Package: GenomicPlot (via r-universe)

July 3, 2024

Type Package

Title Plot profiles of next generation sequencing data in genomic features

Version 1.3.3

Description Visualization of next generation sequencing (NGS) data is essential for interpreting high-throughput genomics experiment results. 'GenomicPlot' facilitates plotting of NGS data in various formats (bam, bed, wig and bigwig); both coverage and enrichment over input can be computed and displayed with respect to genomic features (such as UTR, CDS, enhancer), and user defined genomic loci or regions. Statistical tests on signal intensity within user defined regions of interest can be performed and represented as boxplots or bar graphs. Parallel processing is used to speed up computation on multicore platforms. In addition to genomic plots which is suitable for displaying of coverage of genomic DNA (such as ChIPseq data), metagenomic (without introns) plots can also be made for RNAseq or CLIPseq data as well.

License GPL-2

Encoding UTF-8

LazyData FALSE

Collate ``DrawingFunctions.R'' ``GenomicPlot.R'' ``HandleDataMatrix.R'' ``HandleFeatures.R'' ``Parallel.R'' ``ReadData.R'' ``Setup.R'' ``Plot_5parts_metagene.R'' ``Plot_start_end.R'' ``Plot_start_end_with_random.R'' ``Plot_region.R'' ``Plot_locus.R'' ``data.R'' ``Plot_locus_with_random.R'' ``Plot_peak_annotation.R'' ``Plot_bam_correlation.R''

Depends R (>= 4.3.0), GenomicRanges (>= 1.46.1)

Imports methods, Rsamtools, parallel, tidyr, rtracklayer (>= 1.54.0), plyranges (>= 1.14.0), cowplot (>= 1.1.1), VennDiagram, ggplotify, GenomeInfoDb, IRanges, ComplexHeatmap, RCAS (>= 1.20.0), scales (>= 1.2.0), GenomicAlignments (>= 1.30.0), edgeR, circlize, viridis, ggsignif (>= 0.6.3), ggsci (>= 2.9),

ggpubr, grDevices, graphics, stats, utils, GenomicFeatures, genomation (>= 1.36.0), txdbmaker, ggplot2 (>= 3.3.5), BiocGenerics, dplyr, grid

Suggests knitr, rmarkdown, R.utils, Biobase, BiocStyle, testthat, AnnotationDbi

biocViews AlternativeSplicing, ChIPSeq, Coverage, GeneExpression, RNASeq, Sequencing, Software, Transcription, Visualization, Annotation

RoxygenNote 7.3.1

VignetteBuilder knitr

BugReports https://github.com/shuye2009/GenomicPlot/issues

URL https://github.com/shuye2009/GenomicPlot

Repository https://bioc.r-universe.dev

RemoteUrl https://github.com/bioc/GenomicPlot

RemoteRef HEAD

RemoteSha bec60b7cc44105b40a50bde730f42237d394345d

Contents

`aov_TukeyHSD` *Perform one-way ANOVA and post hoc TukeyHSD tests*

Description

This is a helper function for performing one-way ANOVA analysis and post hoc Tukey's Honest Significant Differences tests

Usage

```
aov_TukeyHSD(df, xc = "Group", yc = "Intensity", op = NULL, verbose = FALSE)
```

Arguments

df	a dataframe
xc	a string denoting column name for grouping
yc	a string denoting column name for numeric data to be plotted
op	output prefix for statistical analysis results
verbose	logical, to indicate whether a file should be produced to save the test results

Value

a list of two elements, the first is the p-value of ANOVA test and the second is a matrix of the output of TukeyHSD tests

Note

used in `plot_locus`

Author(s)

Shuye Pu

Examples

```
stat_df <- data.frame(
    Feature = rep(c("A", "B"), c(20, 30)),
    Intensity = c(rnorm(20, mean = 2, sd = 1), rnorm(30, mean = 3, sd = 1))
)

out <- aov_TukeyHSD(stat_df, xc = "Feature")
out
```

---

check_constraints *Check constraints of genomic ranges*

Description

Make sure the coordinates of GRanges are within the boundaries of chromosomes, and trim anything that goes beyond. Also, remove entries whose seqname is not in the seqname of a query GRanges.

Usage

```
check_constraints(gr, genome, queryRle = NULL)
```

Arguments

gr	a GenomicRanges object
genome	genomic version name such as "hg19"
queryRle	a RleList object used as a query against gr

Value

a GRanges object

Author(s)

Shuye Pu

Examples

```
subject <- GRanges("chr19",
    IRanges(rep(c(10, 15), 2), width = c(1, 20, 400, 2e+8)),
    strand = c("+", "+", "-", "-")
)

g <- check_constraints(gr = subject, genome = "hg19")
identical(g, subject)

subject1 <- GRanges("chr19",
    IRanges(rep(c(10, 15), 2), width = c(1, 20, 400, 28)),
    strand = c("+", "+", "-", "-")
)

g1 <- check_constraints(gr = subject1, genome = "hg19")
identical(g1, subject1)
```

---

custom_TxDb_from_GTF *Make custom TxDb object from a GTF/GFF file*

Description

This is a helper function for creating custom TxDb object from a GTF/GFF file. Mitochondrial chromosome is excluded.

Usage

```
custom_TxDb_from_GTF(gtfFile, genome = "hg19")
```

Arguments

gtfFile	path to a gene annotation gtf file
genome	a string denoting the genome name and version

Value

a TxDb object defined in the GenomicFeatures package.

Author(s)

Shuye Pu

Examples

```
gtfFile <- system.file("extdata", "gencode.v19.annotation_chr19.gtf",
    package = "GenomicPlot"
)

txdb <- custom_TxDb_from_GTF(gtfFile, genome = "hg19")
```

---

draw_boxplot_by_factor

Plot boxplot with two factors

Description

Plot violin plot with boxplot components for data with one or two factors, p-value significance levels are displayed, "***" = 0.001, "**" = 0.01, "*" = 0.05.

Usage

```
draw_boxplot_by_factor(
  stat_df,
  xc = "Feature",
  yc = "Intensity",
  fc = xc,
  comp = list(c(1, 2)),
  stats = "wilcox.test",
  Xlab = xc,
  Ylab = yc,
  nf = 1
)
```

Arguments

stat_df	a dataframe with column names c(xc, yc)
xc	a string denoting column name for grouping
yc	a string denoting column name for numeric data to be plotted
fc	a string denoting column name for sub-grouping based on an additional factor
comp	a list of vectors denoting pair-wise comparisons to be performed between groups

stats	the name of pair-wise statistical tests, like t.test or wilcox.test
Xlab	a string for x-axis label
Ylab	a string for y-axis label
nf	a integer normalizing factor for correct count of observations when the data table has two factors, such as those produced by 'pivot_longer', equals to the number of factors

Value

a ggplot object

Note

used by plot_locus, plot_locus_with_random, plot_region

Author(s)

Shuye Pu

Examples

```
stat_df <- data.frame(
    Feature = rep(c("A", "B"), c(20, 30)),
    Intensity = c(rnorm(20, 2, 0.5), rnorm(30, 3, 0.6))
)
p <- draw_boxplot_by_factor(stat_df,
    xc = "Feature", yc = "Intensity",
    Ylab = "Signal Intensity"
)
p
```

---

draw_boxplot_wo_outlier

Plot boxplot without outliers

Description

Plot boxplot without outliers, useful when outliers have a wide range and the median is squeezed at the bottom of the plot. The p-value significance level is the same as those in draw_boxplot_by_factor, but not displayed.

Usage

```
draw_boxplot_wo_outlier(
  stat_df,
  xc = "Feature",
  yc = "Intensity",
  fc = xc,
  comp = list(c(1, 2)),
  stats = "wilcox.test",
  Xlab = xc,
  Ylab = yc,
  nf = 1
)
```

Arguments

stat_df	a dataframe with column names c(xc, yc)
xc	a string denoting column name for grouping
yc	a string denoting column name for numeric data to be plotted
fc	a string denoting column name for sub-grouping
comp	a list of vectors denoting pair-wise comparisons to be performed between groups
stats	the name of pair-wise statistical tests, like t.test or wilcox.test
Xlab	a string for x-axis label
Ylab	a string for y-axis label
nf	a integer normalizing factor for correct count of observations when the data table has two factors, such as those produced by ‘pivot_longer‘, equals to the number of factors

Value

a ggplot object

Examples

```
stat_df <- data.frame(
    Feature = rep(c("A", "B"), c(20, 30)),
    Intensity = c(rnorm(20, 2), rnorm(30, 3))
)

p <- draw_boxplot_wo_outlier(stat_df,
    xc = "Feature", yc = "Intensity",
    Ylab = "Signal Intensity"
)
p
```

`draw_combo_plot` *Make combo plot for statistics plots*

Description

Place violin plot, boxplot without outliers, mean+se barplot and quantile plot on the same page

Usage

```
draw_combo_plot(
  stat_df,
  xc = "Feature",
  yc = "Intensity",
  comp = list(c(1, 2)),
  Xlab = xc,
  Ylab = yc,
  stats = "wilcox.test",
  fc = xc,
  Ylim = NULL,
  title = "",
  nf = 1
)
```

Arguments

`stat_df`	a dataframe with column names c(xc, yc)
`xc`	a string denoting column name for grouping
`yc`	a string denoting column name for numeric data to be plotted
`comp`	a list of vectors denoting pair-wise comparisons to be performed between groups
`Xlab`	a string for x-axis label
`Ylab`	a string for y-axis label
`stats`	the name of pair-wise statistical tests, like t.test or wilcox.test
`fc`	a string denoting column name for sub-grouping based on an additional factor
`Ylim`	a numeric vector of two elements, defining custom limits of y-axis
`title`	a string for plot title
`nf`	a integer normalizing factor for correct count of observations when the data table has two factors, such as those produced by pivot_longer, equals to the number of factors

Value

a ggplot object

Author(s)

Shuye Pu

Examples

```
stat_df <- data.frame(
    Feature = rep(c("A", "B"), c(200, 300)),
    Intensity = c(rnorm(200, 2, 5), rnorm(300, 3, 5)),
    Height = c(rnorm(200, 5, 5), rnorm(300, 1, 5))
)
stat_df_long <- tidyr::pivot_longer(stat_df,
    cols = c(Intensity, Height),
    names_to = "type", values_to = "value"
)

print(draw_combo_plot(stat_df_long,
    xc = "Feature", yc = "value", fc = "type",
    Ylab = "value", comp = list(c(1, 2), c(3, 4), c(1, 3), c(2, 4)), nf = 2
))
```

---

draw_locus_profile *Plot signal profile around genomic loci*

Description

Plot lines with standard error as the error band

Usage

```
draw_locus_profile(
  plot_df,
  xc = "Position",
  yc = "Intensity",
  cn = "Query",
  sn = NULL,
  Xlab = "Center",
  Ylab = "Signal Intensity",
  shade = FALSE,
  hl = c(0, 0)
)
```

Arguments

plot_df	a dataframe with column names c(xc, yc, cn, "lower", "upper")
xc	a string denoting column name for values on x-axis
yc	a string denoting column name for numeric data to be plotted

cn	a string denoting column name for sample grouping, like 'Query' or 'Reference'
sn	a string denoting column name for the subject of sample grouping, if 'cn' is 'Query', then 'sn' will be 'Reference'
Xlab	a string for x-axis label
Ylab	a string for y-axis label
shade	logical indicating whether to place a shaded rectangle around the loci bounded by hl
hl	a vector of two integers defining upstream and downstream boundaries of the rectangle

Value

a ggplot object

Note

used by plot_locus, plot_locus_with_random

Author(s)

Shuye Pu

Examples

```
library(dplyr)
Reference <- rep(rep(c("Ref1", "Ref2"), each = 100), 2)
Query <- rep(c("Query1", "Query2"), each = 200)
Position <- rep(seq(-50, 49), 4)
Intensity <- rlnorm(400)
se <- runif(400)
df <- data.frame(Intensity, se, Position, Query, Reference) %>%
    mutate(lower = Intensity - se, upper = Intensity + se) %>%
    mutate(Group = paste(Query, Reference, sep = ":"))

p <- draw_locus_profile(df, cn = "Group", shade = TRUE, hl = c(-10, 20))
p
```

---

draw_matrix_heatmap *Display matrix as a heatmap*

Description

Make a complex heatmap with column annotations

Usage

```
draw_matrix_heatmap(
  fullMatrix,
  dataName = "geneData",
  labels_col = NULL,
  levels_col = NULL,
  ranking = "Sum",
  ranges = NULL,
  verbose = FALSE
)
```

Arguments

fullMatrix	a numeric matrix
dataName	the nature of the numeric data
labels_col	a named vector for column annotation
levels_col	factor levels for names of labels_col, specifying the order of labels_col
ranking	method for ranking the rows of the input matrix, options are c("Sum", "Max", "Hierarchical", "None")
ranges	a numeric vector with three elements, defining custom range for color ramp, default=NULL, i.e. the range is defined automatically based on the c(minimun, median, maximum) of fullMatrix
verbose	logical, whether to output the input matrix for inspection

Value

a grob object

Author(s)

Shuye Pu

Examples

```
fullMatrix <- matrix(rnorm(10000), ncol = 100)
for (i in seq_len(80)) {
    fullMatrix[i, 16:75] <- runif(60) + i
}
labels_col <- as.character(seq_len(100))
levels_col <- c("start", "center", "end")
names(labels_col) <- rep(levels_col, c(15, 60, 25))

draw_matrix_heatmap(fullMatrix, dataName = "test", labels_col, levels_col,
  ranges = c(-2, 0, 20))
```

`draw_mean_se_barplot` *Plot barplot for mean with standard error bars*

Description

Plot barplot for mean with standard error bars, no p-value significance levels are displayed, but ANOVA p-value is provided as tag and TukeyHSD test are displayed as caption.

Usage

```
draw_mean_se_barplot(
  stat_df,
  xc = "Feature",
  yc = "Intensity",
  fc = xc,
  comp = list(c(1, 2)),
  Xlab = xc,
  Ylab = yc,
  Ylim = NULL,
  nf = 1
)
```

Arguments

`stat_df`	a dataframe with column names c(xc, yc)
`xc`	a string denoting column name for grouping
`yc`	a string denoting column name for numeric data to be plotted
`fc`	a string denoting column name for sub-grouping based on an additional factor
`comp`	a list of vectors denoting pair-wise comparisons to be performed between groups
`Xlab`	a string for x-axis label
`Ylab`	a string for y-axis label
`Ylim`	a numeric vector of two elements, defining custom limits of y-axis
`nf`	a integer normalizing factor for correct count of observations when the data table has two factors, such as those produced by pivot_longer, equals to the number of factors

Value

a ggplot object

Note

used by `plot_locus`, `plot_locus_with_random`

Author(s)

Shuye Pu

Examples

```
stat_df <- data.frame(
    Feature = rep(c("A", "B"), c(20, 30)),
    Intensity = c(rnorm(20, 2), rnorm(30, 3))
)
p <- draw_mean_se_barplot(stat_df,
    xc = "Feature", yc = "Intensity",
    Ylab = "Intensity"
)
p
```

---

draw_quantile_plot *Plot quantile over value*

Description

Plot quantiles as y-axis, and values as x-axis. Same as 'geom_ecdf', but allows sub-grouping by a second factor.

Usage

```
draw_quantile_plot(
  stat_df,
  xc = "Feature",
  yc = "Intensity",
  Ylab = yc,
  fc = xc
)
```

Arguments

stat_df	a dataframe with column names c(xc, yc)
xc	a string denoting column name for grouping
yc	a string denoting column name for numeric data to be plotted
Ylab	a string for y-axis label
fc	a string denoting column name for sub-grouping based on an additional factor

Value

a ggplot object

Note

used by plot_locus, plot_locus_with_random

Author(s)

Shuye Pu

Examples

```
stat_df <- data.frame(
    Feature = rep(c("A", "B"), c(20, 30)),
    Intensity = c(rnorm(20, 2, 5), rnorm(30, 3, 5)),
    Height = c(rnorm(20, 5, 5), rnorm(30, 1, 5))
)
stat_df_long <- tidyr::pivot_longer(stat_df,
    cols = c(Intensity, Height), names_to = "type",
    values_to = "value"
)

print(draw_quantile_plot(stat_df, xc = "Feature", yc = "Intensity"))
print(draw_quantile_plot(stat_df, xc = "Feature", yc = "Height"))
print(draw_quantile_plot(stat_df_long,
    xc = "Feature", yc = "value",
    fc = "type", Ylab = "value"
))
```

---

draw_rank_plot *Plot fraction of cumulative sum over rank*

Description

Plot cumulative sum over rank as line plot, both cumulative sum and rank are scaled between 0 and 1. This is the same as the fingerprint plot of the deepTools.

Usage

```
draw_rank_plot(stat_df, xc = "Feature", yc = "Intensity", Ylab = yc)
```

Arguments

stat_df	a dataframe with column names c(xc, yc)
xc	a string denoting column name for grouping
yc	a string denoting column name for numeric data to be plotted
Ylab	a string for y-axis label

Value

a ggplot object

Author(s)

Shuye Pu

Examples

```
stat_df <- data.frame(
    Feature = rep(c("A", "B"), c(20, 30)),
    Intensity = c(rlnorm(20, 5, 5), rlnorm(30, 1, 5))
)
stat_df1 <- data.frame(
    Feature = rep(c("A", "B"), c(20, 30)),
    Height = c(rnorm(20, 5, 5), rnorm(30, 1, 5))
)

print(draw_rank_plot(stat_df,
    xc = "Feature", yc = "Intensity",
    Ylab = "Intensity"
))
print(draw_rank_plot(stat_df1,
    xc = "Feature", yc = "Height",
    Ylab = "Height"
))
```

---

draw_region_landmark *Plot genomic region landmark indicator*

Description

Plot a gene centered polygon for demarcating gene and its upstream and downstream regions

Usage

```
draw_region_landmark(featureNames, vx, xmax)
```

Arguments

featureNames	a string vector giving names of sub-regions
vx	a vector on integers denoting the x coordinates of start of each sub-region
xmax	an integer denoting the left most boundary

Value

a ggplot object

Note

used by plot_5parts_metagene, plot_region

Author(s)

Shuye Pu

Examples

```
fn <- c("5'UTR", "CDS", "3'UTR")
mark <- c(1, 5, 20)
xmax <- 25

p <- draw_region_landmark(featureNames = fn, vx = mark, xmax = xmax)
```

---

draw_region_name *Plot genomic region names*

Description

Plot sub-region labels under the landmark

Usage

```
draw_region_name(featureNames, scaled_bins, xmax)
```

Arguments

featureNames	a string vector giving names of sub-regions
scaled_bins	a vector of integers denoting the lengths of each sub-region
xmax	an integer denoting the right most boundary

Value

a ggplot object

Note

used by plot_5parts_metagene, plot_region

Author(s)

Shuye Pu

Examples

```
fn <- c("5'UTR", "CDS", "3'UTR")
bins <- c(5, 15, 5)
xmax <- 25

p <- draw_region_name(featureNames = fn, scaled_bins = bins, xmax = xmax)
```

`draw_region_profile` *Plot signal profile in genomic regions*

Description

Plot lines with standard error as the error band

Usage

```
draw_region_profile(
  plot_df,
  xc = "Position",
  yc = "Intensity",
  cn = "Query",
  sn = NULL,
  Ylab = "Signal Intensity",
  vx
)
```

Arguments

`plot_df`	a dataframe with column names c(xc, yc, cn, "lower", "upper")
`xc`	a string denoting column name for values on x-axis
`yc`	a string denoting column name for numeric data to be plotted
`cn`	column name in plot_df for query samples grouping
`sn`	column name in plot_df for subject name to be shown in the plot title
`Ylab`	a string for Y-axis label
`vx`	a vector on integers denoting the x coordinates of start of each sub-region

Value

a ggplot object

Note

used by `plot_5parts_metagene`, `plot_region`

Author(s)

Shuye Pu

Examples

```
library(dplyr)
Reference <- rep(rep(c("Ref1", "Ref2"), each = 100), 2)
Query <- rep(c("Query1", "Query2"), each = 200)
Position <- rep(seq_len(100), 4)
Intensity <- rlnorm(400)
se <- runif(400)
df <- data.frame(Intensity, se, Position, Query, Reference) %>%
    mutate(lower = Intensity - se, upper = Intensity + se) %>%
    mutate(Group = paste(Query, Reference, sep = ":"))
vx <- c(1, 23, 70)

p <- draw_region_profile(df, cn = "Group", vx = vx)
p
```

`draw_stacked_plot` *draw stacked plot*

Description

Plot profile on top of heatmap, and align feature labels.

Usage

```
draw_stacked_plot(plot_list, heatmap_list)
```

Arguments

`plot_list`	a list of profile plots
`heatmap_list`	a list of heatmaps

Value

a null value

Note

used by `plot_locus`, `plot_5parts_metagene`, `plot_region`

Author(s)

Shuye Pu

`draw_stacked_profile` *Plot signal profile around start, center, and end of genomic regions*

Description

Plot lines with standard error as the error band, also plots number of regions having non-zero signals

Usage

```
draw_stacked_profile(
  plot_df,
  xc = "Position",
  yc = "Intensity",
  cn = "Query",
  ext = c(0, 0, 0, 0),
  hl = c(0, 0, 0, 0),
  atitle = "title",
  insert = 0,
  Ylab = "Signal Intensity",
  shade = FALSE,
  stack = TRUE
)
```

Arguments

`plot_df`	a dataframe with column names c(xc, yc, cn, "Interval", "lower", "upper")
`xc`	a string denoting column name for values on x-axis
`yc`	a string denoting column name for numeric data to be plotted
`cn`	a string denoting column name for grouping
`ext`	a vector of 4 integers denoting upstream and downstream extension around start and end, the range of extensions must be within the range of ‘xc‘ of the ‘plot_df‘
`hl`	a vector of 4 integers defining upstream and downstream boundaries of the rectangle for start and end
`atitle`	a string for the title of the plot
`insert`	a integer denoting the width of the center region
`Ylab`	a string for y-axis label
`shade`	logical, indicating whether to place a shaded rectangle around the point of interest
`stack`	logical, indicating whether to plot the number of valid (non-zero) data points in each bin

Value

a ggplot object

Note

used by plot_start_end, plot_start_end_with_random

Author(s)

Shuye Pu

Examples

```
library(dplyr)
Reference <- rep(rep(c("Ref1", "Ref2"), each = 100), 2)
Query <- rep(c("Query1", "Query2"), each = 200)
Position <- rep(seq(-50, 49), 4)
Intensity <- rlnorm(400)
se <- runif(400)
start_df <- data.frame(Intensity, se, Position, Query, Reference) %>%
    mutate(lower = Intensity - se, upper = Intensity + se) %>%
    mutate(Group = paste(Query, Reference, sep = ":")) %>%
    mutate(Location = rep("Start", 400)) %>%
    mutate(Interval = sample.int(1000, 400))
Intensity <- rlnorm(400, meanlog = 1.5)
se <- runif(400)
center_df <- data.frame(Intensity, se, Position, Query, Reference) %>%
    mutate(lower = Intensity - se, upper = Intensity + se) %>%
    mutate(Group = paste(Query, Reference, sep = ":")) %>%
    mutate(Location = rep("Center", 400)) %>%
    mutate(Interval = sample.int(600, 400))
Intensity <- rlnorm(400, meanlog = 2)
se <- runif(400)
end_df <- data.frame(Intensity, se, Position, Query, Reference) %>%
    mutate(lower = Intensity - se, upper = Intensity + se) %>%
    mutate(Group = paste(Query, Reference, sep = ":")) %>%
    mutate(Location = rep("End", 400)) %>%
    mutate(Interval = sample.int(2000, 400))

df <- rbind(start_df, center_df, end_df)
p <- draw_stacked_profile(df, cn = "Group", shade = TRUE,
    ext = c(-50, 50, -50, 50),
    hl = c(-20, 20, -25, 25), insert = 100)
p
```

`effective_size` *Normalize sample library size to effective size*

Description

This is a helper function for handle_input. edgeR::calcNormFactors function is used to estimate normalizing factors, which is used to multiply library sizes.

Usage

```
effective_size(outlist, outRle, genome = "hg19", nc = 2, verbose = FALSE)
```

Arguments

outlist	a list of list objects with four elements, 'query' is a GRanges object, 'size' is the library size, 'type' is the input file type, 'weight' is the name of the metadata column
outRle	logical, indicating whether the 'query' element of the output should be an RleList object or a GRanges object
genome	a string denoting the genome name and version
nc	integer, number of cores for parallel processing
verbose	logical, whether to output additional information

Value

a list of list objects with four elements ('query', 'size', 'type', 'weight'), with the 'size' element modified.

Author(s)

Shuye Pu

Examples

```
queryFiles <- system.file("extdata", "chip_treat_chr19.bam",
    package = "GenomicPlot"
)
names(queryFiles) <- "query"

inputFiles <- system.file("extdata", "chip_input_chr19.bam",
    package = "GenomicPlot"
)
names(inputFiles) <- "input"

chipImportParams <- setImportParams(
    offset = 0, fix_width = 150, fix_point = "start", norm = TRUE,
    useScore = FALSE, outRle = FALSE, useSizeFactor = FALSE, genome = "hg19"
)

out_list <- handle_input(
    inputFiles = c(queryFiles, inputFiles),
    importParams = chipImportParams, verbose = TRUE, nc = 2
)

out <- effective_size(out_list, outRle = TRUE)
```

`extdata` *Toy data for examples and testing of the 'GenomicPlot' package*

Description

The data files in the extdata directory contain data for next generation sequencing read alignments, MACS2 peaks and gene annotation, which are used to test the package and generate plots in the package vignettes. To meet the package file size limit, all data are restricted to chr19:58000-507000 of the human genome version hg19. Details for each file are as follows.

Details

- "gencode.v19.annotation_chr19.gtf" is an excerpt of a gene annotation file by limiting to chr19:58000-507000 of the human genome.
- "gencode.v19.annotation_chr19.gtf.granges.rds" is a GRanges object produced by importing the above gtf file using RCAS::importGtf.
- "chip_treat_chr19.bam(.bai)" and "chip_input_chr19.bam(.bai)" are paired-end read alignment data from ChIPseq experiments.
- "treat_chr19.bam(.bai)" and "input_chr19.bam(.bai)" are single-end read alignment data from iCLIP experiments.
- "test_wig_chr19_+(-).wig", "test_wig_chr19_+(-).bw" are iCLIP alignment data in WIG and BIGWIG format, respectively; '+' and '-' represent forward and reverse strand, respectively.
- "test_clip_peak_chr19.bed" contains strand-specific iCLIP peak in BED format.
- "test_chip_peak_chr19.bed" and "test_chip_peak_chr19.narrowPeak" contain ChIPseq peaks generated with MACS2, in summit peak and narrow peak format, respectively. "test_chr19.bedGraph" contains the same data in bedGraph format.
- "test_file1.txt", "test_file2.txt", "test_file3.txt" and "test_file4.txt" are tab-delimited text files, each contains various human gene names in different columns.

Value

Various files used as inputs to run examples and tests

Author(s)

Shuye Pu

Source

The original gene annotation (gtf) file is downloaded from https://www.gencodegenes.org/human/. Except for the gtf file, all other files are derived from experimental data produced in-house at the Greenblatt Lab, University of Toronto, Canada.

`extract_longest_tx` *Extract the longest transcript for each protein-coding genes*

Description

Gene level computations require selecting one transcript per gene to avoid bias by genes with multiple isoforms. In ideal case, the most abundant transcript (principal or canonical isoform) should be chosen. However, the most abundant isoform may vary depending on tissue type or physiological condition, the longest transcript is usually the principal isoform, and alternatively spliced isoforms are not. This method get the longest transcript for each gene. The longest transcript is defined as the isoform that has the longest transcript length. In case of tie, the one with longer CDS is selected. If the lengths of CDS tie again, the transcript with smaller id is selected arbitrarily.

Usage

```
extract_longest_tx(txdb)
```

Arguments

`txdb`	a TxDb object defined in the GenomicFeatures package

Value

a dataframe of transcript information with the following columns: "tx_id tx_name gene_id nexon tx_len cds_len utr5_len utr3_len"

Author(s)

Shuye Pu

Examples

```
gtfFile <- system.file("extdata", "gencode.v19.annotation_chr19.gtf",
    package = "GenomicPlot"
)

txdb <- custom_TxDb_from_GTF(gtfFile, genome = "hg19")
longestTx <- extract_longest_tx(txdb)
```

`filter_by_nonoverlaps_stranded` *Filter GRanges by nonoverlaps in a stranded way*

Description

This function reports all query GRanges that do not overlaps GRanges in subject. Strand information is used to define overlap.

Usage

```
filter_by_nonoverlaps_stranded(
  query,
  subject,
  maxgap = -1L,
  minoverlap = 0L,
  ignore.order = TRUE
)
```

Arguments

`query`	a GRanges object
`subject`	a GRanges object
`maxgap`	an integer denoting the distance that define overlap
`minoverlap`	The minimum amount of overlap between intervals as a single integer greater than 0. If you modify this argument, maxgap must be held fixed.
`ignore.order`	logical, indicating whether the order of query and subject can be switched, default = TRUE. This parameter is used to avoid the situation that the size of overlaps is bigger than the size of subject, which will produce an error when plotting Venn diagrams.

Value

a GRanges object

Author(s)

Shuye Pu

Examples

```
query <- GRanges("chr19",
    IRanges(rep(c(10, 15), 2), width = c(1, 20, 40, 50)),
    strand = c("+", "+", "-", "-")
)

subject <- GRanges("chr19",
```

```
    IRanges(rep(c(13, 150), 2), width = c(10, 14, 20, 28)),
    strand = c("+", "-", "-", "+")
)

res <- filter_by_nonoverlaps_stranded(query, subject)
res
```

`filter_by_overlaps_nonstranded` *Filter GRanges by overlaps in a nonstranded way*

Description

This function reports all query GRanges that have overlaps in subject GRanges. Strand information is not required.

Usage

```
filter_by_overlaps_nonstranded(
  query,
  subject,
  maxgap = -1L,
  minoverlap = 0L,
  ignore.order = TRUE
)
```

Arguments

`query`	a GRanges object
`subject`	a GRanges object
`maxgap`	an integer denoting the distance that define overlap
`minoverlap`	The minimum amount of overlap between intervals as a single integer greater than 0. If you modify this argument, maxgap must be held fixed.
`ignore.order`	logical, indicating whether the order of query and subject can be switched, default = TRUE. This parameter is used to avoid the situation that the size of overlaps is bigger than the size of subject, which will produce an error when plotting Venn diagrams.

Value

a GRanges object

Author(s)

Shuye Pu

Examples

```
query <- GRanges("chr19",
    IRanges(rep(c(10, 15), 2), width = c(1, 20, 40, 50)),
    strand = c("+", "+", "-", "-")
)

subject <- GRanges("chr19",
    IRanges(rep(c(13, 150), 2), width = c(10, 14, 20, 28)),
    strand = c("+", "-", "-", "+")
)

res <- filter_by_overlaps_nonstranded(query, subject, ignore.order = TRUE)
res
```

`filter_by_overlaps_stranded` *Filter GRanges by overlaps in a stranded way*

Description

This function reports all query GRanges that have overlaps in subject GRanges. Strand information is used to define overlap.

Usage

```
filter_by_overlaps_stranded(
  query,
  subject,
  maxgap = -1L,
  minoverlap = 0L,
  ignore.order = TRUE
)
```

Arguments

`query`	a GRanges object
`subject`	a GRanges object
`maxgap`	an integer denoting the distance that define overlap
`minoverlap`	The minimum amount of overlap between intervals as a single integer greater than 0. If you modify this argument, maxgap must be held fixed.
`ignore.order`	logical, indicating whether the order of query and subject can be switched, default = TRUE. Overlaps in query and subject often have different sizes. This parameter will make the function use whichever is smaller to avoid errors when making Venn diagrams.

Value

a GRanges object

Author(s)

Shuye Pu

Examples

```
query <- GRanges("chr19",
    IRanges(rep(c(10, 15), 2), width = c(1, 20, 40, 50)),
    strand = c("+", "+", "-", "-")
)

subject <- GRanges("chr19",
    IRanges(rep(c(13, 150), 2), width = c(10, 14, 20, 28)),
    strand = c("+", "-", "-", "+")
)

res <- filter_by_overlaps_stranded(query, subject)
res
resf <- filter_by_overlaps_stranded(query, subject, ignore.order = FALSE)
resf
```

---

gene2tx *Translate gene names to transcript ids using a GTF file for a subset of genes*

Description

Given a list of gene names in a file or in a character vector, turn them into a vector of transcript ids.

Usage

```
gene2tx(gtfFile, geneList, geneCol = 1)
```

Arguments

gtfFile	path to a GTF file
geneList	path to a tab-delimited text file with one gene name on each line, or a character vector of gene names (eg. RPRD1B)
geneCol	the position of the column that containing gene names in the case that geneList is a file

Value

a vector of transcript ids (eg. ENST00000577222.1)

Author(s)

Shuye Pu

Examples

```
gtfFile <- system.file("extdata", "gencode.v19.annotation_chr19.gtf",
    package = "GenomicPlot"
)
genes <- c("RPRD1A", "RPAP2", "RPRD1B", "RPRD2", "ZNF281", "YTHDF2")

tx <- gene2tx(gtfFile = gtfFile, geneList = genes)
```

GenomicPlot *GenomicPlot-package*

Description

An R package for efficient and flexible visualization of genome-wide NGS coverage profiles

Details

The goal of 'GenomicPlot' is to provide an efficient visualization tool for next generation sequencing (NGS) data with rich functionality and flexibility. 'GenomicPlot' enables plotting of NGS data in various formats (bam, bed, wig and bigwig); both coverage and enrichment over input can be computed and displayed with respect to genomic features (such as UTR, CDS, enhancer), and user defined genomic loci or regions. Statistical tests on signal intensity within user defined regions of interest can be performed and presented as box plots or pie charts. Parallel processing is enabled to speed up computation on multi-core platforms. Main functions are as follows:

- `plot_5parts_metagene` generates genomic (with introns) or metagenomic (without introns) plots around gene body and its upstream and downstream regions, the gene body can be further segmented into 5'UTR, CDS and 3'UTR.
- `plot_start_end` plots genomic profiles around the start and end of genomic features (like exons or introns), or user defined genomic regions. A center region with user defined width can be plotted simultaneously.
- `plot_locus` plots distance between sample peaks and genomic features, or distance from one set of peaks to another set of peaks.
- `plot_region` plots signal profiles within and around genomic features, or user defined genomic regions.
- `plot_peak_annotation` plots peak annotation statistics (distribution in different type of genes, and in different parts of genes).
- `plot_overlap_bed` plots peak overlaps as Venn diagrams.
- Random features can be generated and plotted to serve as contrast to real features in `plot_locus_with_random` and `plot_start_end_with_random`.
- All profile line plots have error bands.
- Statistical analysis results on user defined regions of interest are plotted along with the profile plots in `plot_region`, `plot_locus` and `plot_locus_with_random`.

Author(s)

Shuye Pu

_PACKAGE

`get_genomic_feature_coordinates` *Extract genomic features from TxDb object*

Description

Extract genomic coordinates and make bed or bed 12 files from a TxDb object for a variety of annotated genomic features. The output of this function is a list. The first element of the list is a GRanges object that provide the start and end information of the feature. The second element is a GRangesList providing information for sub-components. The third element is the name of a bed file.

Usage

```
get_genomic_feature_coordinates(
  txdb,
  featureName,
  featureSource = NULL,
  export = FALSE,
  longest = FALSE,
  protein_coding = FALSE
)
```

Arguments

`txdb`	a TxDb object defined in the GenomicFeatures package
`featureName`	one of the genomic feature in c("utr3", "utr5", "cds", "intron", "exon", "transcript", "gene")
`featureSource`	the name of the gtf/gff3 file or the online database from which txdb is derived, used as name of output file
`export`	logical, indicating if the bed file should be produced
`longest`	logical, indicating whether the output should be limited to the longest transcript of each gene
`protein_coding`	logical, indicating whether to limit to protein_coding genes

Details

For "utr3", "utr5", "cds" and "transcript", the GRanges object denotes the start and end of the feature in one transcript, and the range is named by the transcript id and may span introns; the GrangesList object is a list of exons comprising each feature and indexed on transcript id. The bed file is in bed12 format. For "exon" and "intron", the GRanges object denotes unnamed ranges of individual exon and intron, and the GrangesList object is a list of exons or introns belonging to one transcript and indexed on transcript id. The bed file is in bed6 format. For "gene", both GRanges object and GRangesList object have the same ranges and names. The bed file is in bed6 format.

Value

a list of three objects, the first is a GRanges object, the second is a GRangesList object, the last is the output file name if export is TRUE.

Author(s)

Shuye Pu

Examples

```
gtfFile <- system.file("extdata", "gencode.v19.annotation_chr19.gtf",
    package = "GenomicPlot"
)

txdb <- custom_TxDb_from_GTF(gtfFile, genome = "hg19")

output <- get_genomic_feature_coordinates(txdb,
    featureName = "cds", featureSource = "gencode",
    export = FALSE, longest = TRUE, protein_coding = TRUE
)
```

---

get_targeted_genes *Get the number of peaks overlapping each feature of all protein-coding genes*

Description

Annotate each peak with genomic features based on overlap, and produce summary statistics for distribution of peaks in features of protein-coding genes. If a peak overlap multiple features, a feature is assigned to the peak in the following order of precedence: "5'UTR", "3'UTR", "CDS", "Intron", "Promoter", "TTS".

Usage

```
get_targeted_genes(peak, features, stranded = TRUE)
```

Arguments

peak	a GRanges object defining query ranges
features	a GRangesList object representing genomic features
stranded	logical, indicating whether the overlap should be strand-specific

Value

a list object

Note

used in plot_peak_annotation

Author(s)

Shuye Pu

Examples

```
gtfFile <- system.file("extdata", "gencode.v19.annotation_chr19.gtf",
    package = "GenomicPlot"
)

txdb <- custom_TxDb_from_GTF(gtfFile, genome = "hg19")
f <- get_txdb_features(txdb, dsTSS = 100, fiveP = 0, threeP = 1000)

p <- RCAS::importBed(system.file("extdata", "test_chip_peak_chr19.bed",
    package = "GenomicPlot"
))
ann <- get_targeted_genes(peak = p, features = f, stranded = FALSE)
```

---

get_txdb_features *Get genomic coordinates of features of protein-coding genes*

Description

Get genomic coordinates of promoter, 5'UTR, CDS, 3'UTR, TTS and intron for the longest transcript of protein-coding genes. The range of promoter is defined by fiveP and dsTSS upstream and downstream TSS, respectively, the TTS ranges from the 3' end of the gene to threeP downstream, or the start of a downstream gene, whichever is closer.

Usage

```
get_txdb_features(txdb, fiveP = -1000, dsTSS = 300, threeP = 1000, nc = 2)
```

Arguments

txdb	a TxDb object defined in the GenomicFeatures package
fiveP	extension upstream of the 5' boundary of genes
dsTSS	range of promoter extending downstream of TSS
threeP	extension downstream of the 3' boundary of genes
nc	number of cores for parallel processing

Value

a GRangesList object

Author(s)

Shuye Pu

Examples

```
gtfFile <- system.file("extdata", "gencode.v19.annotation_chr19.gtf",
    package = "GenomicPlot"
)

txdb <- custom_TxDb_from_GTF(gtfFile, genome = "hg19")

f <- get_txdb_features(txdb, dsTSS = 100, fiveP = -100, threeP = 100)
```

---

gf5_genomic *Toy data for examples and testing of the 'GenomicPlot' package*

Description

Genomic coordinates of 72 transcripts in hg19 for genomic features promoter, 5'UTR, CDS, 3'UTR, TTS, as well as user inputs for processing these features. See prepare_5parts_genomic_features for details.

Value

A named list with the following elements:

windowRs a list of 5 GrangesList objects for the 5 genomic features

nbins a positive integer

scaled_bins a vector of 5 integers

fiveP a negative integer

threeP a positive integer

meta logical

longest logical

Author(s)

Shuye Pu

Source

The data is produced by running the following code:
txdb <- AnnotationDbi::loadDb(system.file("extdata", "txdb.sql", package = "GenomicPlot"))
gf5_genomic <- GenomicPlot::prepare_5parts_genomic_features(txdb, meta = FALSE, nbins = 100, fiveP = -2000, threeP = 1000, longest = TRUE)

---

gf5_meta *Toy data for examples and testing of the 'GenomicPlot' package*

Description

Metagenomic coordinates of 72 transcripts in hg19 for genomic features promoter, 5'UTR, CDS, 3'UTR, TTS, as well as user inputs for processing these features. See `prepare_5parts_genomic_features` for details.

Value

A named list with the following elements:

windowRs a list of 5 GrangesList objects for the 5 genomic features

nbins a positive integer

scaled_bins a vector of 5 integers

fiveP a negative integer

threeP a positive integer

meta logical

longest logical

Author(s)

Shuye Pu

Source

The data is produced by running the following code:
txdb <- AnnotationDbi::loadDb(system.file("extdata", "txdb.sql", package = "GenomicPlot"))
gf5_meta <- GenomicPlot::prepare_5parts_genomic_features(txdb, meta = TRUE, nbins = 100, fiveP = -2000, threeP = 1000, longest = TRUE)

gr2df *Convert GRanges to dataframe*

Description

Convert a GRanges object with meta data columns to a dataframe, with the first 6 columns corresponding those of BED6 format, and the meta data as additional columns

Usage

```
gr2df(gr)
```

Arguments

gr	a GRanges object

Value

a dataframe

Author(s)

Shuye Pu

Examples

```
gr2 <- GenomicRanges::GRanges(c("chr1", "chr1"),
    IRanges::IRanges(c(7, 13), width = 3),
    strand = c("+", "-")
)
GenomicRanges::mcols(gr2) <- data.frame(
    score = c(0.3, 0.9),
    cat = c(TRUE, FALSE)
)
df2 <- gr2df(gr2)
```

handle_bam *Handle files in bam format*

Description

This is a function for read NGS reads data in bam format, store the input data in a list of GRanges objects or RleList objects. For paired-end reads, only take the second read in a pair, assuming which is the sense read for strand-specific RNAseq.

Usage

```
handle_bam(inputFile, importParams = NULL, verbose = FALSE)
```

Arguments

inputFile	a string denoting path to the input file
importParams	a list of parameters, refer to handle_input for details
verbose	logical, whether to output additional information

Details

The reads are filtered using mapq score >= 10 by default, only mapped reads are counted towards library size.

Value

a list object with four elements, 'query' is a list GRanges objects or RleList objects, 'size' is the library size, 'type' is the input file type, weight' is the name of the metadata column to be used as weight for coverage calculation

Author(s)

Shuye Pu

Examples

```
queryFiles <- system.file("extdata", "treat_chr19.bam",
    package = "GenomicPlot"
)
names(queryFiles) <- "query"

bamimportParams <- setImportParams(
    offset = -1, fix_width = 0, fix_point = "start", norm = TRUE,
    useScore = FALSE, outRle = TRUE, useSizeFactor = FALSE, genome = "hg19"
)

out <- handle_bam(
    inputFile = queryFiles, importParams = bamimportParams, verbose = TRUE
)
```

handle_bed *Handle files in bed\|narrowPeak\|broadPeak format*

Description

This is a function for read peaks data in bed format, store the input data in a list of GRanges objects or RleList objects.

Usage

```
handle_bed(inputFile, importParams = NULL, verbose = FALSE)
```

Arguments

inputFile	a string denoting path to the input file
importParams	a list of parameters, refer to handle_input for details
verbose	logical, whether to output additional information

Value

a list object with four elements, 'query' is a list GRanges objects or RleList objects, 'size' is the library size, 'type' is the input file type, 'weight' is the name of the metadata column to be used as weight for coverage calculation

Author(s)

Shuye Pu

Examples

```
queryFiles <- system.file("extdata", "test_chip_peak_chr19.narrowPeak",
    package = "GenomicPlot"
)
names(queryFiles) <- "narrowPeak"

bedimportParams <- setImportParams(
    offset = 0, fix_width = 100, fix_point = "center", norm = FALSE,
    useScore = FALSE, outRle = TRUE, useSizeFactor = FALSE, genome = "hg19"
)

out <- handle_bed(queryFiles, bedimportParams, verbose = TRUE)
lapply(out$query, sum)
```

handle_bedGraph *Handle files in bedGraph format*

Description

This is a function for read peaks data in bedGraph format, store the input data in a list of GRanges objects or RleList objects.

Usage

```
handle_bedGraph(inputFile, importParams = NULL, verbose = FALSE)
```

Arguments

inputFile	a string denoting path to the input file
importParams	a list of parameters, refer to handle_input for details
verbose	logical, whether to output additional information

Value

a list object with four elements, 'query' is a list GRanges objects or RleList objects, 'size' is the library size, 'type' is the input file type, 'weight' is the name of the metadata column to be used as weight for coverage calculation

Author(s)

Shuye Pu

Examples

```
queryFiles <- system.file("extdata", "test_chr19.bedGraph",
    package = "GenomicPlot"
)
names(queryFiles) <- "chipPeak"

importParams <- setImportParams(
    offset = 0, fix_width = 0, fix_point = "start", norm = FALSE,
    useScore = TRUE, outRle = FALSE, useSizeFactor = FALSE, genome = "hg19",
    val = 4, skip = 1
)

out <- handle_bedGraph(queryFiles, importParams, verbose = TRUE)
out$query
```

handle_bw *Handle files in bw|bigwig|bigWig|BigWig|BW|BIGWIG format*

Description

This is a function for read NGS coverage data in bigwig format, store the input data in a list of GRanges objects or RleList objects. The input bw file can be stranded or non-stranded. Library size is calculate as the sum of all coverage.

Usage

```
handle_bw(inputFile, importParams, verbose = FALSE)
```

Arguments

inputFile	a string denoting path to the input file
importParams	a list of parameters, refer to handle_input for details
verbose	logical, whether to output additional information

Details

For stranded files, forward and reverse strands are stored in separate files, with '+' or 'p' in the forward strand file name and '-' or 'm' in the reverse strand file name.

Value

a list object with four elements, 'query' is a list GRanges objects or RleList objects, 'size' is the estimated library size, 'type' is the input file type, weight' is the name of the metadata column to be used as weight for coverage calculation

Author(s)

Shuye Pu

Examples

```
queryFiles <- system.file("extdata", "test_wig_chr19_+.bw",
    package = "GenomicPlot"
)
names(queryFiles) <- "test_bw"

wigimportParams <- setImportParams(
    offset = 0, fix_width = 0, fix_point = "start", norm = FALSE,
    useScore = FALSE, outRle = TRUE, useSizeFactor = FALSE, genome = "hg19"
)

out <- handle_bw(queryFiles, wigimportParams, verbose = TRUE)
```

handle_input *Handle import of NGS data with various formats*

Description

This is a wrapper function for read NGS data in different file formats, store the input data in a list of GRanges objects or RleList objects. File names end in bed|bam|bw|bigwig|bigWig|BigWig|BW|BIGWIG are recognized, and a named list of files with mixed formats are allowed.

Usage

```
handle_input(inputFiles, importParams = NULL, verbose = FALSE, nc = 2)
```

Arguments

inputFiles	a vector of strings denoting file names
importParams	a list with the 9 elements: list(offset, fix_width, fix_point, useScore, outRle, norm, genome, useSizeFactor). Details are described in the documentation of setImportParams function
verbose	logical, whether to output additional information
nc	integer, number of cores for parallel processing

Details

when 'useScore' is TRUE, the score column of the bed file will be used in the metadata column 'score' of the GRanges object, or the 'Values' field of the RleList object. Otherwise the value 1 will be used instead. When the intended use of the input bed is a reference feature, both 'useScore' and 'outRle' should be set to FALSE.

Value

a list object with four elements, 'query' is a list GRanges objects or RleList objects, 'size' is the library size, 'type' is the input file type, 'weight' is the name of the metadata column to be used as weight for coverage calculation

Author(s)

Shuye Pu

Examples

```
queryFiles1 <- system.file("extdata", "treat_chr19.bam",
    package = "GenomicPlot"
)
names(queryFiles1) <- "query"

inputFiles1 <- system.file("extdata", "input_chr19.bam",
```

```
    package = "GenomicPlot"
)
names(inputFiles1) <- "input"

bamimportParams <- setImportParams(
    offset = -1, fix_width = 0, fix_point = "start", norm = TRUE,
    useScore = FALSE, outRle = TRUE, useSizeFactor = FALSE, genome = "hg19"
)

out_list <- handle_input(
    inputFiles = c(queryFiles1, inputFiles1),
    importParams = bamimportParams, verbose = TRUE, nc = 2
)

queryFiles2 <- system.file("extdata", "test_wig_chr19_+.wig",
    package = "GenomicPlot"
)
names(queryFiles2) <- "test_wig"

wigimportParams <- setImportParams(
    offset = 0, fix_width = 0, fix_point = "start", norm = FALSE,
    useScore = FALSE, outRle = TRUE, useSizeFactor = FALSE, genome = "hg19"
)

out <- handle_input(queryFiles2, wigimportParams, verbose = TRUE)

queryFiles3 <- system.file("extdata", "test_wig_chr19_+.bw",
    package = "GenomicPlot"
)
names(queryFiles3) <- "test_bw"

out <- handle_input(c(queryFiles1, queryFiles2, queryFiles3),
    wigimportParams,
    verbose = TRUE
)
```

---

handle_wig *Handle files in wig format*

Description

This is a function for read NGS coverage data in wig format, store the input data in a list of GRanges objects or RleList objects. The input wig file can be stranded or non-stranded. Library size is calculate as the sum of all coverage.

Usage

```
handle_wig(inputFile, importParams, verbose = FALSE)
```

Arguments

inputFile	a string denoting path to the input file
importParams	a list of parameters, refer to handle_input for details
verbose	logical, whether to output additional information

Details

For stranded files, forward and reverse strands are stored in separate files, with '+' or 'p' in the forward strand file name and '-' or 'm' in the reverse strand file name.

Value

a list object with four elements, 'query' is a list GRanges objects or RleList objects, 'size' is the library size, 'type' is the input file type, 'weight' is the name of the metadata column to be used as weight for coverage calculation

Author(s)

Shuye Pu

Examples

```
queryFiles <- system.file("extdata", "test_wig_chr19_+.wig",
    package = "GenomicPlot"
)
names(queryFiles) <- "test_wig"

wigimportParams <- setImportParams(
    offset = 0, fix_width = 0, fix_point = "start", norm = FALSE,
    useScore = FALSE, outRle = TRUE, useSizeFactor = FALSE, genome = "hg19"
)

out <- handle_wig(queryFiles, wigimportParams, verbose = TRUE)
```

---

make_subTxDb_from_GTF *Make TxDb object from a GTF file for a subset of genes*

Description

Make a partial TxDb object given a GTF file and a list of gene names in a file or in a character vector.

Usage

```
make_subTxDb_from_GTF(gtfFile, genome = "hg19", geneList, geneCol = 1)
```

Arguments

gtfFile	path to a GTF file
genome	version of genome, like "hg19"
geneList	path to a tab-delimited text file with one gene name on each line, or a character vector of gene names
geneCol	the position of the column that containing gene names in the case that geneList is a file

Value

a TxDb object

Author(s)

Shuye Pu

Examples

```
gtfFile <- system.file("extdata", "gencode.v19.annotation_chr19.gtf",
    package = "GenomicPlot"
)
genes <- c("RPRD1A", "RPAP2", "RPRD1B", "RPRD2", "ZNF281", "YTHDF2")

txdb <- make_subTxDb_from_GTF(gtfFile = gtfFile, geneList = genes)
```

---

overlap_pair *Plot two-sets Venn diagram*

Description

This is a helper function for Venn diagram plot. A Venn diagram is plotted as output. For GRanges, as A overlap B may not be the same as B overlap A, the order of GRanges in a list matters, certain order may produce an error.

Usage

```
overlap_pair(apair, overlap_fun, title = NULL)
```

Arguments

apair	a list of two vectors
overlap_fun	the name of the function that defines overlap, depending on the type of object in the vectors. For GRanges, use filter_by_overlaps_stranded or filter_by_nonoverlaps_stranded, for gene names, use intersect.
title	main title of the figure

Value

a VennDiagram object

Author(s)

Shuye Pu

Examples

```
test_list <- list(A = c(1, 2, 3, 4, 5), B = c(4, 5, 7))
overlap_pair(test_list, intersect, title = "test")

## GRanges overlap
query <- GRanges("chr19",
    IRanges(rep(c(10, 15), 2), width = c(1, 20, 40, 50)),
    strand = c("+", "+", "-", "-")
)

subject <- GRanges("chr19",
    IRanges(rep(c(13, 150), 2), width = c(10, 14, 20, 28)),
    strand = c("+", "-", "-", "+")
)

overlap_pair(
    list(query = query, subject = subject),
    filter_by_overlaps_stranded
)
```

---

overlap_quad *Plot four-sets Venn diagram*

Description

This is a helper function for Venn diagram plot. A Venn diagram is plotted as output. For GRanges, as A overlap B may not be the same as B overlap A, the order of GRanges in a list matters, certain order may produce an error.

Usage

```
overlap_quad(aquad, overlap_fun, title = NULL)
```

Arguments

aquad	a list of four vectors
overlap_fun	the name of the function that defines overlap, depending on the type of object in the vectors. For GRanges, use filter_by_overlaps_stranded or filter_by_nonoverlaps_stranded for gene names, use intersect.
title	main title of the figure

Value

a VennDiagram object

Author(s)

Shuye Pu

Examples

```
test_list <- list(A = c(1, 2, 3, 4, 5), B = c(4, 5, 7), C = c(1, 3), D = 6)
overlap_quad(test_list, intersect)

## GRanges overlap
query1 <- GRanges("chr19",
    IRanges(rep(c(10, 15), 2), width = c(1, 20, 40, 50)),
    strand = c("+", "+", "-", "-")
)

query2 <- GRanges("chr19",
    IRanges(rep(c(1, 15), 2), width = c(1, 20, 40, 50)),
    strand = c("+", "+", "-", "-")
)

subject1 <- GRanges("chr19",
    IRanges(rep(c(13, 150), 2), width = c(10, 14, 20, 28)),
    strand = c("+", "-", "-", "+")
)

subject2 <- GRanges("chr19",
    IRanges(rep(c(13, 50), 2), width = c(10, 14, 20, 21)),
    strand = c("+", "-", "-", "+")
)

overlap_quad(list(
    subject1 = subject1, subject2 = subject2, query1 = query1,
    query2 = query2
), filter_by_overlaps_stranded)
```

`overlap_triple` *Plot three-sets Venn diagram*

Description

This is a helper function for Venn diagram plot. A Venn diagram is plotted as output. For GRanges, as A overlap B may not be the same as B overlap A, the order of GRanges in a list matters, certain order may produce an error.

Usage

```
overlap_triple(atriple, overlap_fun, title = NULL)
```

Arguments

atriple	a list of three vectors
overlap_fun	the name of the function that defines overlap, depending on the type of object in the vectors. For GRanges, use filter_by_overlaps_stranded or filter_by_nonoverlaps_stranded for gene names, use intersect.
title	main title of the figure

Value

a VennDiagram object

Author(s)

Shuye Pu

Examples

```
test_list <- list(A = c(1, 2, 3, 4, 5), B = c(4, 5, 7), C = c(1, 3))
overlap_triple(test_list, intersect, title = "test")

## GRanges overlap
query <- GRanges("chr19",
    IRanges(rep(c(10, 15), 2), width = c(1, 20, 40, 50)),
    strand = c("+", "+", "-", "-")
)

subject1 <- GRanges("chr19",
    IRanges(rep(c(13, 150), 2), width = c(10, 14, 20, 28)),
    strand = c("+", "-", "-", "+")
)

subject2 <- GRanges("chr19",
    IRanges(rep(c(13, 50), 2), width = c(10, 14, 20, 21)),
    strand = c("+", "-", "-", "+")
)

overlap_triple(
    list(subject1 = subject1, subject2 = subject2, query = query),
    filter_by_overlaps_stranded
)
```

parallel_countOverlaps *Parallel execution of countOverlaps*

Description

Function for parallel computation of countOverlaps function in the GenomicRanges package

Usage

```
parallel_countOverlaps(grange_list, tileBins, nc = 2, switch = FALSE)
```

Arguments

grange_list	a list of GRanges objects.
tileBins	a GRanges object of tiled genome
nc	integer, number of cores for parallel processing
switch	logical, switch the order of query and feature

Value

a list of numeric vectors

Author(s)

Shuye Pu

Examples

```
bedQueryFiles <- c(
    system.file("extdata", "test_chip_peak_chr19.narrowPeak",
        package = "GenomicPlot"
    ),
    system.file("extdata", "test_chip_peak_chr19.bed",
        package = "GenomicPlot"),
    system.file("extdata", "test_clip_peak_chr19.bed",
        package = "GenomicPlot")
)
names(bedQueryFiles) <- c("NarrowPeak", "SummitPeak", "iCLIPPeak")

bedimportParams <- setImportParams(
    offset = 0, fix_width = 100, fix_point = "center", norm = FALSE,
    useScore = FALSE, outRle = FALSE, useSizeFactor = FALSE, genome = "hg19"
)

out_list <- handle_input(
    inputFiles = bedQueryFiles,
    importParams = bedimportParams, verbose = TRUE, nc = 2
```

```
)

chromInfo <- circlize::read.chromInfo(species = "hg19")$df
seqi <- Seqinfo(seqnames = chromInfo$chr, seqlengths = chromInfo$end,
               isCircular = rep(FALSE, nrow(chromInfo)),
               genome = "hg19")
grange_list <- lapply(out_list, function(x) x$query)
tilewidth <- 100000
tileBins <- tileGenome(seqi,
    tilewidth = tilewidth,
    cut.last.tile.in.chrom = TRUE
)

score_list1 <- parallel_countOverlaps(grange_list, tileBins, nc = 2)
dplyr::glimpse(score_list1)
```

`parallel_scoreMatrixBin` *Parallel execution of scoreMatrixBin on a huge target windows object split into chunks*

Description

Function for parallel computation of scoreMatrixBin. The 'windows' parameter of the scoreMatrixBin method is split into nc chunks, and scoreMatrixBin is called on each chunk simultaneously to speed up the computation.

Usage

```
parallel_scoreMatrixBin(
  queryRegions,
  windowRs,
  bin_num,
  bin_op,
  weight_col,
  stranded,
  nc = 2
)
```

Arguments

`queryRegions`	a RleList object or Granges object providing input for the 'target' parameter of the scoreMatrixBin method.
`windowRs`	a single GRangesList object.
`bin_num`	number of bins the windows should be divided into
`bin_op`	operation on the signals in a bin, a string in c("mean", "max", "min", "median", "sum") is accepted.

weight_col	if the queryRegions is a GRanges object, a numeric column in meta data part can be used as weights.
stranded	logical, indicating if the strand of the windows should be considered to determine upstream and downstream.
nc	an integer denoting the number of cores requested, 2 is the default number that is allowed by CRAN but 5 gives best trade-off between speed and space.

Value

a numeric matrix

Author(s)

Shuye Pu

Examples

```
queryFiles <- system.file("extdata", "chip_treat_chr19.bam",
    package = "GenomicPlot"
)
names(queryFiles) <- "query"

chipimportParams <- setImportParams(
    offset = 0, fix_width = 150, fix_point = "start", norm = TRUE,
    useScore = FALSE, outRle = TRUE, useSizeFactor = FALSE, genome = "hg19"
)

queryRegion <- handle_input(queryFiles, chipimportParams,
    verbose = TRUE
)[[1]]$query

windowFiles <- system.file("extdata", "test_chip_peak_chr19.narrowPeak",
    package = "GenomicPlot"
)
names(windowFiles) <- "narrowPeak"

importParams <- setImportParams(
    offset = 0, fix_width = 0, fix_point = "start", norm = FALSE,
    useScore = FALSE, outRle = FALSE, useSizeFactor = FALSE, genome = "hg19"
)

windowRegion <- handle_bed(windowFiles, importParams, verbose = TRUE)$query

out <- parallel_scoreMatrixBin(
    queryRegions = queryRegion,
    windowRs = windowRegion,
    bin_num = 50,
    bin_op = "mean",
    weight_col = "score",
    stranded = TRUE,
    nc = 2
```

```
)
#
```

`plot_5parts_metagene` *Plot promoter, 5'UTR, CDS, 3'UTR and TTS*

Description

Plot reads or peak Coverage/base/gene of samples given in the query files around genes. The upstream and downstream windows flanking genes can be given separately, metagene plots are generated with 5'UTR, CDS and 3'UTR segments. The length of each segments are prorated according to the median length of each segments. If Input files are provided, ratio over Input is computed and displayed as well.

Usage

```
plot_5parts_metagene(
  queryFiles,
  gFeatures_list,
  inputFiles = NULL,
  importParams = NULL,
  verbose = FALSE,
  transform = NA,
  smooth = FALSE,
  scale = FALSE,
  stranded = TRUE,
  outPrefix = NULL,
  heatmap = FALSE,
  heatRange = NULL,
  rmOutlier = 0,
  Ylab = "Coverage/base/gene",
  hw = c(10, 10),
  nc = 2
)
```

Arguments

`queryFiles`	a vector of sample file names. The file should be in .bam, .bed, .wig or .bw format, mixture of formats is allowed
`gFeatures_list`	a list of genomic features as output of the function `prepare_5parts_genomic_features`
`inputFiles`	a vector of input sample file names. The file should be in .bam, .bed, .wig or .bw format, mixture of formats is allowed
`importParams`	a list of parameters for `handle_input`
`verbose`	logical, indicating whether to output additional information (data used for plotting or statistical test results)
`transform`	logical, whether to log2 transform the matrix

smooth	logical, indicating whether the line should smoothed with a spline smoothing algorithm
scale	logical, indicating whether the score matrix should be scaled to the range 0:1, so that samples with different baseline can be compared
stranded	logical, indicating whether the strand of the feature should be considered
outPrefix	a string specifying output file prefix for plots (outPrefix.pdf)
heatmap	logical, indicating whether a heatmap of the score matrix should be generated
heatRange	a numeric vector with three elements, defining custom range for color ramp, default=NULL, i.e. the range is defined automatically based on the c(minimun, median, maximum) of a data matrix
rmOutlier	a numeric value serving as a multiplier of the MAD in Hampel filter for outliers identification, 0 indicating not removing outliers. For Gaussian distribution, use 3, adjust based on data distribution.
Ylab	a string for y-axis label
hw	a vector of two elements specifying the height and width of the output figures
nc	integer, number of cores for parallel processing

Value

a dataframe containing the data used for plotting

Author(s)

Shuye Pu

Examples

```
data(gf5_meta)
queryfiles <- system.file("extdata", "treat_chr19.bam",
                          package = "GenomicPlot")
names(queryfiles) <- "clip_bam"
inputfiles <- system.file("extdata", "input_chr19.bam",
                          package = "GenomicPlot")
names(inputfiles) <- "clip_input"

bamimportParams <- setImportParams(
    offset = -1, fix_width = 0, fix_point = "start", norm = TRUE,
    useScore = FALSE, outRle = TRUE, useSizeFactor = FALSE, genome = "hg19"
)

plot_5parts_metagene(
    queryFiles = queryfiles,
    gFeatures_list = list("metagene" = gf5_meta),
    inputFiles = inputfiles,
    scale = FALSE,
    verbose = FALSE,
    transform = NA,
    smooth = TRUE,
```

```
    stranded = TRUE,
    outPrefix = NULL,
    importParams = bamimportParams,
    heatmap = TRUE,
    rmOutlier = 0,
    nc = 2
)
```

`plot_bam_correlation` *Plot correlation of bam files*

Description

Plot correlation in reads coverage distributions along the genome for bam files. Generates a fingerprint plot, a heatmap of correlation coefficients with hierarchical clustering, a pairwise correlation plot and a PCA plot.

Usage

```
plot_bam_correlation(
  bamFiles,
  binSize = 1e+06,
  outPrefix = NULL,
  importParams = NULL,
  grouping = NULL,
  verbose = FALSE,
  hw = c(8, 8),
  nc = 2
)
```

Arguments

`bamFiles`	a named vector of strings denoting file names
`binSize`	an integer denoting the tile width for tiling the genome, default 1000000
`outPrefix`	a string denoting output file name in pdf format
`importParams`	a list of parameters for `handle_input`
`grouping`	a named vector for bamFiles group assignment
`verbose`	logical, indicating whether to output additional information
`hw`	a vector of two elements specifying the height and width of the output figures
`nc`	integer, number of cores for parallel processing

Value

a dataframe of read counts per bin per sample

Examples

```
bamQueryFiles <- c(
    system.file("extdata", "chip_input_chr19.bam", package = "GenomicPlot"),
    system.file("extdata", "chip_treat_chr19.bam", package = "GenomicPlot")
)
grouping <- c(1, 2)
names(bamQueryFiles) <- names(grouping) <- c("chip_input", "chip_treat")

bamImportParams <- setImportParams(
    offset = 0, fix_width = 150, fix_point = "start", norm = FALSE,
    useScore = FALSE, outRle = FALSE, useSizeFactor = FALSE, genome = "hg19"
)

plot_bam_correlation(
    bamFiles = bamQueryFiles, binSize = 100000, outPrefix = NULL,
    importParams = bamImportParams, nc = 2, verbose = FALSE
)
```

`plot_locus` *Plot signal around custom genomic loci*

Description

Plot reads or peak Coverage/base/gene of samples given in the query files around reference locus (start, end or center of a genomic region) defined in the centerFiles. The upstream and downstream windows flanking loci can be given separately, a smaller window can be defined to allow statistical comparisons between samples for the same reference, or between references for a given sample. If Input files are provided, ratio over Input is computed and displayed as well.

Usage

```
plot_locus(
  queryFiles,
  centerFiles,
  txdb = NULL,
  ext = c(-100, 100),
  hl = c(0, 0),
  shade = TRUE,
  smooth = FALSE,
  importParams = NULL,
  verbose = FALSE,
  binSize = 10,
  refPoint = "center",
  Xlab = "Center",
  Ylab = "Coverage/base/gene",
  inputFiles = NULL,
  stranded = TRUE,
```

```
  heatmap = TRUE,
  scale = FALSE,
  outPrefix = NULL,
  rmOutlier = 0,
  transform = NA,
  statsMethod = "wilcox.test",
  heatRange = NULL,
  hw = c(8, 8),
  nc = 2
)
```

Arguments

queryFiles	a vector of sample file names. The file should be in .bam, .bed, .wig or .bw format, mixture of formats is allowed
centerFiles	a named vector of reference file names or genomic features in c("utr3", "utr5", "cds", "intron", "exon", "transcript", "gene"). The file should be in .bed format only
txdb	a TxDb object defined in the GenomicFeatures package. Default NULL, needed only when genomic features are used as centerFiles.
ext	a vector of two integers defining upstream and downstream boundaries of the plot window, flanking the reference locus
hl	a vector of two integers defining upstream and downstream boundaries of the highlight window, flanking the reference locus
shade	logical indicating whether to place a shaded rectangle around the point of interest
smooth	logical, indicating whether the line should smoothed with a spline smoothing algorithm
importParams	a list of parameters for handle_input
verbose	logical, indicating whether to output additional information (data used for plotting or statistical test results)
binSize	an integer defines bin size for intensity calculation
refPoint	a string in c("start", "center", "end")
Xlab	a string denotes the label on x-axis
Ylab	a string for y-axis label
inputFiles	a vector of input sample file names. The file should be in .bam, .bed, .wig or .bw format, mixture of formats is allowed
stranded	logical, indicating whether the strand of the feature should be considered
heatmap	logical, indicating whether a heatmap of the score matrix should be generated
scale	logical, indicating whether the score matrix should be scaled to the range 0:1, so that samples with different baseline can be compared
outPrefix	a string specifying output file prefix for plots (outPrefix.pdf)

rmOutlier	a numeric value serving as a multiplier of the MAD in Hampel filter for outliers identification, 0 indicating not removing outliers. For Gaussian distribution, use 3, adjust based on data distribution.
transform	a string in c("log", "log2", "log10"), default = NA indicating no transformation of data matrix
statsMethod	a string in c("wilcox.test", "t.test"), for pair-wise group comparisons
heatRange	a numeric vector with three elements, defining custom range for color ramp, default=NULL, i.e. the range is defined automatically based on the c(minimun, median, maximum) of a data matrix
hw	a vector of two elements specifying the height and width of the output figures
nc	integer, number of cores for parallel processing

Value

a list of two dataframes containing the data used for plotting and for statistical testing

Author(s)

Shuye Pu

Examples

```
centerfiles <- c(
system.file("extdata", "test_clip_peak_chr19.bed", package = "GenomicPlot"),
system.file("extdata", "test_chip_peak_chr19.bed", package = "GenomicPlot"))

names(centerfiles) <- c("iCLIPPeak", "SummitPeak")
queryfiles <- c(
    system.file("extdata", "chip_treat_chr19.bam", package = "GenomicPlot"))

names(queryfiles) <- c("chip_bam")
inputfiles <- c(
    system.file("extdata", "chip_input_chr19.bam", package = "GenomicPlot"))
names(inputfiles) <- c("chip_input")

chipimportParams <- setImportParams(
    offset = 0, fix_width = 150, fix_point = "start", norm = TRUE,
    useScore = FALSE, outRle = TRUE, useSizeFactor = FALSE, genome = "hg19"
)

plot_locus(
  queryFiles = queryfiles,
  centerFiles = centerfiles,
  ext = c(-500, 500),
  hl = c(-100, 100),
  shade = TRUE,
  smooth = TRUE,
  importParams = chipimportParams,
  binSize = 10,
  refPoint = "center",
```

```
  Xlab = "Center",
  inputFiles = inputfiles,
  stranded = TRUE,
  scale = FALSE,
  outPrefix = NULL,
  verbose = FALSE,
  transform = NA,
  rmOutlier = 0,
  Ylab = "Coverage/base/peak",
  statsMethod = "wilcox.test",
  heatmap = TRUE,
  nc = 2
)
```

`plot_locus_with_random` *Plot signal around custom genomic loci and random loci for comparison*

Description

Plot reads or peak Coverage/base/gene of samples given in the query files around reference locus defined in the centerFiles. The upstream and downstream windows flanking loci can be given separately, a smaller window can be defined to allow statistical comparisons between reference and random loci. The loci are further divided into sub-groups that are overlapping with c("5'UTR", "CDS", "3'UTR"), "unrestricted" means all loci regardless of overlapping.

Usage

```
plot_locus_with_random(
  queryFiles,
  centerFiles,
  txdb,
  ext = c(-200, 200),
  hl = c(-100, 100),
  shade = FALSE,
  importParams = NULL,
  verbose = FALSE,
  smooth = FALSE,
  transform = NA,
  binSize = 10,
  refPoint = "center",
  Xlab = "Center",
  Ylab = "Coverage/base/gene",
  inputFiles = NULL,
  stranded = TRUE,
```

```
  scale = FALSE,
  outPrefix = NULL,
  rmOutlier = 0,
  n_random = 1,
  hw = c(8, 8),
  detailed = FALSE,
  statsMethod = "wilcox.test",
  nc = 2
)
```

Arguments

queryFiles	a vector of sample file names. The file should be in .bam, .bed, .wig or .bw format, mixture of formats is allowed
centerFiles	a vector of reference file names. The file should be .bed format only
txdb	a TxDb object defined in the 'GenomicFeatures' package
ext	a vector of two integers defining upstream and downstream boundaries of the plot window, flanking the reference locus
hl	a vector of two integers defining upstream and downstream boundaries of the highlight window, flanking the reference locus
shade	logical indicating whether to place a shaded rectangle around the point of interest
importParams	a list of parameters for handle_input
verbose	logical, indicating whether to output additional information (data used for plotting or statistical test results)
smooth	logical, indicating whether the line should smoothed with a spline smoothing algorithm
transform	a string in c("log", "log2", "log10"), default = NA i ndicating no transformation of data matrix
binSize	an integer defines bin size for intensity calculation
refPoint	a string in c("start", "center", "end")
Xlab	a string denotes the label on x-axis
Ylab	a string for y-axis label
inputFiles	a vector of input sample file names. The file should be in .bam, .bed, .wig or .bw format, mixture of formats is allowed
stranded	logical, indicating whether the strand of the feature should be considered
scale	logical, indicating whether the score matrix should be scaled to the range 0:1, so that samples with different baseline can be compared
outPrefix	a string specifying output file prefix for plots (outPrefix.pdf)
rmOutlier	a numeric value serving as a multiplier of the MAD in Hampel filter for outliers identification, 0 indicating not removing outliers. For Gaussian distribution, use 3, adjust based on data distribution
n_random	an integer denotes the number of randomization should be performed

hw	a vector of two elements specifying the height and width of the output figures
detailed	logical, indicating whether to plot each parts of gene.
statsMethod	a string in c("wilcox.test", "t.test"), for pair-wise groups comparisons
nc	integer, number of cores for parallel processing

Value

a dataframe containing the data used for plotting

Author(s)

Shuye Pu

Examples

```
gtfFile <- system.file("extdata", "gencode.v19.annotation_chr19.gtf",
    package = "GenomicPlot"
)

txdb <- custom_TxDb_from_GTF(gtfFile, genome = "hg19")
bedQueryFiles <- c(
system.file("extdata", "test_chip_peak_chr19.narrowPeak",
            package = "GenomicPlot"),
system.file("extdata", "test_chip_peak_chr19.bed", package = "GenomicPlot"),
system.file("extdata", "test_clip_peak_chr19.bed", package = "GenomicPlot")
)
names(bedQueryFiles) <- c("NarrowPeak", "SummitPeak", "iCLIPPeak")

bamQueryFiles <- system.file("extdata", "treat_chr19.bam",
                             package = "GenomicPlot")
names(bamQueryFiles) <- "clip_bam"
bamInputFiles <- system.file("extdata", "input_chr19.bam",
                             package = "GenomicPlot")
names(bamInputFiles) <- "clip_input"

bamImportParams <- setImportParams(
  offset = -1, fix_width = 0, fix_point = "start", norm = TRUE,
  useScore = FALSE, outRle = TRUE, useSizeFactor = FALSE, genome = "hg19"
)
plot_locus_with_random(
    queryFiles = bamQueryFiles,
    centerFiles = bedQueryFiles[3],
    txdb = txdb,
    ext = c(-200, 200),
    hl = c(-50, 50),
    shade = TRUE,
    importParams = bamImportParams,
    verbose = FALSE,
    smooth = TRUE,
    transform = NA,
    binSize = 10,
```

```
    refPoint = "center",
    Xlab = "Center",
    Ylab = "Coverage/base/peak",
    inputFiles = bamInputFiles,
    stranded = TRUE,
    scale = FALSE,
    outPrefix = NULL,
    rmOutlier = 0,
    n_random = 1,
    hw = c(8, 8),
    detailed = FALSE,
    statsMethod = "wilcox.test",
    nc = 2)
```

`plot_overlap_bed` *Plot Venn diagrams depicting overlap of genomic regions*

Description

This function takes a list of up to 4 bed file names, and produce a Venn diagram

Usage

```
plot_overlap_bed(
  bedList,
  outPrefix = NULL,
  importParams = NULL,
  pairOnly = TRUE,
  stranded = TRUE,
  hw = c(8, 8),
  verbose = FALSE
)
```

Arguments

`bedList`	a named list of bed files, with list length = 2, 3 or 4
`outPrefix`	a string for plot file name
`importParams`	a list of parameters for `handle_input`
`pairOnly`	logical, indicating whether only pair-wise overlap is desirable
`stranded`	logical, indicating whether the feature is stranded. For nonstranded feature, only "*" is accepted as strand
`hw`	a vector of two elements specifying the height and width of the output figures
`verbose`	logical, indicating whether to output additional information

Value

a ggplot object

Author(s)

Shuye Pu

Examples

```
queryFiles <- c(
    system.file("extdata", "test_chip_peak_chr19.narrowPeak",
        package = "GenomicPlot"
    ),
    system.file("extdata", "test_chip_peak_chr19.bed",
        package = "GenomicPlot"
    ),
    system.file("extdata", "test_clip_peak_chr19.bed",
        package = "GenomicPlot"
    )
)
names(queryFiles) <- c("narrowPeak", "summitPeak", "clipPeak")

bedimportParams <- setImportParams(
    offset = 0, fix_width = 100, fix_point = "center", norm = FALSE,
    useScore = FALSE, outRle = FALSE, useSizeFactor = FALSE, genome = "hg19"
)

plot_overlap_bed(
    bedList = queryFiles, importParams = bedimportParams, pairOnly = FALSE,
    stranded = FALSE, outPrefix = NULL
)
```

---

plot_overlap_genes *Plot Venn diagrams depicting overlap of gene lists*

Description

This function takes a list of (at most 4) tab-delimited file names, and produce a Venn diagram

Usage

```
plot_overlap_genes(
  fileList,
  columnList,
  pairOnly = TRUE,
  hw = c(8, 8),
  outPrefix = NULL
)
```

Arguments

fileList	a named list of tab-delimited files
columnList	a vector of integers denoting the columns that have gene names in the list of files
pairOnly	logical, indicating whether only pair-wise overlap is desirable
hw	a vector of two elements specifying the height and width of the output figures
outPrefix	a string for plot file name

Value

a list of vectors of gene names

Author(s)

Shuye Pu

Examples

```
testfile1 <- system.file("extdata", "test_file1.txt",
    package = "GenomicPlot"
)
testfile2 <- system.file("extdata", "test_file2.txt",
    package = "GenomicPlot"
)
testfile3 <- system.file("extdata", "test_file3.txt",
    package = "GenomicPlot"
)
testfile4 <- system.file("extdata", "test_file4.txt",
    package = "GenomicPlot"
)
testfiles <- c(testfile1, testfile2, testfile3, testfile4)
names(testfiles) <- c("test1", "test2", "test3", "test4")

plot_overlap_genes(testfiles, c(3, 2, 1, 1), pairOnly = FALSE)
```

---

plot_peak_annotation *Annotate peaks with genomic features and genes*

Description

Produce a table of transcripts targeted by peaks, and generate plots for target gene types, and peak distribution in genomic features

Usage

```
plot_peak_annotation(
  peakFile,
  gtfFile,
  importParams = NULL,
  fiveP = -1000,
  dsTSS = 300,
  threeP = 1000,
  simple = FALSE,
  outPrefix = NULL,
  verbose = FALSE,
  hw = c(8, 8),
  nc = 2
)
```

Arguments

peakFile	a string denoting the peak file name, only .bed format is allowed
gtfFile	path to a gene annotation gtf file with gene_biotype field
importParams	a list of parameters for handle_input
fiveP	extension out of the 5' boundary of genes for defining promoter: fiveP TSS + dsTSS
dsTSS	extension downstream of TSS for defining promoter: fiveP TSS + dsTSS
threeP	extension out of the 3' boundary of genes for defining termination region: -0 TTS + threeP
simple	logical, indicating whether 5'UTR, CDS and 3'UTR are annotated in the gtfFile
outPrefix	a string denoting output file name in pdf format
verbose	logical, to indicate whether to write the annotation results to a file
hw	a vector of two elements specifying the height and width of the output figures
nc	number of cores for parallel processing

Value

a list of three dataframes, 'annotation' is the annotation of peaks into gene types, 'stat' is the summary stats for pie chart, 'simplified' is the summary stats excluding intron

Author(s)

Shuye Pu

Examples

```
gtfFile <- system.file("extdata", "gencode.v19.annotation_chr19.gtf",
    package = "GenomicPlot"
)
```

```
centerFile <- system.file("extdata", "test_chip_peak_chr19.bed",
    package = "GenomicPlot"
)
names(centerFile) <- c("summitPeak")

bedimportParams <- setImportParams(
    offset = 0, fix_width = 100, fix_point = "center", norm = FALSE,
    useScore = FALSE, outRle = FALSE, useSizeFactor = FALSE, genome = "hg19"
)

plot_peak_annotation(
    peakFile = centerFile, gtfFile = gtfFile, importParams = bedimportParams,
    fiveP = -2000, dsTSS = 200, threeP = 2000, simple = FALSE
)
```

---

plot_region *Plot signal inside as well as around custom genomic regions*

Description

Plot reads or peak Coverage/base/gene of samples given in the query files inside regions defined in the centerFiles. The upstream and downstream flanking windows can be given separately. If Input files are provided, ratio over Input is computed and displayed as well.

Usage

```
plot_region(
  queryFiles,
  centerFiles,
  txdb = NULL,
  regionName = "region",
  inputFiles = NULL,
  nbins = 100,
  importParams = NULL,
  verbose = FALSE,
  scale = FALSE,
  heatmap = FALSE,
  fiveP = -1000,
  threeP = 1000,
  smooth = FALSE,
  stranded = TRUE,
  transform = NA,
  outPrefix = NULL,
  rmOutlier = 0,
  heatRange = NULL,
  Ylab = "Coverage/base/gene",
  statsMethod = "wilcox.test",
```

```
  hw = c(8, 8),
  nc = 2
)
```

Arguments

queryFiles	a named vector of sample file names. The file should be in .bam, .bed, .wig or .bw format, mixture of formats is allowed
centerFiles	a named vector of reference file names or genomic features in c("utr3", "utr5", "cds", "intron", "exon", "transcript", "gene"). The file should be in .bed format only
txdb	a TxDb object defined in the GenomicFeatures package. Default NULL, needed only when genomic features are used as centerFiles.
regionName	a string specifying the name of the center region in the plots
inputFiles	a named vector of input sample file names. The file should be in .bam, .bed, .wig or .bw format, mixture of formats is allowed
nbins	an integer defines the total number of bins
importParams	a list of parameters for handle_input
verbose	logical, indicating whether to output additional information (data used for plotting or statistical test results)
scale	logical, indicating whether the score matrix should be scaled to the range 0:1, so that samples with different baseline can be compared
heatmap	logical, indicating whether a heatmap of the score matrix should be generated
fiveP	an integer, indicating extension out or inside of the 5' boundary of gene by negative or positive number
threeP	an integer, indicating extension out or inside of the 5' boundary of gene by positive or negative number
smooth	logical, indicating whether the line should smoothed with a spline smoothing algorithm
stranded	logical, indicating whether the strand of the feature should be considered
transform	a string in c("log", "log2", "log10"), default = NA indicating no transformation of data matrix
outPrefix	a string specifying output file prefix for plots (outPrefix.pdf)
rmOutlier	a numeric value serving as a multiplier of the MAD in Hampel filter for outliers identification, 0 indicating not removing outliers. For Gaussian distribution, use 3, adjust based on data distribution
heatRange	a numeric vector with three elements, defining custom range for color ramp, default=NULL, i.e. the range is defined automatically based on the c(minimun, median, maximum) of a data matrix
Ylab	a string for y-axis label
statsMethod	a string in c("wilcox.test", "t.test"), for pair-wise group comparisons
hw	a vector of two elements specifying the height and width of the output figures
nc	integer, number of cores for parallel processing

Value

a dataframe containing the data used for plotting

Author(s)

Shuye Pu

Examples

```
centerfiles <- system.file("extdata", "test_chip_peak_chr19.narrowPeak",
package = "GenomicPlot")
names(centerfiles) <- c("NarrowPeak")
queryfiles <- c(
  system.file("extdata", "chip_treat_chr19.bam", package = "GenomicPlot"))
names(queryfiles) <- c("chip_bam")
inputfiles <- c(
  system.file("extdata", "chip_input_chr19.bam", package = "GenomicPlot"))
names(inputfiles) <- c("chip_input")

chipimportParams <- setImportParams(
  offset = 0, fix_width = 150, fix_point = "start", norm = TRUE,
  useScore = FALSE, outRle = TRUE, useSizeFactor = FALSE, genome = "hg19",
  chr = c("chr19"))

plot_region(
  queryFiles = queryfiles,
  centerFiles = centerfiles,
  inputFiles = inputfiles,
  nbins = 100,
  heatmap = TRUE,
  scale = FALSE,
  regionName = "narrowPeak",
  importParams = chipimportParams,
  verbose = FALSE,
  fiveP = -500,
  threeP = 500,
  smooth = TRUE,
  transform = "log2",
  stranded = TRUE,
  outPrefix = NULL,
  Ylab = "Coverage/base/peak",
  rmOutlier = 0,
  nc = 2
)
```

---

plot_start_end *Plot signals around the start and the end of genomic features*

Description

Plot reads or peak Coverage/base/gene of samples given in the query files around start and end of custom features. The upstream and downstream windows can be given separately, within the window, a smaller window can be defined to highlight region of interest. A line plot will be displayed for both start and end of feature. If Input files are provided, ratio over Input is computed and displayed as well.

Usage

```
plot_start_end(
  queryFiles,
  inputFiles = NULL,
  centerFiles,
  txdb = NULL,
  importParams = NULL,
  binSize = 10,
  insert = 0,
  verbose = FALSE,
  ext = c(-500, 100, -100, 500),
  hl = c(-50, 50, -50, 50),
  stranded = TRUE,
  scale = FALSE,
  smooth = FALSE,
  rmOutlier = 0,
  outPrefix = NULL,
  transform = NA,
  shade = TRUE,
  Ylab = "Coverage/base/gene",
  hw = c(8, 8),
  nc = 2
)
```

Arguments

queryFiles	a vector of sample file names. The file should be in .bam, .bed, .wig or .bw format, mixture of formats is allowed
inputFiles	a vector of input sample file names. The file should be in .bam, .bed, .wig or .bw format, mixture of formats is allowed
centerFiles	bed files that define the custom features, or features in c("utr3", "utr5", "cds", "intron", "exon", "transcript", "gene"), multiple features are allowed.
txdb	a TxDb object defined in the GenomicFeatures package. Default NULL, needed only when genomic features are used in the place of centerFiles.
importParams	a list of parameters for handle_input
binSize	an integer defines bin size for intensity calculation
insert	an integer specifies the length of the center regions to be included, in addition to the start and end of the feature

verbose	logical, whether to output additional information (including data used for plotting or statistical test results)
ext	a vector of four integers defining upstream and downstream boundaries of the plot window, flanking the start and end of features
hl	a vector of four integers defining upstream and downstream boundaries of the highlight window, flanking the start and end of features
stranded	logical, indicating whether the strand of the feature should be considered
scale	logical, indicating whether the score matrix should be scaled to the range 0:1, so that samples with different baseline can be compared
smooth	logical, indicating whether the line should smoothed with a spline smoothing algorithm
rmOutlier	a numeric value serving as a multiplier of the MAD in Hampel filter for outliers identification, 0 indicating not removing outliers. For Gaussian distribution, use 3, adjust based on data distribution
outPrefix	a string specifying output file prefix for plots (outPrefix.pdf)
transform	a string in c("log", "log2", "log10"), default = NA, indicating no transformation of data matrix
shade	logical indicating whether to place a shaded rectangle around the point of interest
Ylab	a string for y-axis label
hw	a vector of two elements specifying the height and width of the output figures
nc	integer, number of cores for parallel processing

Value

a list of two objects, the first is a GRanges object, the second is a GRangesList object

Author(s)

Shuye Pu

Examples

```
gtfFile <- system.file("extdata", "gencode.v19.annotation_chr19.gtf",
    package = "GenomicPlot"
)

txdb <- custom_TxDb_from_GTF(gtfFile, genome = "hg19")
bamQueryFiles <- system.file("extdata", "treat_chr19.bam",
    package = "GenomicPlot")
names(bamQueryFiles) <- "clip_bam"
bamInputFiles <- system.file("extdata", "input_chr19.bam",
                             package = "GenomicPlot")
names(bamInputFiles) <- "clip_input"

bamimportParams <- setImportParams(
    offset = -1, fix_width = 0, fix_point = "start", norm = TRUE,
```

```
    useScore = FALSE, outRle = TRUE, useSizeFactor = FALSE, genome = "hg19"
)

plot_start_end(
    queryFiles = bamQueryFiles,
    inputFiles = bamInputFiles,
    txdb = txdb,
    centerFiles = "intron",
    binSize = 10,
    importParams = bamimportParams,
    ext = c(-500, 200, -200, 500),
    hl = c(-100, 100, -100, 100),
    insert = 100,
    stranded = TRUE,
    scale = FALSE,
    smooth = TRUE,
    transform = "log2",
    outPrefix = NULL,
    nc = 2
)
```

`plot_start_end_with_random` *Plot signals around the start and the end of genomic features and random regions*

Description

Plot reads or peak Coverage/base/gene of samples given in the query files around start, end and center of genomic features or custom feature given in a .bed file. The upstream and downstream windows can be given separately. If Input files are provided, ratio over Input is computed and displayed as well. A random feature can be generated to serve as a background for contrasting.

Usage

```
plot_start_end_with_random(
  queryFiles,
  inputFiles = NULL,
  txdb = NULL,
  centerFile,
  importParams = NULL,
  binSize = 10,
  insert = 0,
  verbose = FALSE,
  ext = c(-500, 200, -200, 500),
  hl = c(-50, 50, -50, 50),
  randomize = FALSE,
  stranded = TRUE,
```

```
  scale = FALSE,
  smooth = FALSE,
  rmOutlier = 0,
  outPrefix = NULL,
  transform = NA,
  shade = TRUE,
  nc = 2,
  hw = c(8, 8),
  Ylab = "Coverage/base/gene"
)
```

Arguments

queryFiles	a vector of sample file names. The file should be in .bam, .bed, .wig or .bw format, mixture of formats is allowed
inputFiles	a vector of input sample file names. The file should be in .bam, .bed, .wig or .bw format, mixture of formats is allowed
txdb	a TxDb object defined in the GenomicFeatures package. Default NULL, needed only when genomic features are used in the place of centerFile.
centerFile	a bed file that defines the custom feature, or a feature in c("utr3", "utr5", "cds", "intron", "exon", "transcript", "gene"), multiple features are not allowed.
importParams	a list of parameters for handle_input
binSize	an integer defines bin size for intensity calculation
insert	an integer specifies the length of the center regions to be included, in addition to the start and end of the feature
verbose	logical, whether to output additional information (data used for plotting or statistical test results)
ext	a vector of four integers defining upstream and downstream boundaries of the plot window, flanking the start and end of features
hl	a vector of four integers defining upstream and downstream boundaries of the highlight window, flanking the start and end of features
randomize	logical, indicating if randomized feature should generated and used as a contrast to the real feature. The ransomized feature is generated by shifting the given feature with a random offset within the range of ext[1] and ext[4]
stranded	logical, indicating whether the strand of the feature s hould be considered
scale	logical, indicating whether the score matrix should be scaled to the range 0:1, so that samples with different baseline can be compared
smooth	logical, indicating whether the line should smoothed with a spline smoothing algorithm
rmOutlier	a numeric value serving as a multiplier of the MAD in Hampel filter for outliers identification, 0 indicating not removing outliers. For Gaussian distribution, use 3, adjust based on data distribution
outPrefix	a string specifying output file prefix for plots (outPrefix.pdf)

transform	a string in c("log", "log2", "log10"), default = NA indicating no transformation of data matrix
shade	logical indicating whether to place a shaded rectangle around the point of interest
nc	integer, number of cores for parallel processing
hw	a vector of two elements specifying the height and width of the output figures
Ylab	a string for y-axis label

Value

a list of two objects, the first is a GRanges object, the second is a GRangesList object

Author(s)

Shuye Pu

Examples

```
gtfFile <- system.file("extdata", "gencode.v19.annotation_chr19.gtf",
    package = "GenomicPlot"
)

txdb <- custom_TxDb_from_GTF(gtfFile, genome = "hg19")

bamQueryFiles <- system.file("extdata", "treat_chr19.bam",
                             package = "GenomicPlot")
names(bamQueryFiles) <- "clip_bam"
bamInputFiles <- system.file("extdata", "input_chr19.bam",
                             package = "GenomicPlot")
names(bamInputFiles) <- "clip_input"

bamImportParams <- setImportParams(
  offset = -1, fix_width = 0, fix_point = "start", norm = TRUE,
  useScore = FALSE, outRle = TRUE, useSizeFactor = FALSE, genome = "hg19"
)

plot_start_end_with_random(
  queryFiles = bamQueryFiles,
  inputFiles = bamInputFiles,
  txdb = txdb,
  centerFile = "intron",
  binSize = 10,
  importParams = bamImportParams,
  ext = c(-100, 100, -100, 100),
  hl = c(-20, 20, -20, 20),
  insert = 100,
  stranded = TRUE,
  scale = FALSE,
  smooth = TRUE,
  verbose = TRUE,
  transform = "log2",
```

```
  outPrefix = NULL,
  randomize = TRUE,
  nc = 2
)
```

`prepare_3parts_genomic_features` *Demarcate genes into promoter, gene body and TTS features*

Description

This is a helper function for 'plot_3parts_metagene', used to speed up plotting of multiple data sets with the same configuration. Use featureName='transcript' and meta=FALSE and longest=TRUE for genes.

Usage

```
prepare_3parts_genomic_features(
  txdb,
  featureName = "transcript",
  meta = TRUE,
  nbins = 100,
  fiveP = -1000,
  threeP = 1000,
  longest = TRUE,
  protein_coding = TRUE,
  verbose = FALSE
)
```

Arguments

`txdb`	a TxDb object defined in the GenomicFeatures package
`featureName`	one of the gene feature in c("utr3", "utr5", "cds", "transcript")
`meta`	logical, indicating whether a metagene (intron excluded) or genomic (intron included) plot should be produced
`nbins`	an integer defines the total number of bins
`fiveP`	extension out of the 5' boundary of gene
`threeP`	extension out of the 3' boundary of gene
`longest`	logical, indicating whether the output should be limited to the longest transcript of each gene
`protein_coding`	logical, indicating whether to limit to protein_coding genes
`verbose`	logical, whether to output additional information

Value

a named list with the elements c("windowRs", "nbins", "scaled_bins", "fiveP", "threeP", "meta", "longest")

Author(s)

Shuye Pu

Examples

```
gtfFile <- system.file("extdata", "gencode.v19.annotation_chr19.gtf",
    package = "GenomicPlot"
)

txdb <- custom_TxDb_from_GTF(gtfFile, genome = "hg19")

gf <- prepare_3parts_genomic_features(txdb,
    meta = FALSE, nbins = 100, fiveP = -1000, threeP = 1000,
    longest = FALSE
)
```

---

prepare_5parts_genomic_features

Demarcate genes into promoter, 5'UTR, CDS, 3'UTR and TTS features

---

Description

This is a helper function for 'plot_5parts_metagene', used to speed up plotting of multiple data sets with the same configuration. Only protein-coding genes are considered.

Usage

```
prepare_5parts_genomic_features(
  txdb,
  meta = TRUE,
  nbins = 100,
  fiveP = -1000,
  threeP = 1000,
  longest = TRUE,
  verbose = FALSE,
  subsetTx = NULL
)
```

Arguments

txdb	a TxDb object defined in the GenomicFeatures package
meta	logical, indicating whether a metagene (intron excluded) or gene (intron included) plot should be produced
nbins	an integer defines the total number of bins
fiveP	extension out of the 5' boundary of gene
threeP	extension out of the 3' boundary of gene
longest	logical, indicating whether the output should be limited to the longest transcript of each gene
verbose	logical, whether to output additional information
subsetTx	a vector of transcript names (eg. ENST00000587541.1) for subsetting the genome

Value

a named list with the elements c("windowRs", "nbins", "scaled_bins", "fiveP", "threeP", "meta", "longest")

Author(s)

Shuye Pu

Examples

```
gtfFile <- system.file("extdata", "gencode.v19.annotation_chr19.gtf",
    package = "GenomicPlot"
)

txdb <- custom_TxDb_from_GTF(gtfFile, genome = "hg19")

gf <- prepare_5parts_genomic_features(txdb,
    meta = TRUE, nbins = 100, fiveP = -0, threeP = 0,
    longest = TRUE
)
```

---

process_scoreMatrix *Preprocess scoreMatrix before plotting*

Description

This is a helper function for manipulate the score matrix produced by ScoreMatrix or ScoreMatrinBin functions defined in the 'genomation' package. To facilitate downstream analysis, imputation of missing values is performed implicitly when log transformation is required, otherwise missing values are replaced with 0.

Usage

```
process_scoreMatrix(
  fullmatrix,
  scale = FALSE,
  rmOutlier = 0,
  transform = NA,
  verbose = FALSE
)
```

Arguments

fullmatrix	a numeric matrix, with bins in columns and genomic windows in rows
scale	logical, indicating whether the score matrix should be scaled to the range 0:1, so that samples with different baseline can be compared
rmOutlier	a numeric value to multiple the 'mad' when detecting outliers, can be adjusted based on data. Default 0, indicating not to remove outliers.
transform	a string in c("log", "log2", "log10"), default = NA indicating no transformation of data matrix
verbose	logical, indicating whether to output additional information (data used for plotting or statistical test results)

Details

If inputFiles for the plotting function is null, all operations (scale, rmOutlier and transform) can be applied to the score matrix, in the order of rmOutlier -> transform -> scale. When inputFiles are provided, only rmOutlier can be applied to the score matrix, as transform and scale will affect ratio calculation, especially when log2 transformation of the ratio is intended. However, all these operations can be applied to the resulting ratio matrix. In order to avoid introducing distortion into the processed data, use caution when applying these operations.

Value

a numeric matrix with the same dimension as the fullmatrix

Author(s)

Shuye Pu

Examples

```
fullMatrix <- matrix(rlnorm(100), ncol = 10)
for (i in 5:6) {
    fullMatrix[i, 4:7] <- NaN
    fullMatrix[i + 1, 4:7] <- NA
    fullMatrix[i + 2, 4:7] <- -Inf
    fullMatrix[i - 1, 4:7] <- 0
    fullMatrix[i - 2, 1:3] <- Inf
}
fullMatrix[9, 4:7] <- runif(4) + 90
```

```
wo <- process_scoreMatrix(fullMatrix, rmOutlier = 3, verbose = TRUE)
tf <- process_scoreMatrix(fullMatrix,
    rmOutlier = 0, transform = "log2", verbose = TRUE
)
scaled <- process_scoreMatrix(fullMatrix, scale = TRUE, verbose = TRUE)
```

`rank_rows` *Rank rows of a matrix based on user input*

Description

The rows of a input numeric matrix is ordered based row sum, row maximum, or hierarchical clustering of the rows with euclidean distance and centroid linkage. This a helper function for drawing matrix heatmaps.

Usage

```
rank_rows(fullmatrix, ranking = "Hierarchical")
```

Arguments

`fullmatrix`	a numeric matrix
`ranking`	a string in c("Sum", "Max", "Hierarchical", "None")

Value

a numeric matrix

Author(s)

Shuye Pu

Examples

```
fullMatrix <- matrix(rnorm(100), ncol = 10)
for (i in 5:8) {
    fullMatrix[i, 4:7] <- runif(4) + i
}
apply(fullMatrix, 1, sum)
ranked <- rank_rows(fullMatrix, ranking = "Sum")
apply(ranked, 1, sum)
```

`rm_outlier` *Remove outliers from scoreMatrix*

Description

This is a helper function for dealing with excessively high values using Hampel filter. If outliers are detected, replace the outliers with the up bound = median(rowmax) + multiplier*mad(rowmax). This function is experimental. For data with normal distribution, the multiplier is usually set at 3. As the read counts data distribution is highly skewed, it is difficult to define a boundary for outliers, try the multiplier values between 10 to 1000.

Usage

```
rm_outlier(fullmatrix, verbose = FALSE, multiplier = 1000)
```

Arguments

`fullmatrix`	a numeric matrix, with bins in columns and genomic windows in rows
`verbose`	logical, whether to output the outlier information to the console
`multiplier`	a numeric value to multiple the 'mad', default 1000, maybe adjusted based on data

Value

a numeric matrix

Author(s)

Shuye Pu

Examples

```
fullmatrix <- matrix(rnorm(100), ncol = 10)
maxm <- max(fullmatrix)
fullmatrix[3, 9] <- maxm + 1000
fullmatrix[8, 1] <- maxm + 500
rm_outlier(fullmatrix, verbose = TRUE, multiplier = 100)
rm_outlier(fullmatrix, verbose = TRUE, multiplier = 1000)
```

setImportParams *set parameters for* handle_input *function*

Description

This function save as a template for setting up import parameters for reading NGS data, it provides default values for each parameter.

Usage

```
setImportParams(
  offset = 0,
  fix_width = 0,
  fix_point = "start",
  norm = FALSE,
  useScore = FALSE,
  outRle = TRUE,
  useSizeFactor = FALSE,
  saveRds = FALSE,
  genome = "hg19",
  val = 4,
  skip = 0,
  chr = NULL
)
```

Arguments

offset	an integer, -1 indicating the bam reads should be shrunk to the -1 position at the 5'end of the reads, which corresponds to the cross link site in iCLIP.
fix_width	an integer, for bam file, defines how long the reads should be extended from the start position, ignored when offset is not 0; for bed files, defines the width of each interval centering on the 'fix_point'.
fix_point	a string in c("start", "end", "center") denoting the anchor point for extension, ignored when offset is not 0.
norm	logical, indicating whether the output RleList should be normalized to RPM using library sizes.
useScore	logical, indicating whether the 'score' column of the bed file should be used in calculation of coverage.
outRle	logical, indicating whether the output should be RleList objects or GRanges objects.
useSizeFactor	logical, indicating whether the library size should be adjusted with a size factor, using the 'calcNormFactors' function in the edgeR package, only applicable to ChIPseq data.
saveRds	logical, indicating whether the results of handle_input should be saved for fast reloading

genome	a string denoting the genome name and version.
val	integer, indicating the column that will be used as score/value. default 4 for bedGraph.
skip	integer, indicating how many rows will be skipped before reading in data, default 0.
chr	a vector of string, denoting chromosomes to be included, like c("chr1", "chr2", "chrX"), default NULL indicating all chromosomes will be included.

Value

a list of nine elements

Author(s)

Shuye Pu

Examples

```
importParams1 <- setImportParams()
importParams2 <- setImportParams(offset = -1, saveRds = TRUE)
```

---

set_seqinfo *Set standard chromosome size of model organisms*

Description

This is a helper function for making Seqinfo objects, which is a components of GRanges and TxDb objects. It also serves to unify seqlevels between GRanges and TxDb objects. Mitochondrial chromosome is not included.

Usage

```
set_seqinfo(genome = "hg19")
```

Arguments

genome	a string denoting the genome name and version

Value

a Seqinfo object defined in the GenomeInfoDb package.

Author(s)

Shuye Pu

Examples

```
out <- set_seqinfo(genome = "hg19")
```

`start_parallel` *Prepare for parallel processing*

Description

Creating a virtual cluster for parallel processing

Usage

```
start_parallel(nc = 2, verbose = FALSE)
```

Arguments

`nc`	a positive integer greater than 1, denoting number of cores requested
`verbose`	logical, whether to output additional information

Value

an object of class c("SOCKcluster", "cluster"), depending on platform

Author(s)

Shuye Pu

Examples

```
cl <- start_parallel(2L)
stop_parallel(cl)
```

`stop_parallel` *Stop parallel processing*

Description

Stopping a virtual cluster after parallel processing is finished

Usage

```
stop_parallel(cl)
```

Arguments

cl	a cluster or SOCKcluster object depending on platform

Value

0 if the cluster is stopped successfully, 1 otherwise.

Author(s)

Shuye Pu

Examples

```
cl <- start_parallel(2L)
stop_parallel(cl)
```

---

txdb.sql *Toy data for examples and testing of the 'GenomicPlot' package*

Description

A tiny TxDb object holding genomic feature coordinates of 72 transcripts in hg19.

Value

A SQLlite database

Author(s)

Shuye Pu

Source

The data is produced by running the following code:
gtffile <- system.file("extdata", "gencode.v19.annotation_chr19.gtf", package = "GenomicPlot")
txdb <- custom_TxDb_from_GTF(gtffile, genome = "hg19")
AnnotationDbi::saveDb(txdb, "./inst/extdata/txdb.sql")

Index