SigFuge (Tutorial)

Patrick K. Kimes, Christopher R. Cabanski

May 2, 2014

Contents

1 Summary

SigFuge is a tool that takes as input RNA-seq read depth (coverage) for multiple samples across a genomic locus (gene/transcript) and (1) clusters samples using the coverage data, (2) assesses significance of the clusters using SigClust and (3) visualizes transcript coverage along genomic coordinates. This document provides a tutorial on how to use the functions of the *SigFuge* package. **THERE HAVE BEEN SEVERAL CRITICAL UPDATES TO THE SigFuge PACKAGE. MAKE SURE THAT YOU HAVE DOWNLOADED AND ARE RUNNING SigFuge VERSION 1.1.2 OR GREATER TO ACHIEVE ALL FUNCTIONALITY DESCRIBED IN THIS TUTORIAL.**

2 Introduction

Frequently, one of the main goals of an RNA-seq experiment is to identify genes that are differentially expressed (DE) between two conditions, e.g. tumor and matched normal samples. In this setting where the contrasting groups of samples are known, several existing DE tools, including Bioconductor packages *DESeq* [1] and *edgeR* [2], can be used to identify the set of DE genes.

SigFuge is for expolartory analysis in RNA-seq experiments not designed to compare two pre-defined conditions. For example, several studies sequence multiple tumor samples with few, if any, matched normals. These studies are better suited for identifying subtypes with similar gene expression patterns using unsupervised clustering. Although many previous studies have performed unsupervised clustering using gene expression values across many genes, here we focus on clustering samples using per-base expression levels (coverage) across one gene locus at a time. This allows us to identify genes exhibiting different expression patterns across the gene transcript, such as alternative splicing or gene fusions, that may be missed when summarizing gene expression to one value. Our focus is on identifying genes that could each individually stratify the samples into different clinically important subgroups.

SigFuge is a tool for identifying genes with statistically significant clusters. For each gene, SigFuge treats each sample as a curve where RNA-seq read depth (coverage) is a function of genomic position. See Figure 1 for an example of multiple samples visualized as curves across the *CDKN2A* locus. SigFuge clusters samples based on the shape of these expression curves. The significance of clustering at each locus is quantified by a p-value calculated using *SigClust* [3]. The SigClust method formulates the question of statistical significance in clustering as a hypothesis test. The null hypothesis of SigClust is that the data are from a single Gaussian distribution. The significance of a given clustering is measured by the strength of clustering relative to an appropriate simulated null distribution. As the p-value calculation depends on simulation, results will vary slightly between implementations even for the same data matrix. SigFuge output for each locus includes a SigClust p-value and corresponding visualizations of expression clusters along the locus.

Visualization of RNA-seq transcript coverage is an important aspect of any study. The plotting functions that we present here, which can be used independently of the SigFuge clustering results, show the expression level at the per-base resolution across a gene. Treating these expression profiles as curves allows users to visualize the expression of multiple samples in the same plot.

This tutorial is split into two parts. The first part (Section 3) provides examples for loading the data in the appropriate format. The second part (Section 4) provides a fully worked case study including normalization, clustering, significance testing and plotting.

2.1 Citation

For a more in depth discussion of clustering per-base expression profiles, see our manuscript describing the SigFuge algorithm in further detail. Please try to cite the following article when you publish results obtained using the software.

> Kimes, P., Cabanski, C., Wilkerson, M., Johnson, A., Makowski, L., Maher, C., Liu, Y., Perou, C., Marron, J.S., Hayes, D.N. SigFuge: unsupervised exploration of transcriptional alterations in RNA-Seq data. *In preparation*.

3 Importing data

Two objects are needed to run SigFuge: (1) a *GRanges* object or similar *data.frame* containing information about the locus (gene) of interest and (2) a data matrix of coverage for each sample at each base (position) across the locus. The *GRanges* object is only used to plot the exon boundaries, so it may be omitted if only calcualting SigFuge p-values.

There are two ways to create a *GRanges* object. The first way is to manually input the start and end positions of exons, the chromosome, and the strand orientation. If a gene has mutliple isoforms with overlapping exons, it is advised to take the union of all isoform models in order to detect possible alternative splicing. Below is an example for manually creating a *GRanges* object containing annotation data for the *CDKN2A* locus.

```
> library(SigFuge)
> geneAnnot <- GRanges(seqnames=Rle("chr9", 5),
+             ranges=IRanges(start=c(21967751,21968574,21970901,21974403,21994138),
+                            end=c(21968241,21968770,21971207,21975132,21994490)),
+             strand=Rle(strand("-"), 5))
> geneAnnot
```

```
GRanges with 5 ranges and 0 metadata columns:
      seqnames              ranges strand
         <Rle>           <IRanges>  <Rle>
  [1]     chr9 [21967751, 21968241]      -
  [2]     chr9 [21968574, 21968770]      -
  [3]     chr9 [21970901, 21971207]      -
  [4]     chr9 [21974403, 21975132]      -
  [5]     chr9 [21994138, 21994490]      -
```

```
  ---
  seqlengths:
   chr9
     NA
```

The second way to create a *GRanges* object is to extract the annotation information from an appropriate *R* package database. Below shows how to extract gene annotation data from the set of UCSC known genes. Notice that exons 4 and 5 are overlapping, so we use the `reduce` function to merge these exons.

```
> library(org.Hs.eg.db)
> library(TxDb.Hsapiens.UCSC.hg19.knownGene)
> genesym <- c("CDKN2A")
> geneid <- select(org.Hs.eg.db, keys=genesym, keytype="SYMBOL", columns="ENTREZID")
> txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
> ex <- exonsBy(txdb, "gene")
> geneAnnot <- ex[[geneid$ENTREZID[1]]]
> geneAnnot
```

```
GRanges with 6 ranges and 2 metadata columns:
      seqnames              ranges strand |   exon_id   exon_name
         <Rle>           <IRanges>  <Rle> | <integer> <character>
  [1]     chr9 [21967751, 21968241]      - |    128318        <NA>
  [2]     chr9 [21968574, 21968770]      - |    128319        <NA>
  [3]     chr9 [21970901, 21971207]      - |    128320        <NA>
  [4]     chr9 [21974403, 21974826]      - |    128321        <NA>
  [5]     chr9 [21974677, 21975132]      - |    128322        <NA>
  [6]     chr9 [21994138, 21994490]      - |    128323        <NA>
  ---
  seqlengths:
                   chr1                 chr2 ...        chrUn_gl000249
              249250621            243199373 ...                 38502
```

```
> geneAnnot <- reduce(geneAnnot)
> geneAnnot
```

```
GRanges with 5 ranges and 0 metadata columns:
      seqnames              ranges strand
         <Rle>           <IRanges>  <Rle>
  [1]     chr9 [21967751, 21968241]      -
  [2]     chr9 [21968574, 21968770]      -
  [3]     chr9 [21970901, 21971207]      -
  [4]     chr9 [21974403, 21975132]      -
  [5]     chr9 [21994138, 21994490]      -
  ---
  seqlengths:
                   chr1                 chr2 ...        chrUn_gl000249
              249250621            243199373 ...                 38502
```

Now that we have the gene annotation, the next step is to construct the data matrix. For each locus, a d-by-n coverage matrix is required as input, where d is the locus length and n is the sample size. Each column corresponds to a sample and each row to the coverage (number of overlapping reads) at each base. We assume that the sequencing data has been aligned to a reference genome and the output is a sorted and indexed BAM file [4]. To demonstrate how to extract coverage data from a list of BAM files, we will use the example BAM files contained in the *prebsdata* package (see [5] for the original data source). First, we load, sort and index the BAM files for each sample. We also create a *GRanges* object corresponding to the exons of gene DPM1 located on chromosome 20.

```
> library(Rsamtools)
> library(prebsdata)
```

```
> geneAnnot <- GRanges(seqnames=Rle("20", 9),
+             ranges=IRanges(
+                 start=c(49551405,49552685,49557402,49558568,49562274,
+                     49562384,49565166,49571723,49574900),
+                 end=c(49551773,49552799,49557492,49558663,49562299,
+                     49562460,49565199,49571822,49575060)),
+             strand=Rle(strand("-"), 9))
> bam_file1 <- system.file(file.path("sample_bam_files", "input1.bam"),
+                          package="prebsdata")
> bam_file2 <- system.file(file.path("sample_bam_files", "input2.bam"),
+                          package="prebsdata")
> sorted1 <- sortBam(bam_file1, tempfile())
> indexBam(sorted1)
> sorted2 <- sortBam(bam_file2, tempfile())
> indexBam(sorted2)
> bam_files <- c(sorted1, sorted2)
```

Next, we use the *GRanges* object to generate a matrix of per-base coverage. To do this, we first create a function to extract coverage, `calcInfo`.

```
> calcInfo <- function(x) {
+     info <- apply(x[["seq"]], 2, function(y) {
+         y <- y[c("A", "C", "G", "T"), , drop=FALSE]
+         cvg <- colSums(y)
+     })
+     info
+ }
> param <- PileupParam(which=geneAnnot, what=c("seq","qual"),
+                      yieldBy="position", yieldAll=TRUE)
> fls <- PileupFiles(bam_files, param=param)
> res <- applyPileups(fls, calcInfo, param=param)
> geneDepth <- t(do.call(cbind,res))
> colnames(geneDepth) <- c("Sample1", "Sample2")
> geneDepth[500:505,]
```

```
     Sample1 Sample2
[1,]     110      75
[2,]      95      60
[3,]     105      60
[4,]      95      80
[5,]      95      50
[6,]      85      55
```

Depending on the number of samples, the above method may take several minutes per gene to create the coverage matrix.

It is important to note that the *GRanges* object is only used to define properties of the `SigFuge` figures and does not affect any of the analysis. Most notably, the locus information will not be used to parse the coverage matrix. As such, the matrix should only contain the exonic regions of interest and the matrix dimension d should equal the total length of the exons specified in the *GRanges* object.

4 `SigFuge` Case Study

This section gives a fully worked case study. We will analyze a subset of 179 lung squamous cell tumor samples sequenced as part of the Cancer Genome Atlas [6]. We will restrict attention to the *CDKN2A* locus. For this example, we skip the

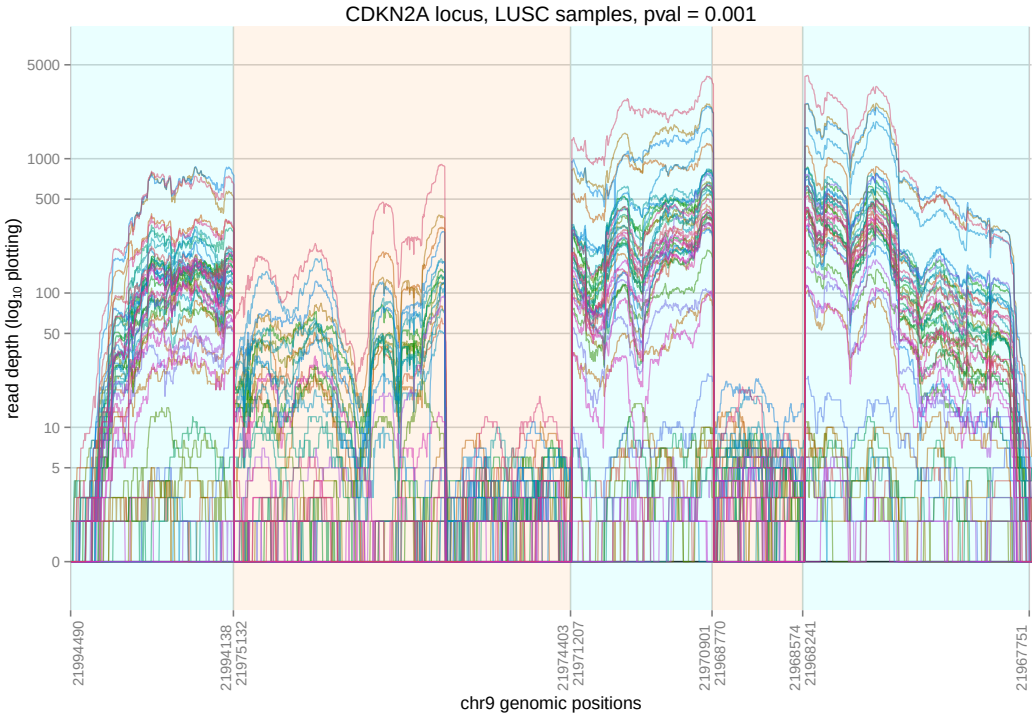

Figure 1: Graphical output from `SFfigure` for `lplots = 1`. Each curve represents the expression level of one sample across the *CDKN2A* locus. The curves are colored using default colors.

data retrieval step and apply `SigFuge` to a depth count matrix included with the package. `geneDepth` is a data matrix of coverage with rows corresponding to genomic positions and columns to samples, and `geneAnnot` is a *GRanges* object containing annotation information about the *CDKN2A* locus.

```
> data(geneAnnot)
> data(geneDepth)
```

4.1 SFfigure

`SFfigure` is a comprehensive function that performs 4 main tasks: (1) normalization of the data matrix, (2) unsupervised clustering on the normalized values, (3) p-value calculation for significance of clustering, and (4) plotting of the data curves. The following code uses the default normalization and clustering procedures to produce 3 plots (Figures 1-3). These plots will be saved in the working directory as 'CDKN2A_1.pdf', 'CDKN2A_2.pdf', and 'CDKN2A_3.pdf'. To speed up calculations, we will only consider a subset of 50 samples.

```
> genename <- "CDKN2A"
> mdata <- geneDepth[,101:150]
> SFfigure(data=mdata, locusname=genename,
+            annot=geneAnnot, lplots=1:3, savestr=genename,
+            titlestr="CDKN2A locus, LUSC samples")
```

Figure 1 shows the raw coverage curves across the *CDKN2A* locus, produced by setting `lplots = 1`. The x-axis shows genomic position, where different exons are colored using alternating colors (introns are not included). The y-axis shows coverage on the $\log_{10}$ scale. Each curve corresponds to a distinct sample (50 curves are overlaid, one for each sample). The `SigClust` p-value is reported in the title. There are only two clear clusters visible in Figure 1: a low expression class and a high expression class. However, the significant p-value indicates that after filtering out samples with low expression,

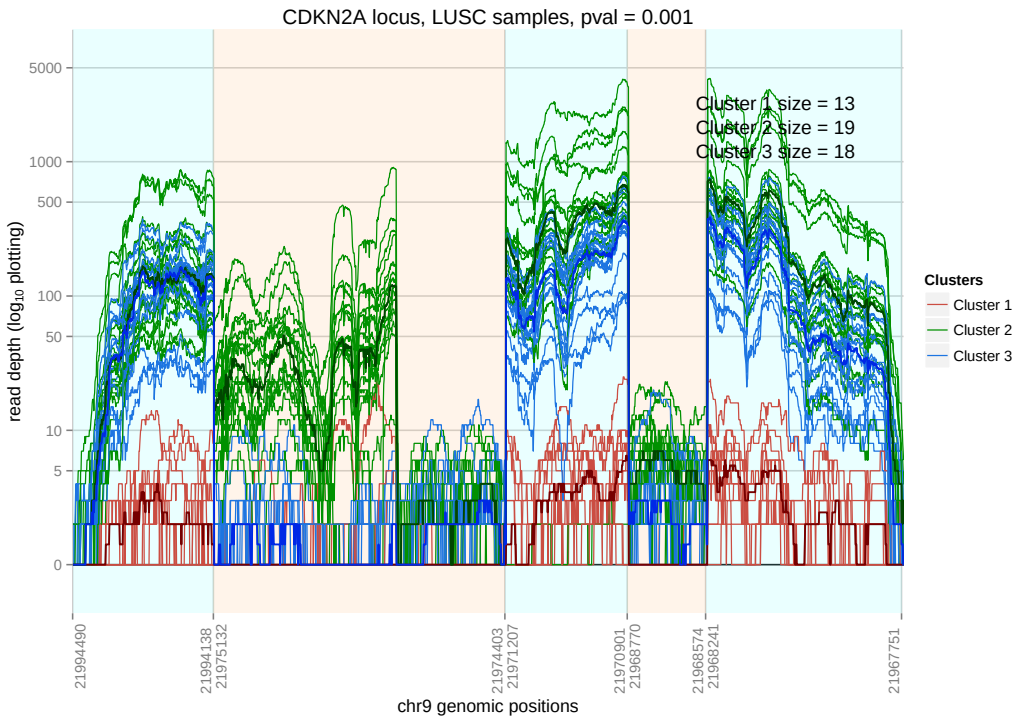

Figure 2: Graphical output from `SFfigure` for `lplots = 2`. The curves are colored according to class label. The red curves represent a low expression class. There is a large difference in expression of the second exon between the blue and green curves. This difference is reflected in the highly significant p-value.

the high expression samples can be further split into two distinct clusters. Although these clusters are difficult to see in Figure 1, they are easily observed in Figures 2 and 3.

Figure 2 shows the same curves as Figure 1, but the curves are now colored according to cluster labels. The red curves represent a class of lowly expressed samples. Because these lowly expressed curves may represent an important biological class (e.g. copy number deleted samples) independent of another transcript alteration such as alternative splicing, they are filtered out when normalizing, clustering, and assessing significance of the clusters. The blue and green curves represent the two clusters as determined by 2-means clustering. The number of curves/samples in each class is also reported on the plot. The thick colored lines represent the median read depth for each class. Although the green and blue classes have similar expression profiles in four of the five exons, there is a clear difference in expression at the second exon. This difference is reflected in the highly significant p-value.

Instead of overlaying all curves in a single panel, Figure 3 plots each class of curves separately. The scale of the y-axis is constant between all plots to allow for easier comparison between classes.

Although not present in this example, occasionally there will be an outlier sample that is of interest. Using `SFfigure` with the option `lplots = 4` produces a series of plots, where each plot shows one individual curve colored by its class label. The column names of the data matrix are assumed to be the sample IDs and reported at the top of each plot. This makes it easy to identify the outlier sample(s).

```
> genename <- "CDKN2A"
> SFfigure(data=mdata, locusname=genename,
+            annot=geneAnnot, lplots=4, savestr ="CDKN2A_individual_curves",
+            titlestr="CDKN2A locus, LUSC samples")
```

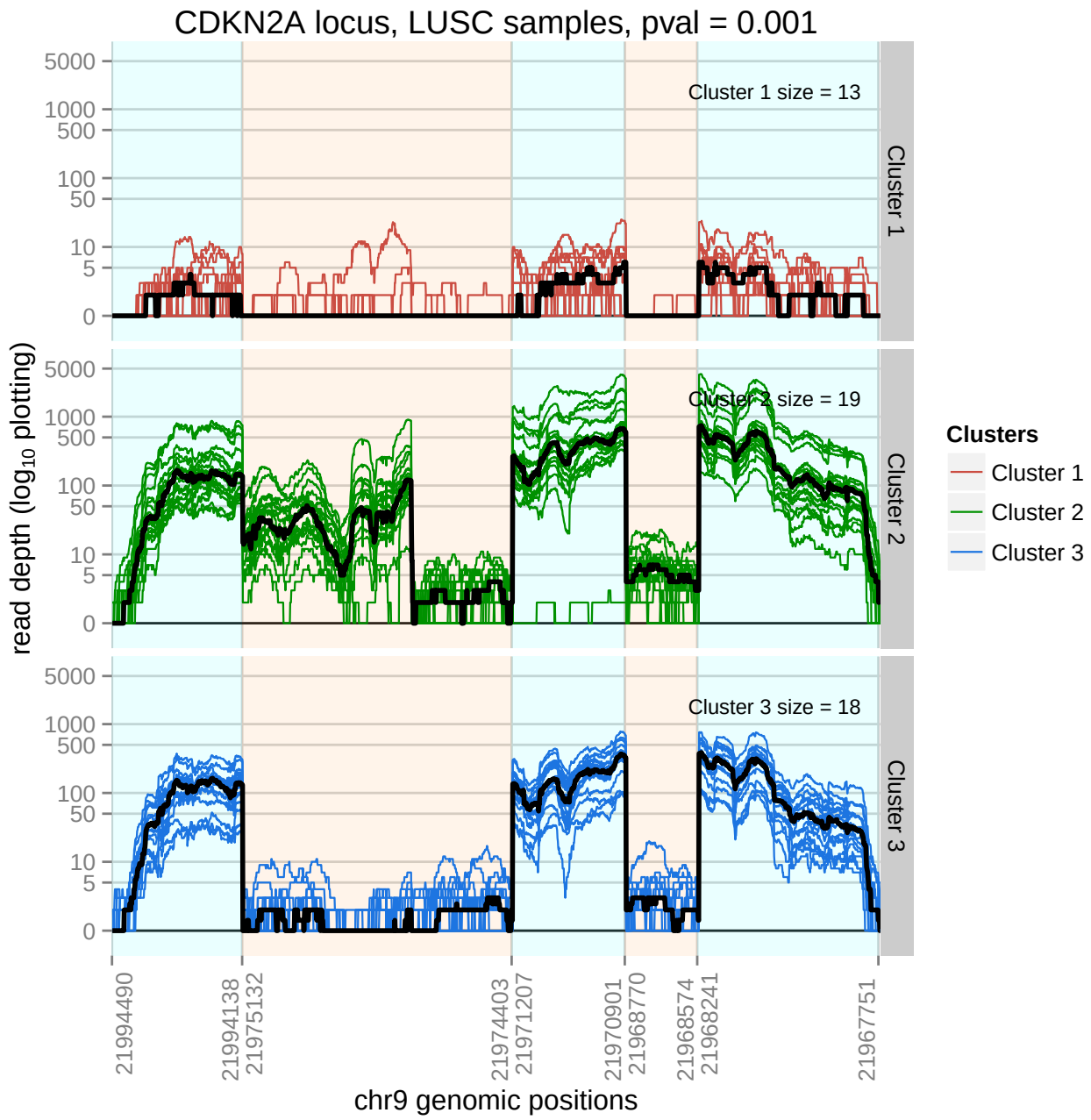

Figure 3: Graphical output from `SFfigure` for `lplots = 3`.

4.2 SFpval

`SFpval` allows you to obatin the p-value without creating any plots.

```
> output <- SFpval(mdata)
> output@pvalnorm
```

```
[1] 0.001853944
```

This reported p-value may not perfectly match the p-value reported in the plots because the p-value calculation depends on simulation. This function is useful if you are interested in testing hundreds or thousands of loci in the genome. Once you have obtained all of the p-values, you can create plots for only the most significant genes. Since `SFpval` does not correct for multiple comparisons, it is important to adjust the p-values using `p.adjust` if iterating over multiple loci.

4.3 `SFlabels` and `SFnormalize`

Cluster labels can be computed with `SFlabels`. First, we have to normalize the data using `SFnormalize`. Because we are interested in alterations such as splicing and fusions instead of differential gene expression, the `SigFuge` normalization procedure adjusts for differences in total expression (read counts). The normalization procedure first filters out lowly expressed samples because these samples are unable to provide any evidence of an alteration.

```
> norm <- SFnormalize(mdata)
> labels <- SFlabels(norm)
> labels[1:10]
```

```
 [1] 1 2 2 2 3 1 2 1 3 1
```

The `SigFuge` normalization procedure reduces the variability between samples and exaggerates any difference not explained by a simple difference in overall expression. See the `SigFuge` manuscript for more details about the normalization procedure employed here. Plots showing the read depth before and after normalization can be produced using the following code. Note, we first remove low-expression samples since they are not included in the normalization procedure.

```
> exp.data <- mdata[,-which(norm$flag == 1)]
> labels <- labels[-which(norm$flag == 1)]
> SFfigure(exp.data, locusname=genename, annot=geneAnnot,
+            data.labels=labels, flag=0, lplots=2,
+            savestr="Raw_CDKN2A", titlestr="Raw coverage",
+            pval=0)
> SFfigure(norm$data.norm, locusname=genename, annot=geneAnnot,
+            data.labels=labels, flag=0, lplots=2,
+            savestr="Norm_CDKN2A", titlestr="Normalized coverage",
+            pval=0)
```

References

[1] Anders, S. and Huber, W. (2010). Differential expression analysis for sequence count data. Genome Biology 11:R106.

[2] Robinson M.D., McCarthy D.J. and Smyth GK (2010). edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics 26, 139−140.

[3] Liu, Y., Hayes, D.N., Nobel, A., Marron, J.S. (2008) Statistical Significance of Clustering for High-Dimension, Low-Sample Size Data. Journal of the American Statistical Association 103(408), 1281−1293.

[4] Li, H., Handsaker, B., Wysoker, A. (2009) The Sequence alignment/map (SAM) format and SAMtools. Bioinformatics 25, 2078−2079.

[5] Marioni, J.C., Mason, C.E., Mane, S.M., Stephens, M., Gilad, Y. (2008) RNA-seq: an assessment of technical reproducibility and comparison with gene expression arrays. Genome Research 18(9), 1509−1517.

[6] The TCGA Research Network (2012). Comprehensive genomic characterization of squamous cell lung cancers. Nature 489(7417), 519−525.