Package ‘primaryTranscriptAnnotation’

July 18, 2019

Type Package

Title Data-driven gene coordinate inference and assignment of annotations to transcriptional units identified de novo

Version 0.1.0

Author Warren Anderson, Mete Civelek, Michael Guertin

Maintainer Warren Anderson <warrena@virginia.edu>

Description This package was developed for two purposes: (1) to generate data-driven transcript annotations based on nascent transcript sequencing data, and (2) to annotate de novo identified transcriptional units (e.g., genes and enhancers) based on existing annotations such as those from (1). The code for this package was employed in analyses underlying our manuscript in preparation: Transcriptional mechanisms and gene regulatory networks underlying early adipogenesis (this note will be updated upon publication).

License NA

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Imports bigWig, pracma, dplyr, RColorBrewer, NMF

R topics documented:

`adjacent.coords.plot` *Plot adjacent gene pair coordinates*

Description

Plot the coordinates, along with read data, for overlapping genes that were identified as an adjacent pair.

Usage

```
adjacent.coords.plot(adjacent.coords = NULL, pair = NULL,
  bw.plus = NULL, bw.minus = NULL, bp.bin = 5, xper = 0.2,
  yper = 1.3)
```

Arguments

`adjacent.coords`	bed6 frame with coordinates for adjacent genes
`pair`	frame with one row, upstream and downstream genes
`bw.plus`	plus strand bigWig data
`bw.minus`	minus strand bigWig data
`xper`	fraction of the x-axis before the gene starts
`yper`	fraction of the y-axis with open space above the read data

Value

A plot with read data and gene coordinate annotations.

Examples

```
# visualize coordinates for a specific pair
adjacent.coords.plot(adjacent.coords=adjacent.coords,
                pair=fix.genes[4,],
                bw.plus=bw.plus, bw.minus=bw.minus)
```

---

adjacent.gene.coords   *Function to identify coordinates for adjacent gene pairs*

Description

We have identified adjacent gene pairs with overlaps based on manual analyses. We empirically define TSSs for these genes by binning the region spanned by both genes, fitting smooth spline curves to the binned read counts, and identifying the two largest peaks separated by a specified distance. We set a bin size and apply the constraint that the identified 'pause' peaks must be a given distance apart. The search region includes a specified distance upstream of the upstream-most gene. We shift the identified peaks upstream. For the spline fits, we set the number of knots to the number of bins divided by specified setting. We identify the TSSs as the exon 1 coordinates closest to the pasue site peaks. We identify the TTS of the upstream gene as an interval from the TSS of the downstream gene. The input TTS for the downstream gene is retained.

Usage

```
adjacent.gene.coords(fix.genes = NULL, bed.long = NULL, exon1 = NULL,
  bw.plus = NULL, bw.minus = NULL, bp.bin = NULL, shift.up = NULL,
  dist.from.start = 50, delta.tss = NULL, knot.div = 4,
  knot.thresh = 5, diff.tss = 1000, pause.bp = 120,
  fname = "adjacentSplines.pdf")
```

Arguments

`fix.genes`	frame with upstream and downstream genes
`bed.long`	long gene annotations, see get.largest.interval()
`exon1`	bed6 frame with first exon gene annotations
`bw.plus`	plus strand bigWig data
`bw.minus`	minus strand bigWig data
`bp.bin`	the interval will be separated into adjacent bins of this size
`shift.up`	look upstream of the long gene start by this amount to search for a viable TSS
`dist.from.start`	distance between the TTS of the upstream gene and the TSS of the downstream gene
`delta.tss`	amount by which to shift the identied TSS upstream of the identified interval
`knot.div`	the number of knots for the spline fit is defined as the number of bins covering the gene end divided by this number

`knot.thresh`	minimum number of knots
`diff.tss`	minimum distance between TSSs
`pause.bp`	number of bp by which a pause site should be considered downstrean of a TSS
`fname`	file name (.pdf) for output plots

Value

A frame with the adjusted coordinates for the input gene set, along with a plot. The titles for the plots indicate the upstream gene, the downstream gene, and the strand. For genes on the plus strand, the upstream gene is on the left. For the minus strand, the upstream gene is on the right. The red line denotes the spline fit and the vertical lines indicate the TSSs. The plot is intended for diagnostic purposes.

Examples

```
# get the start sites for adjacent gene pairs
fix.genes = rbind(fix.genes.id, fix.genes.ov)
bp.bin = 5
knot.div = 40
shift.up = 100
delta.tss = 50
diff.tss = 1000
dist.from.start  = 50
adjacent.coords = adjacent.gene.coords(fix.genes=fix.genes, bed.long=TSS.gene,
                                       exon1=gencode.firstExon,
                                       bw.plus=bw.plus, bw.minus=bw.minus,
                                       knot.div=knot.div, bp.bin=bp.bin,
                                       shift.up=shift.up, delta.tss=delta.tss,
                                       dist.from.start=dist.from.start,
                                       diff.tss=diff.tss, fname="adjacentSplines.pdf")
```

`apply.TSS.coords` *Apply previously identified TSSs to an existing gene annotation*

Description

This function will apply transcription start site (TSS) coordinates to a bed6 frame. This function is embedded inside get.TSS().

Usage

```
apply.TSS.coords(bed = NULL, tss = NULL)
```

Arguments

`bed`	bed6 file with comprehensive gene annotations
`tss`	TSS estimates, output from get.TSS(). Note that the TSS genes must be a subset of the genes in the bed6 data.

Value

A bed6 file with estimated TSSs incorporated

Examples

```
out = apply.TSS.coords(bed=bed.out, tss=tss)
```

---

chrom.size.filter *Filter identified gene coordinates for chromosome sizes.*

Description

First acquire a chrom.sizes file (e.g., http://hgdownload.cse.ucsc.edu/goldenPath/mm10/bigZips/mm10.chrom.sizes; see vignette). If any ends are greater than the species chromosome size, reduce the end to the size limit. Any negative starts will be set to 0.

Usage

```
chrom.size.filter(coords = NULL, chrom.sizes = NULL)
```

Arguments

`coords`	bed6 frame with inferred coordinates
`chrom.sizes`	two column file with chromosomes and their respective sizes in bp

Value

A list with a bed6 frame with corrected coordinates (bed) and a log documenting which corrections, if any, were made (log).

Examples

```
coords.filt = chrom.size.filter(coords=coords, chrom.sizes=chrom.sizes)
head(coords.filt$bed)
coords.filt$log
```

eval.tss *Evaluate TSS inference performance*

Description

This function was designed to evaluate the results of out TSS identification analysis. The user should input inferred and largest interval coordinates (bed1 and bed2, respectively). We specify a region centered on the TSS. We obtain read counts in bins that span this window. We sort the genes based on the bin with the maximal reads and we scale the data to the interval (0,1) for visualization. We compute the distances between the TSS and the bin with the max reads within the specified window. Note that this function calls TSS.count.dist() for the TSS distance analysis.

Usage

```
eval.tss(bed1 = NULL, bed2 = NULL, bw.plus = NULL, bw.minus = NULL,
  window = NULL, bp.bin = NULL, fname = NULL)
```

Arguments

`bed1`	bed frame with inferred TSSs
`bed2`	bed frame with reference TSSs
`bw.plus`	plus strand bigWig data
`bw.minus`	minus strand bigWig data
`window`	region size, centered on the TSS for analysis
`bp.bin`	the interval will be separated into adjacent bins of this size
`fname`	.pdf file name for output plots (if NULL, no plot)

Value

A list of two lists and a plot (optional, specifiy fname for the plot to be printed to file). Each list contains three vector elements: raw, scaled, and dist. $tss.dists.inf cooresponds to the TSS data (bed1) and $tss.dists.lng cooresponds to the largest interval data (bed2). All outputs are organized based on TSS position (left-upstream). raw = binned raw counts. scaled = binned scaled counts (0,1), with genes sorted based on the distance between the TSS and the maax read interval. dist = upper bound distances (min(|dists|) = bp.bin). See examples and vignette for more details. The first row of plots includes dirstibutions of distances between the TSS and the maximal reads for the inferred TSSs (left) and largest interval TSSs (right). The second row of plots shows the relationship between the distances from TSSs to max reads and the read depth in the region of read count evaluation (left inferred, right largest interval). The heatmaps show read profiles (horizontal axis, distances centered in the middlr of the window; vertical axis, distinct genes). The genes are sorted based on the inferred TSSs to the max reads (left), and the largest interval distances (right) are organized based on the gene order for the inferred TSSs.

Examples

```
#
```

`gene.end.plot` *Plot the results of transcription terminination site identification*

Description

This function can be used to evaluate the performance of get.gene.end() and get.TTS(). We also recommend visualizing the results of data-driven gene annotations using a genome browser.

Usage

```
gene.end.plot(bed = NULL, gene = NULL, bw.plus = NULL,
  bw.minus = NULL, bp.bin = NULL, pk.thresh = 0.1, knot.div = 40,
  knot.thresh = 5, cnt.thresh = 5, add.to.end = 50000,
  tau.dist = 10000, frac.max = 1, frac.min = 0.2)
```

Arguments

`bed`	bed6 gene coordinates with gene end intervals for estimation of the TTS
`gene`	a specific gene for plotting
`bw.plus`	plus strand bigWig data
`bw.minus`	minus strand bigWig data
`bp.bin`	the interval at the gene end will be separated into adjacent bins of this size
`pk.thresh`	the TTS is defined as pk.thresh percent of max peak of the spline fit
`knot.div`	the number of knots for the spline fit is defined as the number of bins covering the gene end divided by this number; increasing this parameter results and a smoother curve
`knot.thresh`	minimum number of knots, if knot.div gives a lower number, use this
`cnt.thresh`	read count number below which the TTS is not evaluated
`add.to.end`	the maximal length of the search region (bp)
`tau.dist`	distance constant for the exponential defining the region for peak detection
`frac.max`	maximal fraction of gene end region for peal detection
`frac.min`	minimal fraction of gene end region for peal detection

Value

A single plot is generated.

Examples

```
gene.end.plot(bed=bed.for.tts.eval, gene="Aamp",
              bw.plus=bw.plus, bw.minus=bw.minus,
              bp.bin=bp.bin, add.to.end=add.to.end, knot.div=knot.div,
              pk.thresh=pk.thresh, knot.thresh=knot.thresh,
              cnt.thresh=cnt.thresh, tau.dist=tau.dist,
              frac.max=frac.max, frac.min=frac.min)
```

`gene.overlaps` *Documentation of gene overlaps*

Description

This function identifies gene overlapss.

Usage

```
gene.overlaps(bed = NULL)
```

Arguments

`bed`	bed6 frame with processed gene annotations

Value

A list with has.start.inside and is.a.start.inside. has.start.inside = bed6 file that documents genes in which there are starts of other genes. is.a.start.inside = bed6 file that documents genes that start inside other genes.

Examples

```
# run overlap analysis
overlap.data = gene.overlaps( bed = bed.long.filtered2.tss )
has.start.inside = overlap.data$has.start.inside
is.a.start.inside = overlap.data$is.a.start.inside
dim(has.start.inside)
dim(is.a.start.inside)
```

`get.dups` *Get duplicates in which start sites match or end sites match*

Description

This function identifies an extreme for of gene overlaps in which multiple identifiers correspond to identical start or end coordinates.

Usage

```
get.dups(bed = NULL)
```

Arguments

`bed`	a bed file

Value

A bed file with duplicate entries. Col 7 indicates ordered overlap loci.

Examples

```
dups0 = get.dups(bed = TSS.gene)
```

`get.end.intervals` *Get intervals at the gene ends for transcription termination site (TTS) estimation*

Description

We define regions at the gene ends to examine read counts for TTS identification. Note that transcription frequently extends beyond the poly-A site of a gene. To capture the end of transcription, it is critical to examine regions beyond annotated gene boundries. We evaluate evidence of transcriptional termination in regions extending from a 3' subset of the gene to a selected number of base pairs downstream of the most distal annotated gene end. We initiate the search region with the end of the largest gene annotation (see get.largest.interval()). We extend the search region up to a given distance past the annotated gene end. We also apply the constraint that a TTS cannot be identified closer than a specified distance to the previously identified TSS of a downstream gene. This analysis also incorporates the constraint that a gene end region identified cannot cross the TSS of a downstream gene, thereby preventing gene overlaps on a given strand. Thus, we clip the amount of bases added on to the gene end as necessary to avoid overlaps.

Usage

```
get.end.intervals(bed = NULL, add.to.end = NULL, fraction.end = NULL,
  dist.from.start = NULL)
```

Arguments

`bed`	processed bed6 file with one entry per gene and identified TSSs
`add.to.end`	number of bases to add to the end of each gene to define the search region
`fraction.end`	fraction of the gene annotation to consider at the end of the gene
`dist.from.start`	the maximal allowable distance between the end of one gene and the start of the next

Value

A bed6 for gene end evaluation.

Examples

```
# get intervals for TTS evaluation
add.to.end = 100000
fraction.end=0.1
dist.from.start=50
bed.for.tts.eval = get.end.intervals(bed=bed.long.filtered4.tss,
                                     add.to.end=add.to.end,
                                     fraction.end=fraction.end,
                                     dist.from.start=dist.from.start)
```

---

get.gene.end *Estimate the transcription termination sites (TTSs)*

Description

This function estimates the TTSs, this function is called by get.gene.end() and is not designed to be run on its own.

Usage

```
get.gene.end(bed = NULL, bw = NULL, bp.bin = NULL,
  pk.thresh = NULL, knot.div = NULL, cnt.thresh = NULL,
  knot.thresh = NULL, add.to.end = NULL, tau.dist = NULL,
  frac.max = NULL, frac.min = NULL, sum.thresh = NULL)
```

Arguments

`bed`	bed6 gene coordinates with gene end intervals for estimation of the TTS
`bw`	plus or minus strand bigWig data
`bp.bin`	the interval at the gene end will be separated into adjacent bins of this size
`pk.thresh`	the TTS is defined as pk.thresh percent of max peak of the spline fit
`knot.div`	the number of knots for the spline fit is defined as the number of bins covering the gene end divided by this number
`cnt.thresh`	read count number below which the TTS is not evaluated
`knot.thresh`	minimum number of knots, if knot.div gives a lower number, use this
`add.to.end`	the maximal length of the search region (bp)
`tau.dist`	distance constant for the exponential defining the region for peak detection
`frac.max`	maximal fraction of gene end region for peal detection
`frac.min`	minimal fraction of gene end region for peal detection
`sum.thresh`	threshold for the min sum of read counts, below this default to the input TTS

Value

A vector of TTS coordinates

Examples

```
See get.TTS()
```

`get.largest.interval` *Get the largest interval for each gene, given multiple TSS and TTS annotations*

Description

The input bed6 file can be derived from a gencode annotation file, as described in the vignette

Usage

```
get.largest.interval(bed = NULL)
```

Arguments

`bed`	bed6 frame with comprehensive gene annotations, defaults to NULL

Value

A bed6 frame with the largest annotation for each gene

Examples

```
# get intervals for furthest TSS and TTS +/- interval
bed.long = get.largest.interval(bed=dat0)
```

`get.TSS` *Select an optimal transcription start site (TSS) for each gene*

Description

We set a range around each annotated exon 1 within which to search for a read density. For each gene, we select the upstream coordinate of the first exon with the greatest read density, in the specified region, out of all first exons annotated for the given gene. The output of this function should be processed using the function apply.TSS.coords(). Note that input and output genes must be matched.

Usage

```
get.TSS(bed.in = NULL, bed.out = NULL, bw.plus = NULL,
  bw.minus = NULL, bp.range = NULL, cnt.thresh = NULL,
  low.read = FALSE)
```

Arguments

`bed.in`	bed6 frame with first exon gene annotations
`bed.out`	bed6 frame for output (same genes as bed.in)
`bw.plus`	plus strand bigWig data
`bw.minus`	minus strand bigWig data
`bp.range`	range in bp for identifying the exon 1 with the greatest density
`cnt.thresh`	read count number below which the TSS is not evaluated, by default the gene is removed from the analysis max(counts) < cnt.thresh. Indicate low.read=TRUE to use the upstream-most TSS for max(counts) < cnt.thresh.
`low.read`	Logical for whether to pick the upstream-most TSS if max(counts) < cnt.thresh (default FALSE)

Value

A list with a bed6 frame with inferred TSSs (bed) and a list of problematic genes (issues). Problems documented include start > end and start < 0. For start > end, the original input corrdinates were retained (bed.out). For start < 0, start = 0 was applied.

Examples

```
# select the TSS for each gene and incorporate these TSSs into the largest interval coordinates
bp.range = c(20,120)
cnt.thresh = 5
bed.out = largest.interval.expr.bed
bed.in = gencode.firstExon[gencode.firstExon$gene %in% bed.out$gene,]
TSS.gene = get.TSS(bed.in=bed.in, bed.out=bed.out,
                  bw.plus=bw.plus, bw.minus=bw.minus,
                  bp.range=bp.range, cnt.thresh=cnt.thresh)
```

`get.TTS` *Defining gene ends*

Description

Given regions within which to search for TTSs, we opperationally define the TTSs by binning the gene end regions, counting reads within the bins, fitting smooth spline curves to the bin counts, and detecting points at which the curves decay towards zero. We applied the constraint that there must be a specified number of bases in the gene end interval, otherwise the TTS analysis is not applied. Similarly, if the number of knots identified is too low, then we set the number of knots to a specified threshold. For this analysis, we set a sub-region at the beginning of the gene end region and identify the maximal peak from the spline fit. Then we identify the point at which the spline fit decays to a threshold level of of the peak level. We reasoned that the sub-region should be largest for genes with the greatest numbers of clipped bases, because such cases occur when the conventional gene ends are proximal to identified TSSs, and we should include these entire regions for analysis of the TTS. Similarly, we reasoned that for genes with substantially less clipped bases, and correspondingly larger gene end regions with greater potential for observing enhancers or divergent transcripts, the sub-regions should be smaller sections of the upstream-most gene end region. We use an exponential model to define the sub-regions. The user should use the output of get.end.intervals() as an input to this function. Note that gene ends will be set to zeros if there are no counts or if the interval is too small. This function calls get.gene.end().

Usage

```
get.TTS(bed = NULL, tss = NULL, bw.plus = NULL, bw.minus = NULL,
  bp.bin = NULL, pk.thresh = 0.1, knot.div = 40, knot.thresh = 5,
  cnt.thresh = 5, add.to.end = 50000, sum.thresh = 20,
  tau.dist = 10000, frac.max = 1, frac.min = 0.2)
```

Arguments

`bed`	bed6 gene coordinates with gene end intervals for estimation of the TTS
`tss`	bed6 gene coordinates containing estimated TSSs
`bw.plus`	plus strand bigWig data
`bw.minus`	minus strand bigWig data
`bp.bin`	the interval at the gene end will be separated into adjacent bins of this size
`pk.thresh`	the TTS is defined as pk.thresh percent of max peak of the spline fit
`knot.div`	the number of knots for the spline fit is defined as the number of bins covering the gene end divided by this number; increasing this parameter results and a smoother curve
`knot.thresh`	minimum number of knots, if knot.div gives a lower number, use this
`cnt.thresh`	read count number below which the TTS is not evaluated
`add.to.end`	the maximal length of the search region (bp)
`sum.thresh`	threshold for the min sum of read counts, below this default to the input TTS (default 20)
`tau.dist`	distance constant for the exponential defining the region for peak detection
`frac.max`	maximal fraction of gene end region for peal detection
`frac.min`	minimal fraction of gene end region for peal detection

Value

A bed6 file with TTS estimates incorporated, original TTSs are present if the TTS could not be estimated. The output is a list including the bed frame along with several metrics (see example below and vignette).

Examples

```
# identify gene ends
add.to.end = max(bed.for.tts.eval$xy)
knot.div = 40
pk.thresh = 0.05
bp.bin = 50
```

```
knot.thresh = 5
cnt.thresh = 5
tau.dist = 10000
frac.max = 1
frac.min = 0.2
inferred.coords = get.TTS(bed=bed.for.tts.eval, tss=TSS.gene.filtered3,
                          bw.plus=bw.plus, bw.minus=bw.minus,
                          bp.bin=bp.bin, add.to.end=add.to.end,
                          knot.div=knot.div,
                          pk.thresh=pk.thresh, knot.thresh=knot.thresh,
                          cnt.thresh=cnt.thresh, tau.dist=tau.dist,
                          frac.max=frac.max, frac.min=frac.min)
```

`get.tu.gene.coords` *Integrate inferred gene coordinates with transcriptional units*

Description

Here we annotate all of the regions in the TU frame based on overlaps with genes. The analysis details are handles by single.overlaps() and multi.overlaps().

Usage

```
get.tu.gene.coords(hmm.ann.overlap = NULL, tss.thresh = NULL,
  delta.tss = NULL, delta.tts = NULL)
```

Arguments

`hmm.ann.overlap`	frame with coordinates from intersecting tus with inferred gene annotations
`tss.thresh`	number of bp a TU beginning can be off from an annotation in order to be assigned that annotation
`delta.tss`	max distance between an upstream gene end and downstream gene start
`delta.tts`	max difference distance between and annotated gene end and the start of a downstream gene before an intermediate TU id is assigned

Value

A bed frame with gene/tu coordinates.

Examples

```
tss.thresh = 200
delta.tss = 50
delta.tts = 1000
res = get.tu.gene.coords(hmm.ann.overlap=hmm.ann.overlap,
                         tss.thresh=tss.thresh,
                         delta.tss=delta.tss,
                         delta.tts=delta.tts)
```

inside.starts *Identify genes with multiple starts inside*

Description

This function identifies genes within which >1 start is observed. That is, relatively 'large' genes in which 2 or more genes originate. Data generated from gene.overlaps() serve as input to this function.

Usage

```
inside.starts(vec = NULL)
```

Arguments

vec	input the xy column from the $is.a.start.inside output of gene.overlaps()

Value

A vector of genes within which multiple starts are observed.

Examples

```
overlap.data = gene.overlaps( bed = TSS.gene.filtered1 )
is.a.start.inside = overlap.data$is.a.start.inside
mult.inside.starts = inside.starts(vec = is.a.start.inside$xy)
```

multi.overlap.assign *Function to assign identifiers for class5 overlap TUs*

Description

This function is called inside of multiple.overlaps() and is not recommended to be used on its own.

Usage

```
multi.overlap.assign(fr = NULL, tss.thresh = NULL, delta.tss = NULL,
  delta.tts = NULL, cl = NULL)
```

Arguments

`fr`	= frame with full bedtools overlap data for a class5 TU with multiple internal annotations
`tss.thresh`	= number of bp a TU beginning can be off from an annotation in order to be assigned that annotation
`delta.tss`	= max distance between an upstream gene end and downstream gene start cl = multi-overlap class
`delta.tts`	max difference distance between and annotated gene end and the start of a downstream gene before an intermediate TU id is assigned
`cl`	multi-overlap class

Value

A bed data with chr, start, end, id, class, and strand. Note that id = 0 for regions of large TUs that do no match input annotations.

Examples

```
See multiple.overlaps()
```

`multi.overlaps` *Assign identifiers to TUs with multiple gene overlaps*

Description

This function will assign identifiers to TUs that overlapped with multiple genes. Trusted gene annotations are considered in terms of their degree of overlap with TUs identified in an unbiased manner. Marginal regions of TUs outside of gene overlaps are given generic identifiers.

Usage

```
multi.overlaps(overlaps = NULL, tss.thresh = NULL, delta.tss = NULL,
  delta.tts = NULL)
```

Arguments

`overlaps`	frame with bed intersect of inferred TUs (col1-6) with annotated TUs (col7-12) and overlaping bps (col14)
`tss.thresh`	number of bp a TU beginning can be off from an annotation in order to be assigned that annotation
`delta.tss`	max distance between an upstream gene end and downstream gene start
`delta.tts`	max difference distance between and annotated gene end and the start of a downstream gene before an intermediate TU id is assigned

Value

A list with bed data (bed) and counts for each class (cnt5-8). bed = data with chr, start, end, id, class, and strand. cnt5 = count of TU from class 5 overlaps

Examples

```
dup.hmm.rows = overlap.tu[duplicated(overlap.tu[,c(1:3,6)]) |
    duplicated(overlap.tu[,c(1:3,6)], fromLast=TRUE),]
dup.full = dup.hmm.rows
names(dup.full)[1:6] = c("infr.chr","infr.start","infr.end",
                         "infr.gene","infr.xy","infr.strand")
nrow(dup.full[,1:3] %>% unique)
nrow(dup.full %>% select(ann.gene) %>% unique)
tss.thresh = 200
delta.tss = 50
delta.tts = 1000
class5678 = multi.overlaps(overlaps=dup.full, tss.thresh=tss.thresh,
                           delta.tss=delta.tss, delta.tts=delta.tts)
new.ann.mult = class5678$bed
```

`read.count.body` *Count gene body reads and compute density*

Description

Input coordinates for evaluation of counts and densities. This function works with input that includes one annotation per gene.

Usage

```
read.count.body(bed = NULL, bw.plus = NULL, bw.minus = NULL,
  bp.start = 0)
```

Arguments

`bed`	a bed6 frame
`bw.plus`	plus strand bigWig data
`bw.minus`	minus strand bigWig data
`bp.start`	optional number of bases to omit from the upstream gene boundary

Value

A list with counts and desity. counts = vector with end of gene counts for each gene. density = vector with end of gene densities for each gene.

Examples

```
# get read counts and densities at the body of each annotated gene

body.reads = read.count.body(bed=largest.interval.bed, bw.plus=bw.plus,
bw.minus=bw.minus, bp.start=bp.start)
hist(log(body.reads$density), breaks=200, col="black",xlab="log read density",main="")
hist(log(body.reads$counts), breaks=200, col="black",xlab="log read count",main="")
```

---

read.count.end *Select a regions containing the gene ends*

Description

Select a fraction of the annotated gene end and consider an additional number of base pairs beyond the gene end within which to count reads. This function works with input that includes one annotation per gene.

Usage

```
read.count.end(bed = NULL, bw.plus = NULL, bw.minus = NULL,
  fraction.end = NULL, add.to.end = NULL)
```

Arguments

`bed`	a bed6 frame
`bw.plus`	plus strand bigWig data
`bw.minus`	minus strand bigWig data
`fraction.end`	fraction of the annotated gene end (0,1)
`add.to.end`	number of base pairs beyond the gene end within which to count reads

Value

A list with counts and desity. counts = vector with end of gene counts for each gene. density = vector with end of gene densities for each gene.

Examples

```
# get read counts and densities at the end of each annotated gene
fraction.end = 0.1
add.to.end = 0
end.reads = read.count.end(bed=bed.long, bw.plus=bw.plus, bw.minus=bw.minus,
                          fraction.end=fraction.end, add.to.end=add.to.end)
hist(log(end.reads$density))
hist(log(end.reads$counts))
```

`read.count.transcript` *Evaluate reads and compute densities for each transcript of each gene.*

Description

Input coordinates for evaluation of counts and densities. Return the maximal values across all transcripts of a given gene. In particular, we return the density associated with the max count for a specific gene.

Usage

```
read.count.transcript(bed = NULL, bw.plus = NULL, bw.minus = NULL)
```

Arguments

`bed`	a bed6 frame
`bw.plus`	plus strand bigWig data
`bw.minus`	minus strand bigWig data

Value

A list with counts and desity. counts = vector with end of gene counts for each gene. density = vector with end of gene densities for each gene.

Examples

```
# get read counts and densities for the max transcript of each annotated gene
transcript.reads = read.count.transcript(bed=gencode.transcript, bw.plus=bw.plus, bw.minus=bw.minus)
hist(log(transcript.reads$density), breaks=200, col="black",xlab="log read density",main="")
hist(log(transcript.reads$counts), breaks=200, col="black",xlab="log read count",main="")
```

`remove.overlaps` *Filter genes with overlaps*

Description

For a given case in which multiple genes overlap, this function modifies evaluates read density for all transcripts cooresponding to each gene. The gene with the largest density across all transcripts is retain in the input annotation. The related functions get.dups() and gene.overlaps() can be used to identify overlap cases.

Usage

```
remove.overlaps(bed = NULL, overlaps = NULL, transcripts = NULL,
  bw.plus = NULL, bw.minus = NULL)
```

Arguments

`bed`	bed6 file with gene annotations
`overlaps`	frame with 6 columns in the bed6 format and a 7th column indicating overlap cases

Value

A bed6 frame with overlaps removed. That is, a modified version of the input bed.

Examples

```
# run overlap analysis
overlap.data = gene.overlaps( bed = bed.long.filtered2.tss )
has.start.inside = overlap.data$has.start.inside
is.a.start.inside = overlap.data$is.a.start.inside
dim(has.start.inside)
dim(is.a.start.inside)
```

`single.overlap.assign` *Function to assign identifiers for class1 overlap TUs*

Description

This function is called inside of single.overlaps() and is not recommended to be used on its own.

Usage

```
single.overlap.assign(fr = NULL, tss.thresh = NULL, delta.tss = NULL,
  delta.tts = NULL, cl = NULL)
```

Arguments

`fr`	frame with full bedtools overlap data for a class1 TU with multiple internal annotations
`tss.thresh`	number of bp a TU beginning can be off from an annotation in order to be assigned that annotation
`delta.tss`	max distance between an upstream gene end and downstream gene start
`cl`	multi-overlap class

Value

A bed data frame with chr, start, end, id, class, and strand. Note that id = 0 for regions of large TUs that do no match input annotations.

Examples

```
See single.overlaps()
```

single.overlaps *Assign identifiers to TUs with single gene overlaps*

Description

This function will assign identifiers to TUs that overlapped with single genes. Trusted gene annotations are considered in terms of their degree of overlap with TUs identified in an unbiased manner. Marginal regions of TUs outside of gene overlaps are given generic identifiers.

Usage

```
single.overlaps(overlaps = NULL, tss.thresh = NULL, delta.tss = NULL,
  delta.tts = NULL)
```

Arguments

overlaps	frame with bed intersect of inferred TUs (col1-6) with annotated TUs (col7-12) and overlaping bps (col14)
tss.thresh	= number of bp a TU beginning can be off from an annotation in order to be assigned that annotation
delta.tss	max distance between an upstream gene end and downstream gene start
delta.tts	max difference distance between and annotated gene end and the start of a downstream gene before an intermediate TU id is assigned

Value

A list with bed data (bed) and counts for each class (cnt1-4). bed = data with chr, start, end, id, class, and strand. cnt1 = count of TU from class 1 overlaps

Examples

```
sing.hmm.rows = overlap.tu[!duplicated(overlap.tu[,c(1:3,6)]) &
     !duplicated(overlap.tu[,c(1:3,6)], fromLast=TRUE),]
overlaps = sing.hmm.rows
names(overlaps)[1:6] = c("infr.chr","infr.start","infr.end",
                         "infr.gene","infr.xy","infr.strand")
tss.thresh = 200
delta.tss = 50
delta.tts = 1000
class1234 = single.overlaps(overlaps=overlaps, tss.thresh=tss.thresh,
                            delta.tss=delta.tss, delta.tts=delta.tts)
new.ann.sing = class1234$bed
```

`TSS.count.dist` *Get distances from identified transcription start sites (TSSs) to the nearest regions with peak reads*

Description

This function was designed to evaluate the results of out TSS identification analysis. We specify a region centered on the TSS. We obtain read counts in bins that span this window. We sort the genes based on the bin with the maximal reads and we scale the data to the interval (0,1) for visualization. We compute the distances between the TSS and the bin with the max reads within the specified window.

Usage

```
TSS.count.dist(bed = NULL, bw.plus = NULL, bw.minus = NULL,
  window = NULL, bp.bin = NULL)
```

Arguments

`bed`	bed frame for TSS evaluation
`bw.plus`	= plus strand bigWig data
`bw.minus`	minus strand bigWig data
`window`	region size, centered on the TSS for analysis
`bp.bin`	the interval will be separated into adjacent bins of this size

Value

A list with three vector elements: raw, scaled, and dist. All outputs are organized based on TSS position (left-upstream). raw = binned raw counts. scaled = binned scaled counts (0,1). dist = upper bound distances (min(|dists|) = bp.bin). See examples and vignette for more details.

Examples

```
see eval.tss()
```

`TTS.boundary.match` *Check for the fraction of identified TTSs that match the search boundry*

Description

This function considers information in both coords and bed.for.tts.eval (see vignette) to identify the fraction of TTSs that match the boundary of the search region.

Usage

```
TTS.boundary.match(coords = NULL, bed.for.tts.eval = NULL)
```

Arguments

`coords`	bed6 frame with inferred coordinates
`bed.for.tts.eval`	bed6 frame with TTS search regions (see get.end.intervals())

Value

A fraction [0,1].

Examples

```
frac.match = TTS.boundary.match(coords=coords, bed.for.tts.eval=bed.for.tts.eval)
```

`TTS.boundry.clip` *Look at whether TTSs identified at the search boundry were clipped.*

Description

This function considers information in both coords and bed.for.tts.eval (see vignette) to identify the fractions of search boundary TTSs that were clipped and unclipped.

Usage

```
TTS.boundry.clip(coords = NULL, bed.for.tts.eval = NULL)
```

Arguments

`coords`	bed6 frame with inferred coordinates
`bed.for.tts.eval`	bed6 frame with TTS search regions (see get.end.intervals())

Value

A list with two fractions [0,1], frac.clip and frac.noclip, and a vector of clip distances (clip.tts) for genes with boundary TTSs.

Examples

```
# look at whether TTSs identified at the search boundry had clipped boundries
frac.bound.clip = TTS.boundry.clip(coords=coords, bed.for.tts.eval=bed.for.tts.eval)
frac.bound.clip$frac.clip
frac.bound.clip$frac.noclip

# plot didtribution of clip distances for genes with boundary TTSs
hist(frac.bound.clip$clip.tts,main="",xlab="bp clipped for boundry genes")
```

TTS.count.dist *Get read counts around transcription termination sites (TTSs)*

Description

We set a window around the TTS and segment the window into bins. To sort the read data, we compute cumulative counts of reads, and we sort based on the bin at which a specified percentage of the reads are found. We also take a ratio of gene counts downstream / upstream of the TTS with regions within an interval.

Usage

```
TTS.count.dist(bed = NULL, bw.plus = NULL, bw.minus = NULL,
  window = NULL, bp.bin = NULL, frac.max = NULL,
  ratio.region = NULL)
```

Arguments

`bed`	bed frame for TSS evaluation
`bw.plus`	plus strand bigWig data
`bw.minus`	minus strand bigWig data
`window`	region size, centered on the TTS for analysis
`bp.bin`	the interval will be separated into adjacent bins of this size
`frac.max`	fraction of cumulative distribution for sorting entries are sorted by indices min(cumsum(x)/sum(x)) < frac.max
`ratio.region`	number of bp on either side of the TTS to evaluate the ratio entries

Value

A list with three elements: raw, scaled, and ratio. All outputs are organized based on TTS position (left-upstream, see frac.max). raw = binned raw counts. scaled = binned scaled counts (0,1). ratio = downstream(ratio.region) / upstream(ratio.region).

Examples

```
# look at read distribution around identified TTSs
window = 1000
bp.bin = 10
frac.max = 0.8
ratio.region = 300
tts.dists = TTS.count.dist(bed=bed.tss.tts, bw.plus=bw.plus, bw.minus=bw.minus,
     window=window, bp.bin=bp.bin, frac.max=frac.max,
     ratio.region=ratio.region)
```

`tts.plot` *Plot inferred and largest interval coordinates*

Description

Plot the inferred and largest interval coordinates, along with read data, indicate the peak search region, and indicate the total search region.

Usage

```
tts.plot(coords = NULL, gene.end = NULL, long.gene = NULL,
  bw.plus = NULL, bw.minus = NULL, bp.bin = 5, gene = NULL,
  xper = 0.2, yper = 1.3, add.to.end = NULL, tau.dist = NULL,
  frac.max = NULL, frac.min = NULL)
```

Arguments

`coords`	bed6 frame with inferred coordinates
`gene.end`	bed6 frame with peak search regions
`long.gene`	bed6 frame with largest interval coordinates
`bw.plus`	plus strand bigWig data
`bw.minus`	minus strand bigWig data
`gene`	gene for plotting
`xper`	fraction of the x-axis before the gene starts
`yper`	fraction of the y-axis with open space above the read data
`add.to.end`	the maximal length of the search region (bp)
`tau.dist`	distance constant for the exponential defining the region for peak detection
`frac.max`	maximal fraction of gene end region for peal detection
`frac.min`	minimal fraction of gene end region for peal detection

Value

A plot with reads, interred annptation (black) largest interval annotation (gray), TTS search interval (solid, vertical), and TTS peak interval (dashed, vertical).

Examples

```
gene.end = bed.for.tts.eval
long.gene = largest.interval.bed
tts.plot(coords=coords, gene.end=gene.end, long.gene=long.gene,
         gene = "Pparg", xper=0.2, yper=0.3,
         frac.min=frac.min, frac.max=frac.max,
         bw.plus=bw.plus, bw.minus=bw.minus, bp.bin=5,
         add.to.end=add.to.end, tau.dist=tau.dist)
```

`tu.gene.overlaps` *Assign identifiers to TUs with single gene overlaps*

Description

Summarize gene/tu counts after intersecting the respective annotations. See the vignette for analysis details and instructions.

Usage

```
tu.gene.overlaps(hmm.ann.overlap = NULL)
```

Arguments

`hmm.ann.overlap`	frame with coordinates from intersecting tus with inferred gene annotations

Value

A list of metrics inclusing the total number of TUs (n.tus), the number of TUs without gene overlaps (n.tu.no.gene.overlap), the number of TUs with overlapping genes (n.tu.overlap.gene), the number of genes with overlapping TUs (n.gene.overlap.tu), the number of TUs overlapping single genes (n.tu.overlap.single.genes), and the number of TUs with multiple gene overlaps (n.tu.overlap.multiple.genes). Note that n.tu.overlap.gene = n.tu.overlap.single.genes + n.tu.overlap.multiple.genes.

Examples

```
tg.overlaps = tu.gene.overlaps(hmm.ann.overlap=hmm.ann.overlap)
```

Index