Performing Some Basic Quality Checking and Analysis on Sequencing Data with `Repitools`

Mark Robinson Aaron Statham Dario Strbenac

March 29, 2011

1 Introduction

`Repitools` is a package that allows statistics of differential epigenetic marking to be calculated, as well as summaries of genome - wide trends to be visualised in a variety of formats. Some basic quality checking utilities are also available for sequencing data. The utility of `Repitools` comes from that most of the functionality available is implemented for both microarrays and next generation sequencing, with very similar function calls for both types of data.

In this vignette, quality checking of the sequencing data, followed by analysis and visualisation will be demonstrated. A more detailed description of the package can be found in the associated Bioinformatics Applications Note [1]

To start with, load the `Repitools` package.

```
> library(Repitools)
```

2 Data

A `GRangesList` of mapped short reads from an Illumina Genome Analyser of four samples is included with the package. `GRanges` objects of mapped files from many popular aligners can be created by first reading them into **R** with the `readAligned` function in the `ShortRead` package, then coerced with `as(alnRdObj, "GRanges")`. Only reads on chromosome 21 were kept, to have fast - running examples. The details of the samples are :

```
> dataPath <- system.file("data", package = "Repitools")
> load(paste(dataPath, "samplesList.Rdata", sep = .Platform$file.sep))
> names(samplesList)
```

```
[1] "PC_input"   "PC_MBDIP"   "Norm_input" "Norm_MBDIP"
```

Also, an annotation of genes located on chromosome 21 is included. The annotation is based on one provided from Affymetrix with their expression arrays [2].

[1]Repitools: an R package for the analysis of enrichment-based epigenomic data

[2]http://www.affymetrix.com/Auth/analysis/downloads/na27/wtgene/HuGene-1_0-st-v1.na27.hg18.transcript.csv.zip

```
> geneAnno <- read.csv(paste(dataPath, "chr21genes.csv", sep = .Platform$file.sep),
+     stringsAsFactors = FALSE)
> head(geneAnno)

    chr strand    start      end   symbol            name
1 chr21      + 33837227 33871651  C1orf92        AK127947
2 chr21      - 39636487 39642849 C15orf32 ENST00000380748
3 chr21      + 13341774 13341881      --- ENST00000384369
4 chr21      + 13915768 13915875      --- ENST00000384745
5 chr21      + 14295118 14295292      --- ENST00000330957
6 chr21      + 14365062 14365178      --- ENST00000364942
```

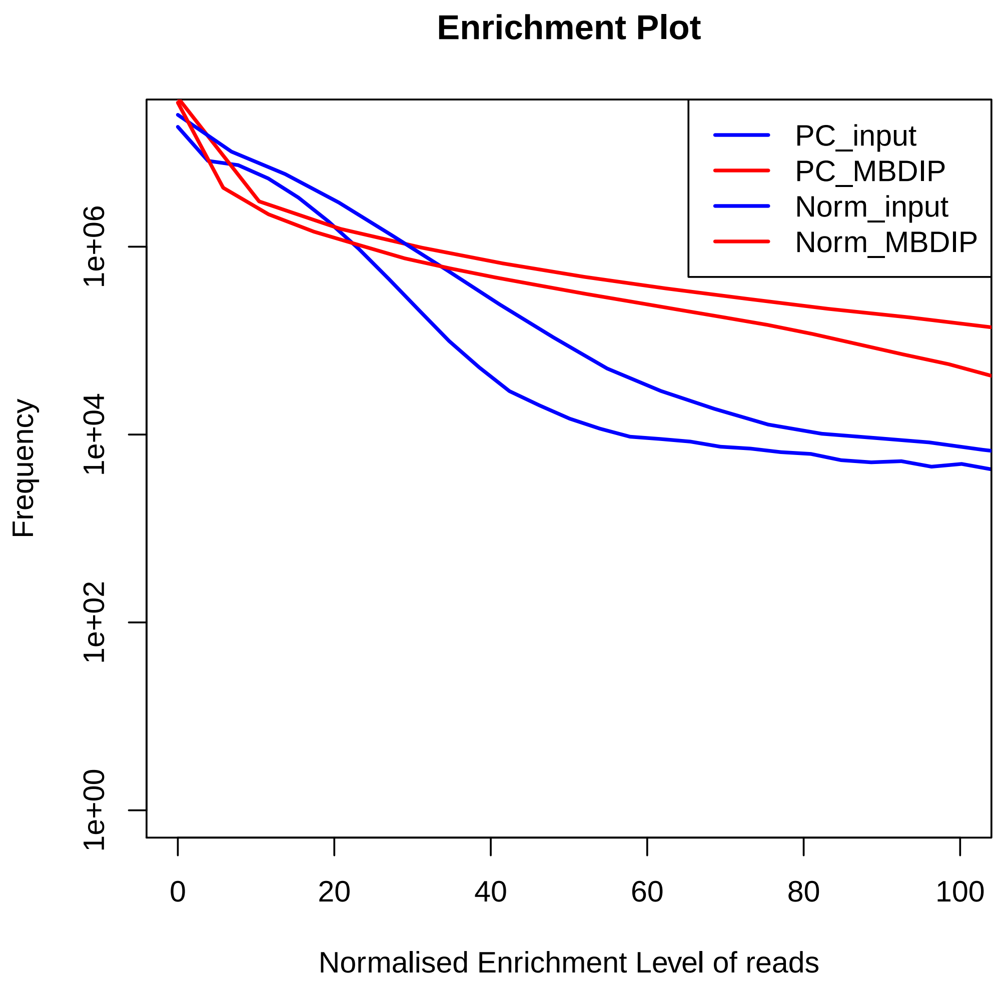
Lastly, there is matrix of gene expression difference data, with each element related to the corresponding row in the gene annotation table. These values are the t-statistics of background corrected and RMA normalised Affymetrix expression array experiments. The expression differences matrix will be used when illustrating some of the visualisation functionality later in the vignette.

```
> load(paste(dataPath, "expr.Rdata", sep = .Platform$file.sep))
> head(expr)

                t-stat
AK127947          0.08
ENST00000380748   1.31
ENST00000384369   0.11
ENST00000384745   0.24
ENST00000330957  -1.88
ENST00000364942  -0.56
```

3 Quality Checking

Notice that two of the samples are MBD2 IPs, and two are inputs. Therefore, the IP samples should differ to the inputs in two ways. Firstly, they should be more CpG rich, since DNA methylation rarely ever occurs outside of this sequence context. Also, since DNA methylation tends to occur in peaks, rather than spread out regions, a higher frequency of bases should have high coverage of reads in the IP samples than in input samples. The `enrichmentPlot` and `cpgDensityPlot` functions allow examination of this.

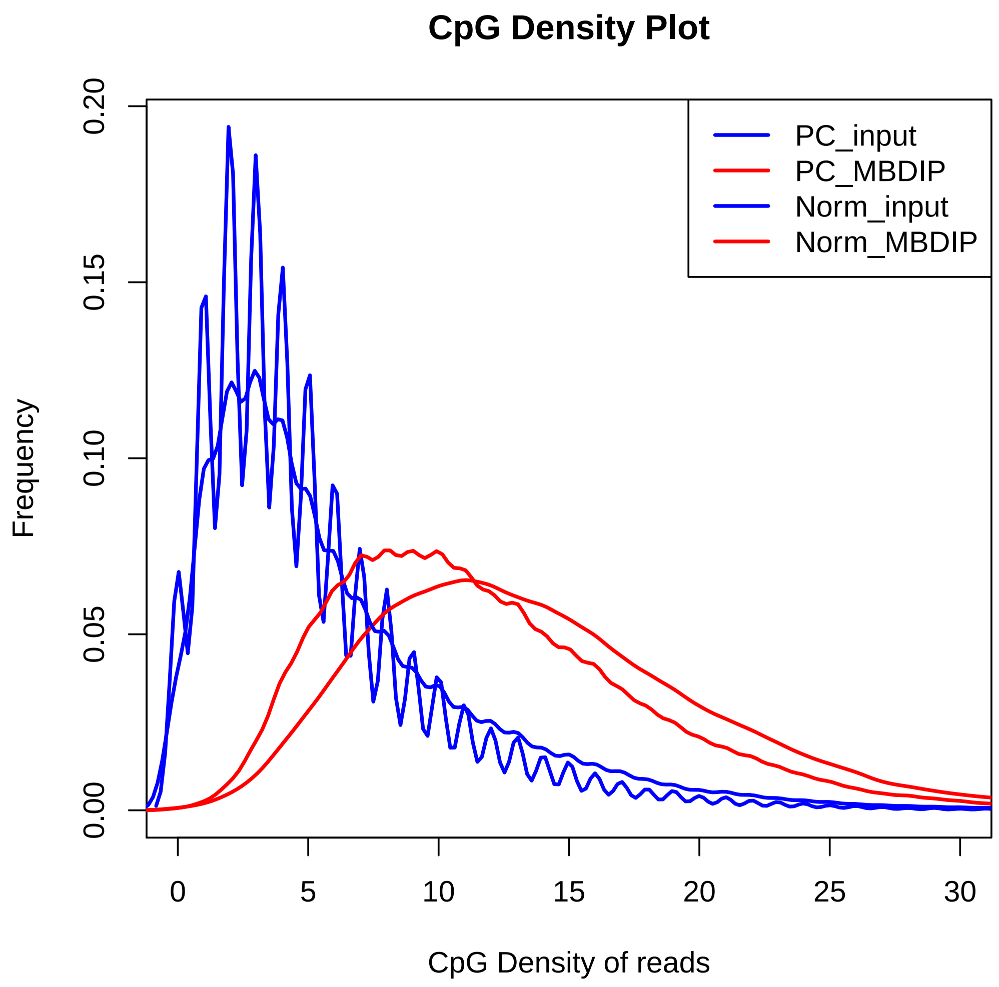
```
> library(BSgenome.Hsapiens.UCSC.hg18)
> enrichmentPlot(samplesList, Hsapiens, 300, cols = c("blue", "red",
+     "blue", "red"), xlim = c(0, 100), lwd = 2)

PC_input; PC_MBDIP; Norm_input; Norm_MBDIP;
```



The above code uses the `Hsapiens` object to get the maximum base of chromosomes. The normalisation of the coverage used is to scale every coverage value by 10 $million/number\_of\_reads\_in\_sample$. 300 is passed in as the `seqLen` parameter, because that is approximately the real length of the fragments sequenced. As expected, many more bases in the IP samples have high read coverages.

Next, the CpG density of reads is examined.

```
> cpgDensityPlot(samplesList, cols = c("blue", "red", "blue", "red"),
+     xlim = c(0, 30), wFunction = "none", organism = Hsapiens,
+     seqLen = 300, lwd = 2)

PC_input; PC_MBDIP; Norm_input; Norm_MBDIP;
```



This time the `Hsapiens` annotation is required so that the 300 base DNA sequence (tags are only 36 bp long) may be fetched. The `wFunction` parameter allows the count of CpGs to be weighted. In this example, raw counts are used.

Notice that at lower CpG densities, the two input samples have a higher frequency of reads than the two IP samples. At higher CpG densities, this trend is reversed. This suggests that the DNA methylation IP has worked.

For more general sequencing quality checking, the FastQC [3] Java application has been gaining in popularity due to its speed and variety of results. A container class, `FastQC`, accessors, and the method `readFastQC` for reading in the raw Java program text file output and creating a FastQC **R** object, have been made to provide a framework for the fast development of quality control report generating pipelines. Higher level container classes with accessor methods are `SequenceQC`, which groups `FastQC` objects for the aligned and unaligned reads of a single sample together, and `SequenceQCSet` which is a collection of `SequenceQC` objects, perhaps of multiple sequencing samples within the same experimental run.

[3]http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/

4 Analyses and Visualisations

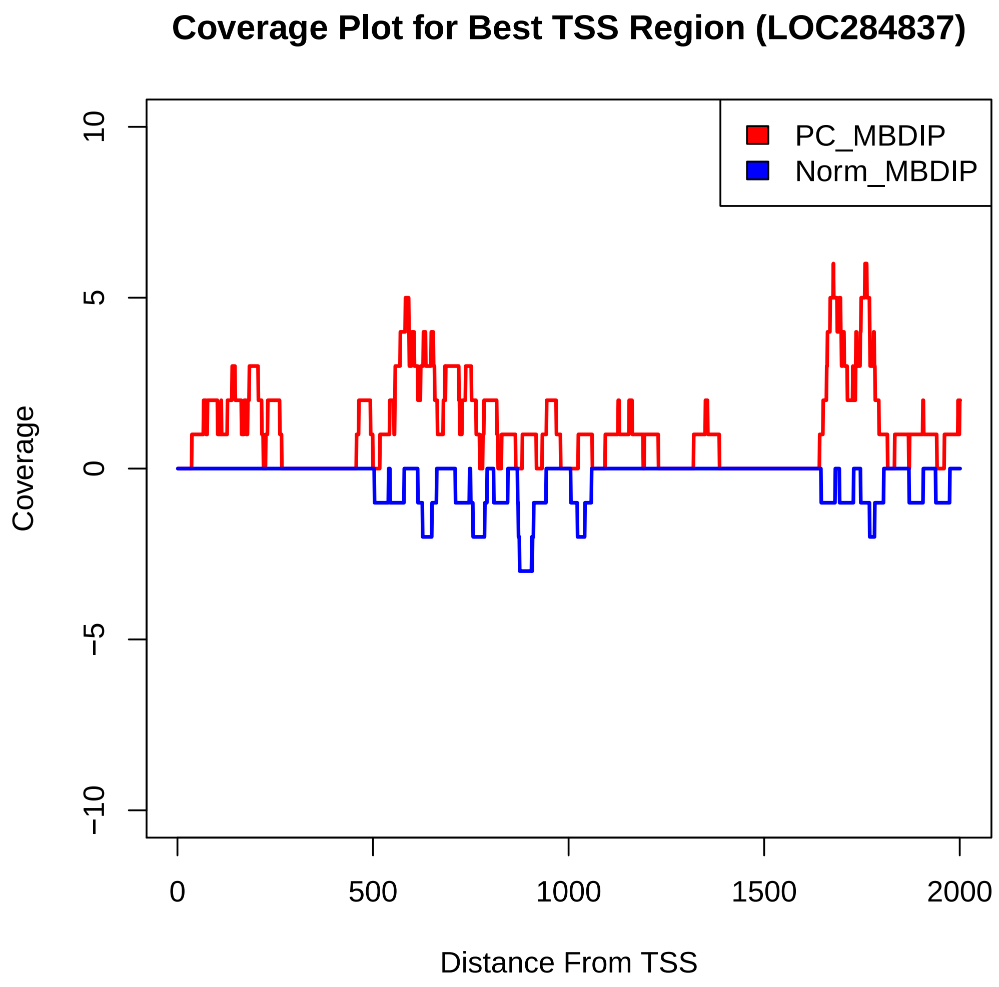
The blocksStats function is a convenient way to do a statistical test of differential enrichment between two groups or treatments, for counts in windows surrounding some genomic landmarks, like TSSs. The function leverages the package edgeR's modelling of counts as negative binomial distributed and its adaptation of Fisher's exact test to overdispersed data.

```
> designMatrix <- matrix(c(0, 1, 0, -1), dimnames = list(names(samplesList),
+     "Cancer - Normal Methylation"))
> designMatrix

           Cancer - Normal Methylation
PC_input                             0
PC_MBDIP                             1
Norm_input                           0
Norm_MBDIP                          -1

> stats <- blocksStats(samplesList, coordinatesTable = geneAnno,
+     design = designMatrix, upStream = 2000, downStream = 0, seqLen = 300)

Generating table of counts
Counting in PC_input
Counting in PC_MBDIP
Counting in Norm_input
Counting in Norm_MBDIP
Counting successful.
Processing column 1 of design matrix
Comparison of groups:  1 - -1

> stats <- stats[order(stats$`FDR_Cancer - Normal Methylation`),
+     ]
> head(stats)

         name   chr strand    start      end    symbol PC_input PC_MBDIP
52   BC111558 chr21      - 44054086 44056875 LOC284837       11       73
26   AK123727 chr21      + 43607639 43610874  FLJ41733       14       78
201 NM_004540 chr21      + 21292503 21833085     NCAM2       14       44
220 NM_006988 chr21      - 27130476 27139599   ADAMTS1       17       44
45   BC031097 chr21      - 45093942 45118023   PTTG1IP       13       26
211 NM_005806 chr21      + 33320108 33323370     OLIG2       22       34
    Norm_input Norm_MBDIP PC_MBDIP_pseudo Norm_MBDIP_pseudo
52          13         21        97.74692         15.664843
26          18         29       104.44012         21.641086
201         12          8        58.92638          5.953344
220         15         12        58.92638          8.941529
45          12          2        34.83090          1.470797
211          3          6        45.54000          4.459219
    logConc_Cancer - Normal Methylation logFC_Cancer - Normal Methylation
52                            -11.68433                          2.639043
```

```
26                                 -11.40371                          2.268957
201                                -12.74569                          3.300968
220                                -12.45320                          2.716005
45                                 -14.12518                          4.541976
211                                -13.13919                          3.344036
    PValue_Cancer - Normal Methylation FDR_Cancer - Normal Methylation
52                        6.125403e-16                    1.892749e-13
26                        2.792528e-14                    4.314456e-12
201                       3.169950e-12                    3.265049e-10
220                       4.322107e-10                    2.877787e-08
45                        4.656613e-10                    2.877787e-08
211                       8.654979e-10                    4.457314e-08
```

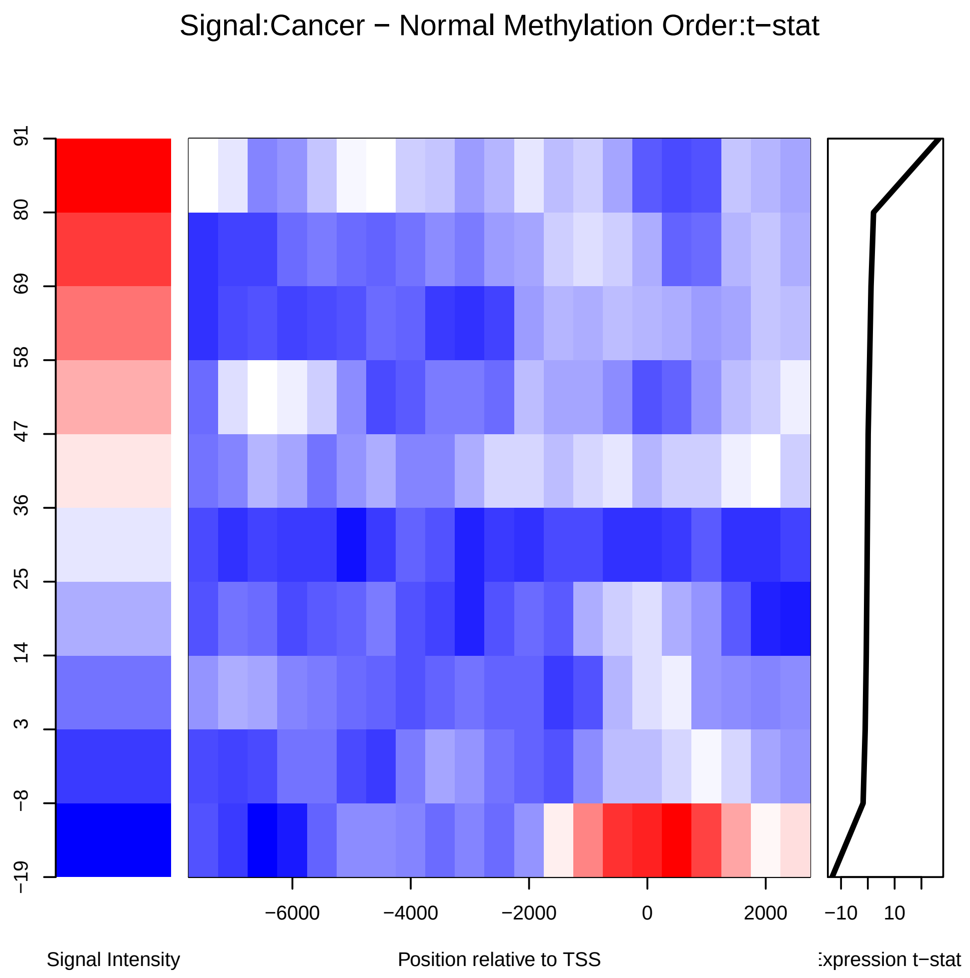
The example calculates statistics on TSS regions which start 2000 bases upstream of the TSS and finish at the TSS, after the reads have been extended to being 300 bases long. A coverage plot is shown to illustrate the best found region.

```
> cvgs <- lapply(samplesList[c(2, 4)], function(sampleRanges) coverage(sampleRanges)[[1]]
+     2000), drop = TRUE])
> plot(cvgs[[1]], type = "l", ylim = c(-10, 10), main = "Coverage Plot for Best TSS Regio
+     xlab = "Distance From TSS", ylab = "Coverage", col = "red",
+     lwd = 2)
> lines(-cvgs[[2]], col = "blue", lwd = 2)
> legend("topright", names(samplesList)[c(2, 4)], fill = c("red",
+     "blue"))
```



Epigenomic data is often gathered with other data, such as gene expression. It may be of interest to see the profile of epigenetic mark enrichment at a variety of distances from TSSs, and stratify this into groups by the expression of genes. The `binPlots` function is a convenient way to look at these interactions.

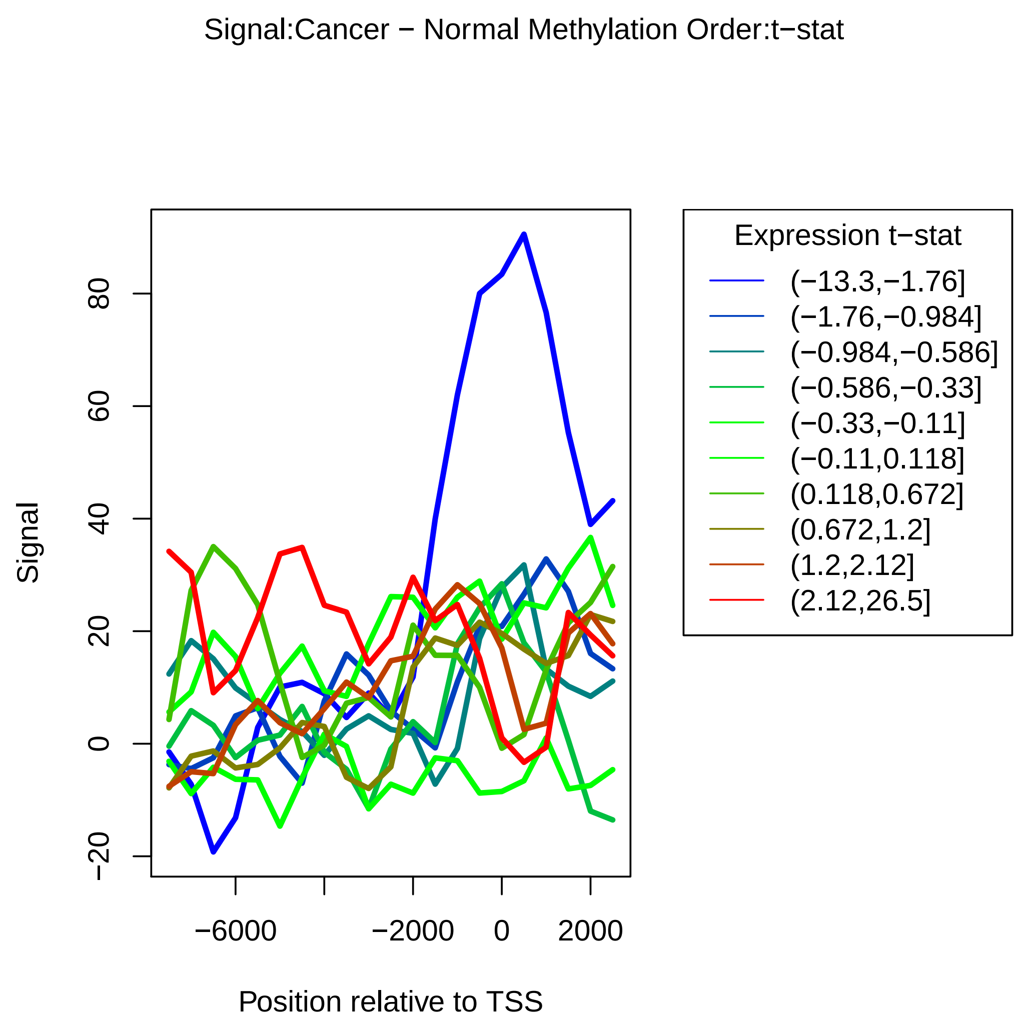
```
> binPlots(samplesList, geneAnno, design = designMatrix, by = 500,
+     bw = 500, seqLen = 300, ordering = expr, ordLabel = "Expression t-stat",
+     plotType = "heatmap", nbins = 10)
```



This example made counts in 500 base non - overlapping windows between -7500 bases upstream and 2500 bases downstream (the default range) for each gene, then split them into categories based on the expression difference value, and averaged over all counts for each particular window and expression category. It is encouraging to see that the lowest level of expression has a rather fine enrichment of DNA methylation about 2000 bases either side of the TSS. Apart from the heatmap visualisation, there are a number of other styles. Details can be found in the documentation of the function. More bins can be used for a nicer-looking heatmap when the full set of genes (genome-wide) is used.

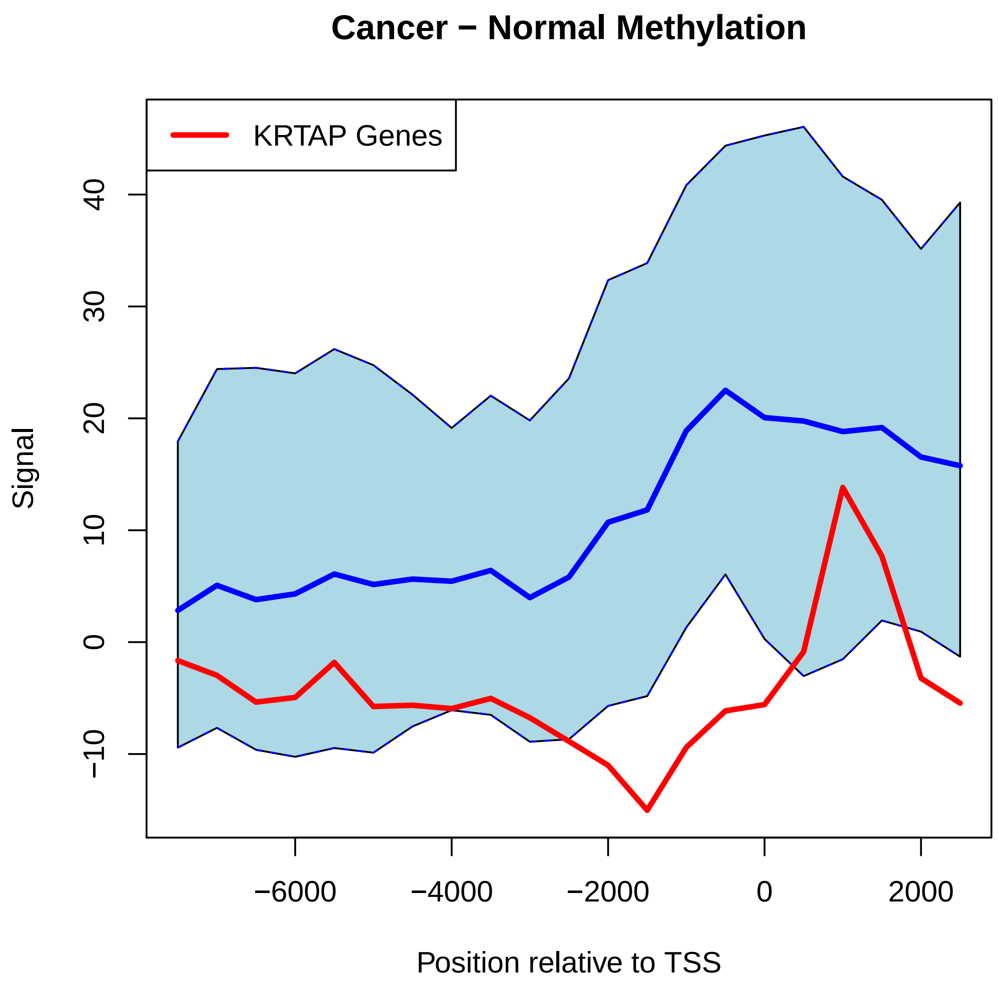
To demonstrate how similar it is to generate another style of binned plot, the next example shows the same data as a line-plot. Note that the function call is the same, apart from the `plotType` parameter. Notice the spike in DNA methylation for the set of lowest expressed genes, which form the blue line.

```
> binPlots(samplesList, geneAnno, design = designMatrix, by = 500,
+     bw = 500, seqLen = 300, ordering = expr, ordLabel = "Expression t-stat",
+     plotType = "line", nbins = 10)
```



Some genes may be of interest to the researcher for some reason. This subset of genes may be known to be strongly marked with another epigenetic mark, or change in expression in the same direction strongly, or many other reasons. No matter what the reason for selecting the subset is, the profile of intensities or counts can be plotted versus the profile of randomly selected gene lists and compared with the `significancePlots` function. In the following example, it will be checked whether the DNA methylation profile of keratin-associated protein coding genes is any different to that of the other genes.

```
> whichKRTAP <- grep("KRTAP", geneAnno$symbol)
> significancePlots(samplesList, geneAnno, geneList = list(`KRTAP Genes` = whichKRTAP),
+     design = designMatrix, by = 500, bw = 500, seqLen = 300)
```



The blue region forms the null distribution that was created by sampling random gene lists of the same size as the user - specified gene list a number of times, as set by the `nSamples` parameter. By default, the null region is a between the 0.025 and 0.975 quantiles of the null distribution. In this example, it appears that the KRTAP gene set has a significant loss of methylation around from the gene TSS to around 2000 bases upstream, in comparison to random sets of other genes.

Another analysis of interest for the epigenomics research community is to find regions of the genome where epigenetics marks or changes in such marks occur in consecutive genes on a particular chromosome. The function `findClusters` addresses this need. The method of determining clusters is to look through the column of statistics for a set of consecutive scores in the same direction. Which potential clusters are significant is determined by randomising the ordering of the statistics column a number of times, and counting the number of clusters found in the real statistics column and the randomised statistics columns, from a loose cutoff to a tight cutoff, and choosing the cutoff to be the first cutoff that meets or is below the user - specified FDR. It will not be demonstrated, as this randomisation procedure is time consuming.

5 Utility Functions

These functions perform some task that is commonly made with the data, but is not a formal analysis. Often, it is required to create a set of windows covering the entire genome, for some analysis. The function `genomeBlocks` dies this.

```
> genomeBlocks(Hsapiens, "chr21", 5000)
```

```
GRanges with 9389 ranges and 0 elementMetadata values
       seqnames               ranges strand   |
          <Rle>            <IRanges>  <Rle>   |
   [1]    chr21       [    1,  5000]      *   |
   [2]    chr21       [ 5001, 10000]      *   |
   [3]    chr21       [10001, 15000]      *   |
   [4]    chr21       [15001, 20000]      *   |
   [5]    chr21       [20001, 25000]      *   |
   [6]    chr21       [25001, 30000]      *   |
   [7]    chr21       [30001, 35000]      *   |
   [8]    chr21       [35001, 40000]      *   |
   [9]    chr21       [40001, 45000]      *   |
   ...      ...                  ...    ... ...
[9381]    chr21 [46900001, 46905000]      *   |
[9382]    chr21 [46905001, 46910000]      *   |
[9383]    chr21 [46910001, 46915000]      *   |
[9384]    chr21 [46915001, 46920000]      *   |
[9385]    chr21 [46920001, 46925000]      *   |
[9386]    chr21 [46925001, 46930000]      *   |
[9387]    chr21 [46930001, 46935000]      *   |
[9388]    chr21 [46935001, 46940000]      *   |
[9389]    chr21 [46940001, 46945000]      *   |

seqlengths
 chr21
    NA
```

This example makes a `GRanges` object of 5 kb windows along chromosome 21.

The researcher might have a set of locations that they want to know the CpG density of.

```
> cpgDensityCalc(head(geneAnno), window = 100, organism = Hsapiens)
```

```
; [1]  9 10  2  2  1  7
```

This example calculates the CpG density of a window 100 bases either side of the TSS for the first six genes on chromosome 21. By default, the CpG density is a linearly weighted count, such that a CpG at the TSS counts as one, and progressively less, towards zero, as the CpGs occur toward the edges of the region.

`annotationCounts` is useful to understand the counts of reads surrounding some landmarks, like TSSs. `annotationBlocksCounts` is the analogous function for counting in user-specified regions of the genome.

```
> annotationCounts(samplesList, head(geneAnno), 2000, 500, 300)

Counting in PC_input
Counting in PC_MBDIP
Counting in Norm_input
Counting in Norm_MBDIP
Counting successful.
                PC_input PC_MBDIP Norm_input Norm_MBDIP
AK127947              13        3         16          9
ENST00000380748        7        4         15         19
ENST00000384369        6        0          0          0
ENST00000384745        5        1          4          0
ENST00000330957        6        0          0          2
ENST00000364942       20       51         23         60
```

This example counts reads that fall within 2000 bases upstream and 500 bases downstream of the first six genes on chromosome 21.

It would be good to know when seeing a lack of reads in some windows, if the mappability of the window is the cause. Some regions of the genome have low complexity sequence, and it is common practice in the analysis of sequencing data to discard reads that map to multiple locations in the genome. The function `mappabilityCalc` calculates the percentage of each region that can be mapped to by reads generated from the experiment. It operates on a user-created `BSgenome` object of a masked genome sequence. The definition of which bases are mappable and which are not depends on the fragment length of the sequencing technology used. Therefore, there is no one masked `BSgenome` object that can be used by all users. Note that by masking, it is meant replacing the unmappable reference sequence bases by 'N', not creating a built-in mask.

6 Summary

Repitools has a number of useful functions for quality checking, analysis, and comparison of trends. Many of the functions work seamlessly on array data, as well as sequencing data. Also, there are numerous utility functions, that perform some common task in the investigation of epigenomic data. Consult the package documentation for instructions on how to use functions that were not demonstrated by this vignette.

7 Environment

This vignette was created in:

```
> sessionInfo()

R version 2.12.0 (2010-10-15)
Platform: x86_64-unknown-linux-gnu (64-bit)

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

attached base packages:
[1] grid      stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] gplots_2.8.0                       caTools_1.11
 [3] bitops_1.0-4.1                     gdata_2.8.1
 [5] gtools_2.6.2                       edgeR_2.0.5
 [7] BSgenome.Hsapiens.UCSC.hg18_1.3.16 Repitools_1.80
 [9] BSgenome_1.18.3                    Biostrings_2.18.4
[11] GenomicRanges_1.2.3                IRanges_1.8.9
[13] R.methodsS3_1.2.1

loaded via a namespace (and not attached):
[1] Biobase_2.10.0 limma_3.6.9
```