sites

Furthermore, we can use the `plot_data` method to build more complicated figures. For example:

```
mean_overlap_data <- plot_data(all_segvis_blocks,FUN = mean,mc = 24,
  condition = dnase_overlaps > 0,
  coord = -window_ext:window_ext)
mean_comp_data <- plot_data(all_segvis_blocks,FUN = mean,mc = 24,
  condition = dnase_overlaps ==  0,
  coord = -window_ext:window_ext)
new_data <- rbind(mean_overlap_data[,overlap:="yes"],
  mean_comp_data[,overlap:="no"])
fancy_plot <- ggplot(new_data,aes(x,y,colour = condition))+geom_line(size=1.1)+
  facet_grid(overlap~.,scales = "free_y")+theme(legend.position = "top")+
  ggtitle("DHS overlaps")+geom_vline(xintercept = 0,linetype=2,size=1.1)+
  xlab("distance to summit")+ylab("average coverage")+
  scale_color_brewer(palette = "Set1")
```

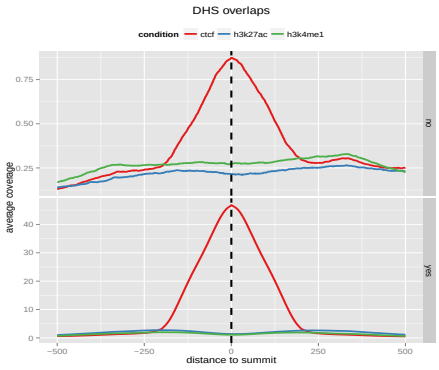


Figure 5: Average coverage of Ctcf, H3k27ac and H3k4me1 over Ctcf peaks overlapping with Dnase Hypersensitive Sites(DHS)

# 4 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, stats4, utils
- Other packages: BiocGenerics 0.12.1, Biostrings 2.34.1, data.table 1.9.4, GenomeInfoDb 1.2.5, GenomicAlignments 1.2.2, GenomicRanges 1.18.4, ggplot2 1.0.1, IRanges 2.0.1, rbamtools 2.10.0, Rsamtools 1.18.3, S4Vectors 0.4.0, Segvis 2.0, XVector 0.6.0
- Loaded via a namespace (and not attached): base64enc 0.1-2, BatchJobs 1.6, BBmisc 1.9, BiocParallel 1.0.3, BiocStyle 1.4.1, bitops 1.0-6, brew 1.0-6, checkmate 1.5.2, chron 2.3-47, codetools 0.2-11, colorspace 1.2-6, curl 0.9.1, DBI 0.3.1, devtools 1.8.0, digest 0.6.8, evaluate 0.7, fail 1.2, foreach 1.4.2, formatR 1.2, git2r 0.10.1, grid 3.1.1, gtable 0.1.2, highr 0.5, iterators 1.0.7, knitr 1.10.5, labeling 0.3, magrittr 1.5, MASS 7.3-39, memoise 0.2.1, munsell 0.4.2, plyr 1.8.2, proto 0.3-10, RColorBrewer 1.1-2, Rcpp 0.11.6, reshape2 1.4.1, roxygen2 4.1.1, RSQLite 1.0.0, rversions 1.0.1, scales 0.2.5, sendmailR 1.2-1, stringi 0.5-2, stringr 1.0.0, tools 3.1.1, xml2 0.1.1, zlibbioc 1.12.0