Package 'JunctionSeq'

October 16, 2019

Version 1.14.0

Title JunctionSeq: A Utility for Detection of Differential Exon and Splice-Junction Usage in RNA-Seq data

Depends R (>= 3.2.2), methods, SummarizedExperiment (>= 0.2.0), Rcpp (>= 0.11.0), RcppArmadillo (>= 0.3.4.4)

Imports DESeq2 (>= 1.10.0), statmod, Hmisc, plotrix, stringr, Biobase (>= 2.30.0), locfit, BiocGenerics (>= 0.7.5), BiocParallel, genefilter, geneplotter, S4Vectors, IRanges, GenomicRanges,

Suggests MASS, knitr, JctSeqData, BiocStyle

Enhances Cairo, pryr

Description A Utility for Detection and Visualization of Differential Exon or Splice-Junction Usage in RNA-Seq data.

License file LICENSE

VignetteBuilder knitr

LinkingTo Rcpp, RcppArmadillo

NeedsCompilation yes

biocViews ImmunoOncology, Sequencing, RNASeq, DifferentialExpression

URL http://hartleys.github.io/JunctionSeq/index.html

BugReports https://github.com/hartleys/JunctionSeq/issues

Author Stephen Hartley [aut, cre] (PhD),
Simon Anders [cph],
Alejandro Reyes [cph]

Maintainer Stephen Hartley <JunctionSeq-contact@list.nih.gov>

git_url https://git.bioconductor.org/packages/JunctionSeq

git_branch RELEASE_3_9

git_last_commit ea9dcd8

git_last_commit_date 2019-05-02

Date/Publication 2019-10-15

R topics documented:

`buildAllPlots` *Create and save a full battery of JunctionSeq expression plots.*

Description

Saves a large battery of plots displaying the analysis results, for the purposes of data visualization. By default it saves a full set of plots for every gene that shows statistical significance and the adjusted-p < 0.01 level. Alternatively, it can be supplied with a specific gene list using the `gene.list` parameter, and will plot those specific genes.

Note that this function has MANY parameters, allowing the user to tweak the appearance of the plots to suit their particular needs and preferences. Don't be daunted: the default parameters are probably fine for most purposes.

Usage

```
buildAllPlots(jscs,
    outfile.prefix = "./",
    gene.list = NULL, FDR.threshold = 0.01, max.gene.ct,
    method.selectionCriterion = c("feature-pAdjust", "genewise-pAdjust"),
    use.plotting.device = c("png","CairoPNG","svg",
                            "tiff","cairo_ps","custom"),
    sequencing.type = c("paired-end","single-end"),
    use.vst=FALSE,use.log = TRUE,
    exon.rescale.factor = 0.3,
    subdirectories.by.type = TRUE,
    ma.plot=TRUE, variance.plot=TRUE,
    with.TX=TRUE,without.TX=TRUE,
    expr.plot=TRUE,normCounts.plot=TRUE,
    rExpr.plot=TRUE,rawCounts.plot=FALSE,
```

```
    colorRed.FDR.threshold = FDR.threshold,
    colorList=list(),
    plot.gene.level.expression = TRUE,
    plot.exon.results, plot.junction.results, plot.novel.junction.results,
    plot.untestable.results = FALSE,
    plot.lwd=3, axes.lwd = plot.lwd, anno.lwd = plot.lwd,
    gene.lwd = plot.lwd / 2,
    par.cex = 1, anno.cex.text = 1, anno.cex.axis = anno.cex.text,
    anno.cex.main = anno.cex.text * 1.2,
    drawCoordinates = TRUE,
    yAxisLabels.inExponentialForm = FALSE,
    show.strand.arrows = 1,
    graph.margins = c(2, 3, 3, 3),
    base.plot.height = 12, base.plot.width = 12,
    base.plot.units = "in",
    GENE.annotation.relative.height = 0.15,
    TX.annotation.relative.height = 0.05,
    CONNECTIONS.relative.height = 0.1,
    SPLICE.annotation.relative.height = 0.1,
    TX.margins = c(0,0.5),
    autoscale.height.to.fit.TX.annotation = TRUE,
    autoscale.width.to.fit.bins = 35,
    plotting.device.params = list(),
    number.plots = FALSE,
    name.files.with.geneID = TRUE,
    condition.legend.text, include.TX.names = TRUE,
    draw.start.end.sites = TRUE,
    openPlottingDeviceFunc, closePlottingDeviceFunc,
    writeHTMLresults = TRUE,
    html.cssFile, html.cssLink, html.imgFileExtension,
    html.plot.height = 90, html.plot.height.units = "vh",
    html.compare.results.list = NULL,
    minimalImageFilenames = writeHTMLresults,
    verbose=TRUE, debug.mode = FALSE,
    INTERNAL.VARS = list(),
    ...)
```

Arguments

jscs	A JunctionSeqCountSet. Usually created by runJunctionSeqAnalyses. Alternatively, this can be created manually by readJunctionSeqCounts. However in this case a number of additional steps will be necessary: Dispersions and size factors must then be set, usually using functions estimateSizeFactors and estimateJunctionSeqDispersions. Hypothesis tests must be performed by testForDiffUsage. Effect sizes and parameter estimates must be created via estimateEffectSizes.
outfile.prefix	The prefix file path to save the images to.
gene.list	Character vector. List of genes to plot. Either this variable OR FDR.threshold must be set.
FDR.threshold	If this option is used, genes will be selected for plotting based on the presence of statistically significant junctions. The adjusted-p-value threshold used to de-

termine significance. Only genes containing at least 1 significant feature will be plotted.

`max.gene.ct` Integer or numeric value. This option is usually only used with the FDR.threshold parameter (as opposed to the gene.list parameter). This option sets an upper limit to the number of genes to plot. This prevents JunctionSeq from taking too long to complete, or from using too much disk space if an enormous number of genes turn out to be significant at the selected significance value. If there are more genes than max.gene.ct, then JunctionSeq will only plot the top max.gene.ct genes.

`method.selectionCriterion`
Determines the method used to select genes for plotting. If set to "feature-pAdjust", genes will be selected if one or more features show significance. If "genewise-pAdjust" is used, gene-wise adjusted p-values will be generated and genes will be filtered on that basis.

`use.plotting.device`
The plotting device to use.

`sequencing.type`
The type of sequencing used, either "paired-end" or "single-end". This only affects the labelling of the y-axis, and does not affect the actual plots in any way.

`use.vst` Logical. If `TRUE`, all plots will be scaled via a variance stabilizing transform.

`use.log` Logical. If `TRUE`, all plots will be log-scaled.

`exon.rescale.factor`
Floating point numeric value. To improve readability the exons drawn in the coordinate annotation are rescaled by default so that they take up 30 percent of the x axis. This makes the plots easier to read, as exons are usually much smaller than introns and thus a group of clustered exons can be hard to distinguish when plotted on a simple scale. If this value is set to NA or a value not between 0 and 1 inclusive, then the exons and introns will be drawn on the same scale. Note that this function can also take the `exonRescaleFunction` parameter, which is passed to `plotJunctionSeqResultsForGene`.

`subdirectories.by.type`
Logical value. If `TRUE`, then subdirectories will be created for each plot type using outfile.prefix as a parent directory. Note that `FALSE` is not compatible with some functionality, including HTML generation. For advanced users only.

`ma.plot` if TRUE, generate and save a MA plot. A MA-plot is a plot of fold change versus base mean normalized counts.

`variance.plot` if TRUE, generate and save a plot of the dispersion as a function of the base mean.

`with.TX` if TRUE, save expression plots with the full transcripts printed

`without.TX` if TRUE, save expression plots with only the compiled exons printed. Note that if this and with.TX.plot are both TRUE, both versions will be saved seperately.

`expr.plot` if TRUE, save an expression plot of the expression parameter estimates for each splice site, for each condition.

`normCounts.plot`
if TRUE, save an expression plot of the normalized mean counts for each splice site, for each sample.

`rExpr.plot` if TRUE, save an expression plot of the expression parameter estimates, relative to gene-wide expression, for each splice site, for each condition.

`rawCounts.plot` if TRUE, save an expression plot of the raw counts for each splice site, for each sample. Note that these will never be VST-transformed, even when use.vst == TRUE.

`colorRed.FDR.threshold`
The adjusted-p-value threshold used to determine whether a feature should be marked as "significant" and colored pink. By default this will be the same as the FDR.threshold.

`colorList` A named list of R colors, setting the colors used for various things. See `plotJunctionSeqResultsFo`

`plot.gene.level.expression`
Logical value. If `TRUE`, gene-level expression (when applicable) will be plotted beside the sub-element-specific expression in a small seperate plotting box. For the "relative expression" plots the simple mean normalized expression will be plotted (since it doesn't make sense to plot something relative to itself).

`plot.exon.results`
Logical. If `TRUE`, plot results for exons. By default everything that was tested will be plotted.

`plot.junction.results`
Logical. If `TRUE`, plot results for splice junctions. By default everything that was tested will be plotted.

`plot.novel.junction.results`
Logical. If `TRUE`, plot results for novel splice junctions. If false, novel splice junctions will be ignored. By default everything that was tested will be plotted.

`plot.untestable.results`
Logical. If `TRUE`, plots splice junctions that had coverage that was too low to be tested. Note that, in general, only normCounts and rawCounts plots will have non-`NA` values for untestable counting bins.

`plot.lwd` The line width for the plotting lines.

`axes.lwd` The line width for the axes.

`anno.lwd` The line width for the various other annotation lines.

`gene.lwd` The line width used for the gene annotation lines. The default is half the standard line width.

`par.cex` The base cex value to be passed to par() immediately before all plots are created. See `par`.

`anno.cex.text` The font size multiplier for most annotation text. This will be multiplied by a factor of the par.cex value. More specifically: The cex value to be passed to all function calls that take graphical parameters. See `par`.

`anno.cex.axis` The font size multiplier for the axis text. This will be multiplied by a factor of the par.cex value. More specifically: The cex.axis value to be passed to all function calls that take graphical parameters. See `par`.

`anno.cex.main` The font size multiplier for the main title text. This will be multiplied by a factor of the par.cex value. More specifically: The cex.main value to be passed to all function calls that take graphical parameters. See `par`.

`drawCoordinates`
Whether to label the genomic coordinates at the bottom of the plot.

`yAxisLabels.inExponentialForm`
Logical. If `TRUE`, then the y-axis will be labelled in exponential form.

show.strand.arrows
: The number of strand-direction arrows to display. If equal to 1 (the default) then the arrow will extend from the end of the gene drawing, if it is greater than 1 then arrows will be drawn along the gene length. If it is 0 or NA then arrows will not be drawn.

graph.margins
: Numeric vector of length 4. These margins values used (as if for par("mar")) for the main graph. The lower part of the plot uses the same left and right margins.

base.plot.height
: The base height of the standard-sized plots. Plots that include the full transcript annotation will be expanded by the height of these additional rows. See the withTxPlot.height.multiplier parameter, below.

base.plot.width
: The width of the plots.

base.plot.units
: The units of measurement for the plot height and width. Default is px, or pixels.

GENE.annotation.relative.height
: The height of the "gene track" displayed underneath the main graph, relative to the height of the main graph. By default it is 20 percent.

TX.annotation.relative.height
: For all plots that draw the annotated-transcript set (when the with.TX parameter is TRUE), this sets the height of each transcript, as a fraction of the height of the main graph. By default it is 2.5 percent.

CONNECTIONS.relative.height
: The height of the panel that connects the plotting area to the gene annotation area, relative to the height of the plotting area. This panel has the lines that connects the counting bin columns to their actual loci on the gene. By default it is 10 percent.

SPLICE.annotation.relative.height
: The height of the area that shows the splice junction loci, relative to the size of the plotting area.

TX.margins
: A numeric vector of length 2. The size of the blank space between the gene plot and the transcript list and then beneath the transcript list, relative to the size of each transcript line.

autoscale.height.to.fit.TX.annotation
: Logical. Plots that include the full transcript annotation generally need to have a larger height in order to maintain readability. If TRUE) (the default), all plots that include transcripts will be expanded vertically to fit the additional transcripts. This maintains the same appearance and aspect ratio of the main graph area, but means that the height of the plot will differ between genes when TX are included. This parameter can be used to override that behavior if a specific figure size is desired. If FALSE, then the height of the entire output image will always be equal to base.plot.height.

autoscale.width.to.fit.bins
: Integer value. JunctionSeq will automatically go to great lengths to autofit the data in a readable way. By default, any plots that have more than 35 plotting columns will be widened linearly to fit the excess columns. This parameter can be used to change that value, or turn it off entirely by setting this parameter to NA.

condition.legend.text
: List or named vector of character strings. This optional parameter can be used to assign labels to each condition variable values. It should be a list or named vector with length equal to factor(condition). Each element should be named

with one of the values from `factor(condition)`, and should contain the label. They will be listed in this order in the figure legend.

`include.TX.names`
: Logical value. If `TRUE`, then for the plots that include the annotated transcript, the transcript names will be listed. The labels will be drawn at half the size of `anno.cex.text`.

`plotting.device.params`
: Additional parameters to be passed to the plotting device.

`number.plots`
: Whether to number each gene in the image names, based on either the order they appear in the input gene.list, or in order of ascending p-values.

`name.files.with.geneID`
: Whether to use the geneID (rather than gene name) for naming the files.

`draw.start.end.sites`
: Logical value. If `TRUE`, then transcript start/end sites will be marked on the main gene annotation.

`openPlottingDeviceFunc`
: An R function. This option can be used to use plotting devices other than the ones directly supported by JunctionSeq. This must be a function that must have 3 parameters: filename, heightMult, and widthMult. It should open the desired plotting device. For advanced users only.

`closePlottingDeviceFunc`
: An R function. This must be used in conjunction with openPlottingDeviceFunc. For most devices, you can just use the function "dev.off". For advanced users only.

`writeHTMLresults`
: If TRUE, write an index html file to present the results in a navigable way.

`html.cssFile`
: Optional: specify a css file to use. Copies the entire contents of the supplied file into the page directory and links to it with relative links.

`html.cssLink`
: Optional: specify an external css file to use. This can be an absolute or relative link.

`html.imgFileExtension`
: The file extension of the image files. This is only needed if you are using a custom device. If you are using one of the default devices, it will autodetect the file extension.

`html.plot.height`
: Numeric. The base height of the plot, for the plots without TX annotation. The default is 90.

`html.plot.height.units`
: The units used for the `html.plot.height` parameter. The default is "vh", which sets the height relative to the available max height.

`html.compare.results.list`
: Named list of character strings. (Advanced) Optional parameter that allows you to cross-link multiple analyses for easy navigation between analysis for specific genes of interest. In order to create such cross-linking, you will need to run `builtAllPlots` separately for each analysis. The `outfile.prefix` for each run must be a sub-directory of the same parent directory. The `html.compare.results.list` must be a named list of these subdirectories. `names(html.compare.results.list)` must be the title of each analysis as you want it to appear in the navigation links. Note: This parameter is incompatible with the `number.plots` option.

`minimalImageFilenames`
: Logical. If `TRUE`, then the image files will not include the gene names, but instead will be numbered in order. The html files will still have the full length names. This option was added because many web host servers will refuse to host image files whose length exceeds 32 characters. By default this option will be `TRUE` iff writeHTMLresults is `TRUE`.

`verbose`
: if TRUE, send debugging and progress messages to the console / stdout.

`debug.mode`
: if TRUE, send even more debugging and progress messages to the console / stdout.

`INTERNAL.VARS`
: NOT FOR GENERAL USE. Intended only for use by JunctionSeq itself, internally. This is used for passing pre-generated data (when generating many similar plots, for example), and for internally-generated parameters. DO NOT USE.

`...`
: Additional options to pass to `buildAllPlotsForGene`, `plotJunctionSeqResultsForGene`, or graphical parameters passed to plotting functions.

Value

This is a side-effecting function, and does not return a value.

Examples

```
data(exampleDataSet,package="JctSeqData");
buildAllPlots(jscs);

## Not run:
########################################
#Set up example data:
decoder.file <- system.file(
                  "extdata/annoFiles/decoder.bySample.txt",
                  package="JctSeqData");
decoder <- read.table(decoder.file,
                  header=TRUE,
                  stringsAsFactors=FALSE);
gff.file <- system.file(
            "extdata/cts/withNovel.forJunctionSeq.gff.gz",
            package="JctSeqData");
countFiles <- system.file(paste0("extdata/cts/",
     decoder$sample.ID,
     "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
     package="JctSeqData");
########################################
#Run example analysis:
jscs <- runJunctionSeqAnalyses(sample.files = countFiles,
           sample.names = decoder$sample.ID,
           condition=factor(decoder$group.ID),
           flat.gff.file = gff.file,
           analysis.type = "junctionsAndExons"
);
########################################

#Generate all plots and the html index
#   Save them as pngs to the current directory:
buildAllPlots(jscs);
```

```
## End(Not run)
```

---

buildAllPlotsForGene *Create and save one or more JunctionSeq expression plots.*

Description

Generates and saves one or more plots, displaying counts or averages for all counting bins across one particular gene. The parameters `expr.plot`, `normCounts.plot`, `rExpr.plot`, and `rawCounts.plot` determine which plot types are to be generated, and the parameters `with.TX` and `without.TX` determines whether these plots should include or not include the full transcript information, or if separate plots should be generated with and without the full transcript information.

Note that this function has MANY parameters, allowing the user to tweak the behavior and appearance of the plots to suit their particular needs and preferences. Don't be daunted: the default parameters are probably fine for most purposes.

Usage

```
buildAllPlotsForGene(geneID, jscs,
     outfile.prefix = "./",
     use.plotting.device = c("png","CairoPNG","svg",
                             "tiff","cairo_ps","custom"),
     sequencing.type = c("paired-end","single-end"),
     use.vst=FALSE, use.log = TRUE,
     exon.rescale.factor = 0.3,
     with.TX=TRUE,without.TX=TRUE,
     expr.plot=TRUE,normCounts.plot=TRUE,
     rExpr.plot=TRUE,rawCounts.plot=FALSE,
     colorRed.FDR.threshold = 0.01,
     colorList=list(),
     plot.gene.level.expression = TRUE,
     plot.exon.results, plot.junction.results, plot.novel.junction.results,
     plot.untestable.results = FALSE,
     plot.lwd=3, axes.lwd = plot.lwd, anno.lwd = plot.lwd,
     gene.lwd = plot.lwd / 2,
     par.cex = 1, name.files.with.geneID = TRUE,
     anno.cex.text = 1,
     anno.cex.axis = anno.cex.text, anno.cex.main = anno.cex.text * 1.2,
     drawCoordinates = TRUE,
     yAxisLabels.inExponentialForm = FALSE,
     show.strand.arrows = 1,
     graph.margins = c(2, 3, 3, 3),
     base.plot.height = 12, base.plot.width = 12,
     base.plot.units = "in",
     GENE.annotation.relative.height = 0.15,
     TX.annotation.relative.height = 0.05,
     CONNECTIONS.relative.height = 0.1,
     SPLICE.annotation.relative.height = 0.1,
     TX.margins = c(0,0.5),
```

```
    autoscale.height.to.fit.TX.annotation = TRUE,
    autoscale.width.to.fit.bins = 35,
    plotting.device.params = list(),
    condition.legend.text, include.TX.names = TRUE,
    draw.start.end.sites = TRUE, draw.nested.SJ = TRUE,
    openPlottingDeviceFunc = NULL, closePlottingDeviceFunc = NULL,
    minimalImageFilenames = FALSE,
    verbose=TRUE, debug.mode = FALSE,
    INTERNAL.VARS=list(),
    ...)
```

Arguments

geneID	Character string. Which gene to plot.
jscs	A JunctionSeqCountSet. Usually created by runJunctionSeqAnalyses. Alternatively, this can be created manually by readJunctionSeqCounts. However in this case a number of additional steps will be necessary: Dispersions and size factors must then be set, usually using functions estimateSizeFactors and estimateJunctionSeqDispersions. Hypothesis tests must be performed by testForDiffUsage. Effect sizes and parameter estimates must be created via estimateEffectSizes.
outfile.prefix	Character string or vector. Sets the prefix file path where image files should be saved. If the destination directory does not exist then JunctionSeq will automatically create it. If the parent directory does not exist then JunctionSeq will throw an error. Optionally, outfile.prefix can be a vector of strings, assigning a different file prefix to each plot. This alternative is primarily intended for internal use by buildAllPlots.
use.plotting.device	The plotting device to use.
sequencing.type	The type of sequencing used, either "paired-end" or "single-end". This only affects the labelling of the y-axis, and does not affect the actual plots in any way.
use.vst	Logical. If TRUE, all plots will be scaled via a variance stabilizing transform.
use.log	Logical. If TRUE, all plots will be log-scaled.
exon.rescale.factor	Numeric. Exons will be proportionately scaled-up so that the exonic regions make up this fraction of the horizontal plotting area. If negative, exons and introns will be plotted to a common scale.
with.TX	if TRUE, save expression plots with the full transcripts printed
without.TX	if TRUE, save expression plots with only the compiled exons printed. Note that if this and with.TX.plot are both TRUE, both versions will be saved seperately.
expr.plot	if TRUE, save an expression plot of the expression parameter estimates for each splice site, for each condition.
normCounts.plot	if TRUE, save an expression plot of the normalized mean counts for each splice site, for each sample.
rExpr.plot	if TRUE, save an expression plot of the expression parameter estimates, relative to gene-wide expression, for each splice site, for each condition.

`rawCounts.plot` if TRUE, save an expression plot of the raw counts for each splice site, for each sample. Note that these will never be VST-transformed, even when use.vst == TRUE.

`colorRed.FDR.threshold`
The adjusted-p-value threshold used to determine whether a feature should be marked as "significant" and colored pink. By default this will be the same as the FDR.threshold.

`colorList` A named list of R colors, setting the colors used for various things. See `plotJunctionSeqResultsFo`

`plot.gene.level.expression`
Logical value. If TRUE, gene-level expression (when applicable) will be plotted beside the sub-element-specific expression in a small seperate plotting box. For the "relative expression" plots the simple mean normalized expression will be plotted (since it doesn't make sense to plot something relative to itself).

`plot.exon.results`
Logical. If TRUE, plot results for exons. By default everything that was tested will be plotted.

`plot.junction.results`
Logical. If TRUE, plot results for splice junctions. By default everything that was tested will be plotted.

`plot.novel.junction.results`
Logical. If TRUE, plot results for novel splice junctions. If false, novel splice junctions will be ignored. By default everything that was tested will be plotted.

`plot.untestable.results`
Logical. If TRUE, plots splice junctions that had coverage that was too low to be tested.

`plot.lwd` the line width for the plotting lines.

`axes.lwd` the line width for the axes.

`anno.lwd` the line width for the various other annotation lines.

`gene.lwd` the line width used for the gene annotation lines.

`par.cex` The base cex value to be passed to par() immediately before all plots are created. See `par`.

`name.files.with.geneID`
Whether to use the geneID (rather than gene name) for naming the files.

`anno.cex.text` The font size multiplier for most annotation text. This will be multiplied by a factor of the par.cex value. More specifically: The cex value to be passed to all function calls that take graphical parameters. See `par`.

`anno.cex.axis` The font size multiplier for the axis text. This will be multiplied by a factor of the par.cex value. More specifically: The cex.axis value to be passed to all function calls that take graphical parameters. See `par`.

`anno.cex.main` The font size multiplier for the main title text. This will be multiplied by a factor of the par.cex value. More specifically: The cex.main value to be passed to all function calls that take graphical parameters. See `par`.

`drawCoordinates`
Whether to label the genomic coordinates at the bottom of the plot.

`yAxisLabels.inExponentialForm`
Logical. If TRUE, then the y-axis will be labelled in exponential form.

show.strand.arrows
: The number of strand-direction arrows to display. If equal to 1 (the default) then the arrow will extend from the end of the gene drawing, if it is greater than 1 then arrows will be drawn along the gene length. If it is 0 or NA then arrows will not be drawn.

graph.margins
: Numeric vector of length 4. These margins values used (as if for par("mar")) for the main graph. The lower part of the plot uses the same left and right margins.

base.plot.height
: The base height of the standard-sized plots. Plots that include the full transcript annotation will be expanded by the height of these additional rows. See the withTxPlot.height.multiplier parameter, below.

base.plot.width
: The base width of the plots (plots with a large number of features may be scaled up, see parameter autoscale.width.to.fit.bins).

base.plot.units
: The units of measurement for the plot height and width. Default is px, or pixels.

GENE.annotation.relative.height
: The height of the "gene track" displayed underneath the main graph, relative to the height of the main graph. By default it is 20 percent.

TX.annotation.relative.height
: For all plots that draw the annotated-transcript set (when the with.TX parameter is TRUE), this sets the height of each transcript, as a fraction of the height of the main graph. By default it is 2.5 percent.

CONNECTIONS.relative.height
: The height of the panel that connects the plotting area to the gene annotation area, relative to the height of the plotting area. This panel has the lines that connects the counting bin columns to their actual loci on the gene. By default it is 10 percent.

SPLICE.annotation.relative.height
: The height of the area that shows the splice junction loci, relative to the size of the plotting area.

TX.margins
: A numeric vector of length 2. The size of the blank space between the gene plot and the transcript list and then beneath the transcript list, relative to the size of each transcript line.

autoscale.height.to.fit.TX.annotation
: Plots that include the full transcript annotation generally need to have a larger height in order to maintain readability. By default, all plots that include transcripts will be expanded vertically by the height of the additional transcripts. This maintains the same appearance and aspect ratio of the main graph, but also means that the height of the plot will differ between genes. This parameter can be used to override that behavior if a specific figure size is desired. If TRUE, the base.plot.height will be used as the height of the plot, regardless of how many transcripts are included.

autoscale.width.to.fit.bins
: Integer value. JunctionSeq will automatically go to great lengths to autofit the data in a readable way. By default, any plots that have more than 35 plotting columns will be widened linearly to fit the excess columns. This parameter can be used to change that value, or turn it off entirely by setting this parameter to NA.

plotting.device.params
: Additional parameters to be passed to the plotting device.

`condition.legend.text`
: List or named vector of character strings. This optional parameter can be used to assign labels to each condition variable values. It should be a list or named vector with length equal to `factor(condition)`. Each element should be named with one of the values from `factor(condition)`, and should contain the label. They will be listed in this order in the figure legend.

`include.TX.names`
: Logical value. If `TRUE`, then for the plots that include the annotated transcript, the transcript names will be listed. The labels will be drawn at half the size of `anno.cex.text`.

`draw.start.end.sites`
: Logical value. If `TRUE`, then transcript start/end sites will be marked on the main gene annotation.

`draw.nested.SJ`
: Logical. If `TRUE`, overlapping splice junctions will be drawn layered under one another. This can vastly improve readability when there are a large number of overlapping splice junctions. Default is `TRUE`.

`openPlottingDeviceFunc`
: An R function. This option can be used to use plotting devices other than the ones directly supported by JunctionSeq. This must be a function that must have 3 parameters: filename, heightMult, and widthMult. It should open the desired plotting device. For advanced users only.

`closePlottingDeviceFunc`
: An R function. This must be used in conjunction with openPlottingDeviceFunc. For most devices, you can just use the function "dev.off". For advanced users only.

`minimalImageFilenames`
: Logical. If `TRUE`, then the image files will not include the gene names. By default this is `FALSE`, but may be set to `TRUE` when this function is called internally by `buildAllPlots` using the default options. This option is sometimes needed so that the image files are compatible with certain web hosts that have file name length restrictions.

`verbose`
: if `TRUE`, send debugging and progress messages to the console / stdout.

`debug.mode`
: if TRUE, send even more debugging and progress messages to the console / stdout.

`INTERNAL.VARS`
: NOT FOR GENERAL USE. Intended only for use by JunctionSeq itself, internally. This is used for passing pre-generated data (when generating many similar plots, for example), and for internally-generated parameters. DO NOT USE.

`...`
: Additional options to pass to `plotJunctionSeqResultsForGene`, or graphical parameters passed to plotting functions.

Value

This is a side-effecting function, and does not return a value.

Examples

```
data(exampleDataSet,package="JctSeqData");
buildAllPlotsForGene(geneID = "ENSRNOG00000009281", jscs);

## Not run:
########################################
```

```
#Set up example data:
decoder.file <- system.file(
                  "extdata/annoFiles/decoder.bySample.txt",
                  package="JctSeqData");
decoder <- read.table(decoder.file,
                  header=TRUE,
                  stringsAsFactors=FALSE);
gff.file <- system.file(
            "extdata/cts/withNovel.forJunctionSeq.gff.gz",
            package="JctSeqData");
countFiles <- system.file(paste0("extdata/cts/",
     decoder$sample.ID,
     "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
     package="JctSeqData");
########################################
#Run example analysis:
jscs <- runJunctionSeqAnalyses(sample.files = countFiles,
           sample.names = decoder$sample.ID,
           condition=factor(decoder$group.ID),
           flat.gff.file = gff.file,
           analysis.type = "junctionsAndExons"
);
########################################

#Generate several related plots for the selected gene:
buildAllPlotsForGene(geneID = "ENSRNOG00000009281", jscs);


## End(Not run)
```

`defaultColorList` *JunctionSeq Color Parameters*

Description

This data set specifies some of the default color and style parameters for JunctionSeq. Any of these parameters can be overridden by using the `colorList` parameter in `buildAllPlots`, `buildAllPlotsForGene`, or `plotJunctionSeqResultsForGene`.

Usage

```
JUNCTIONSEQ.DEFAULT.COLOR.LIST
```

Format

A list.

Details

The parameter `colorList` must be a named list. Any elements with the names listed below will override the default JunctionSeq colors.

`SIG.FEATURE.COLOR` The color of vertical lines in the plotting panel, for significant features.

NOSIG.FEATURE.COLOR The color of vertical lines in the plotting panel, for tested but non-significant features.

UNTESTABLE.FEATURE.COLOR The color of vertical lines in the plotting panel, for untestable features.

EXCLUDED.FEATURE.COLOR The color of vertical lines in the plotting panel, for features that were not part of the analysis (for example: splice junctions in an exon-based analysis).

SIG.VERTLINE.COLOR,NOSIG.VERTLINE.COLOR,UNTESTABLE.VERTLINE.COLOR: The colors of the vertical dotted lines in the plotting panel.

SIG.FEATURE.BORDER.COLOR, NOSIG.FEATURE.BORDER.COLOR, UNTESTABLE.FEATURE.BORDER.COLOR, EXCLUDED.FEATURE.BORDER.COLOR: The color for borders of features in the gene and transcript annotation.

SIG.FEATURE.FILL.COLOR, NOSIG.FEATURE.FILL.COLOR, UNTESTABLE.FEATURE.FILL.COLOR, EXCLUDED.FEATURE.F The colors for exon shading in the gene and transcript annotation.

KNOWN.SPLICE.LTY, NOVEL.SPLICE.LTY: These set the "lty" parameter for the known and novel splice junction diagrams. These must be specified as one of the character specifications. In other words, either a descriptive specification like "solid","dotted",etc., or by an even number of nonzero hexadecimal digits. See the section "Line Type Specification" in the graphical parameters help (linkpar). You CANNOT use the single-digit index specification, as this will fail when used in vector form with the others (technically you could use them, but only if you reset ALL lty parameters).

EXON.CONNECTION.LTY, NOVEL.SPLICE.CONNECTION.LTY, KNOWN.SPLICE.CONNECTION.LTY: The "lty" parameters for the lines that connect the features on the gene schematic diagram to the plotting panel. These have the same requirements as the previous set of lty parameters.

PLOTTING.LINE.COLORS Character vector. The colors for each condition, in order. Will be shortened to the number of conditions.

Value

See above.

Examples

```
data(exampleDataSet,package="JctSeqData");

#Set a few alternative colors:
buildAllPlotsForGene(geneID = "ENSRNOG00000009281", jscs,
                     outfile.prefix = "./oddColors.",
                     colorList = list(SIG.FEATURE.COLOR = "red",
                                      SIG.FEATURE.FILL.COLOR = "green",
                                      NOSIG.FEATURE.FILL.COLOR = "blue"
                                      ));
```

---

estimateEffectSizes *Estimate Effect Sizes, parameter estimates, etc.*

Description

This function runs fits another generalized linear model to the data, this one intended for use in estimating the effect sizes and expression estimates for each analysis.

This function is called internally by the runJunctionSeqAnalyses function, and thus for most purposes users should not need to call this function directly. It may be useful to advanced users performing non-standard analyses.

Usage

```
estimateEffectSizes(jscs,
   method.expressionEstimation = c("feature-vs-gene",
                                   "feature-vs-otherFeatures"),
   effect.formula = formula(~ condition + countbin + condition : countbin),
   geneLevel.formula = formula(~ condition),
   calculate.geneLevel.expression = TRUE,
   keep.estimation.fit = FALSE,
   nCores=1,
   dispColumn="dispersion",
   verbose = TRUE)
```

Arguments

`jscs`	A `JunctionSeqCountSet`. Usually initially created by `readJunctionSeqCounts`. Size factors must be set, usually using functions `estimateSizeFactors` and `estimateJunctionSeqDispersions`.
`method.expressionEstimation`	Character string. Can be used to apply alternative methodologies or implementations. Intended for advanced users who have strong opinions about the underlying statistical methodologies. Determines the methodology used to generate feature expression estimates and relative fold changes. By default each feature is modeled separately. Under the default count-vector method, this means that the resultant relative fold changes will be a measure of the relative fold change between the feature and the gene as a whole. Alternatively, the "feature-vs-otherFeatures" method builds a large, complex model containing all features belonging to the gene. The coefficients for each feature are then "balanced" using linear contrasts weighted by the inverse of their variance. In general we have found this method to produce very similar results but less efficiently and less consistently. Additionally, this alternative method "multi-counts" reads that cover more than one feature. This can result in over-weighting of exonic regions with a large number of annotated variations in a small genomic area, as each individual read or read-pair may be counted many times in the model. Under the default option, no read or read-pair is ever counted more than once in a given model.
`effect.formula`	For advanced users. The base formula for the model used for effect size estimation. NOTE: the biological condition to be tested must be named "condition".
`geneLevel.formula`	For advanced users. The base formula for the model used to estimate total gene-level expression. NOTE: the biological condition to be tested must be named "condition".
`calculate.geneLevel.expression`	Logical value. If `TRUE`, gene-level expression will be estimated using the same maximum-likelihood method used in other analyses. Default: `TRUE`.
`keep.estimation.fit`	Logical value. If `TRUE`, save the complete model fits for every gene. This will require a lot of memory, but may be useful for statistical diagnostics. Default: `FALSE`.

nCores	Either an integer or a BiocParallelParam object. Either way, this determines The number of cores to use. Note that multicore functionality may not be available on all platforms. If parallel execution is not available then JunctionSeq will automatically fallback to single-core execution. See the BiocParallel package for more information.
dispColumn	Character value. The name of the fData(jscs) column in which the model dispersion is stored.
verbose	if TRUE, send debugging and progress messages to the console / stdout.

Value

A JunctionSeqCountSet, with effect size results included.

Examples

```
data(exampleDataSet,package="JctSeqData");
jscs <- estimateEffectSizes(jscs);

## Not run:
#Full example (from scratch):

########################################
#Set up example data:
decoder.file <- system.file(
                  "extdata/annoFiles/decoder.bySample.txt",
                  package="JctSeqData");
decoder <- read.table(decoder.file,
                  header=TRUE,
                  stringsAsFactors=FALSE);
gff.file <- system.file(
            "extdata/cts/withNovel.forJunctionSeq.gff.gz",
            package="JctSeqData");
countFiles <- system.file(paste0("extdata/cts/",
     decoder$sample.ID,
     "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
     package="JctSeqData");
########################################
#Advanced Analysis:

#Make a "design" dataframe:
design <- data.frame(condition = factor(decoder$group.ID));
#Read the QoRTs counts.
jscs = readJunctionSeqCounts(countfiles = countFiles,
           samplenames = decoder$sample.ID,
           design = design,
           flat.gff.file = gff.file
);
#Generate the size factors and load them into the JunctionSeqCountSet:
jscs <- estimateJunctionSeqSizeFactors(jscs);
#Estimate feature-specific dispersions:
jscs <- estimateJunctionSeqDispersions(jscs);
#Fit dispersion function and estimate MAP dispersion:
jscs <- fitJunctionSeqDispersionFunction(jscs);
#Test for differential usage:
jscs <- testForDiffUsage(jscs);
#Estimate effect sizes and expression estimates:
```

```
jscs <- estimateEffectSizes( jscs);


## End(Not run)
```

`estimateJunctionSeqDispersions` *JunctionSeq Dispersion Estimation*

Description

This method estimates the sample dispersion for each counting bin (in other words, each splice junction locus).

This function is called internally by the `runJunctionSeqAnalyses` function, and thus for most purposes users should not need to call this function directly. It may be useful to advanced users performing non-standard analyses.

Usage

```
estimateJunctionSeqDispersions( jscs,
    method.GLM = c(c("advanced","DESeq2-style"),
                   c("simpleML","DEXSeq-v1.8.0-style")),
    test.formula1 = formula(~ sample + countbin + condition : countbin),
    meanCountTestableThreshold="auto", nCores=1,
    use.multigene.aggregates = FALSE,
    verbose = TRUE)
```

Arguments

`jscs`	A `JunctionSeqCountSet`. Usually initially created by `readJunctionSeqCounts`. Size factors must be set, usually using functions `estimateSizeFactors` and `estimateJunctionSeqDispersions`.
`method.GLM`	Character string. Can be used to apply alternative methodologies or implementations. Intended for advanced users who have strong opinions about the underlying statistical methodologies. The default is "advanced" or, equivalently, "DESeq2-style". This uses the dispersion estimation methodology used by DESeq2 and DEXSeq v1.12.0 or higher to generate the initial (feature-specific) dispersion estimates. The alternative method is "simpleML" or, equivalently, "DEXSeq-v1.8.0-style". This uses a simpler maximum-likelihood-based method used by the original DESeq and by DEXSeq v1.8.0 or less.
`test.formula1`	The model formula. Note that this formula is different from the formula used to calculate parameter estimates and effect size. This is because the two noise components (gene-level and countbin-level noise) are folded into the sample term. Since we only intend to test the condition-countbin interaction, we do not need to model the gene-level differential expression. NOTE: the biological condition to be tested MUST be named "condition".

meanCountTestableThreshold
: "auto" or Numeric value. Features with a total mean normalized count of less than this value will be excluded from the analyses. If left as the default ("auto"), then the cutoff threshold will be determined automatically using the DESeq2 independent filtering method.

nCores
: Either an integer or a BiocParallelParam object. Either way, this determines The number of cores to use. Note that multicore functionality may not be available on all platforms. If parallel execution is not available then JunctionSeq will automatically fallback to single-core execution. See the BiocParallel package for more information.

use.multigene.aggregates
: Logical value. Whether to attempt to test "aggregate genes" which consist of multiple genes that overlap with one another. Note that inclusion of aggregate genes may affect the false discovery rate, since by their very nature aggregate genes will often show differential splice junction usage, as the two genes will often be regulated independently.

verbose
: A boolean flag indicating whether or not to print progress information during execution. (Default=FALSE)

Value

A JunctionSeqCountSet, with dispersion results included.

Examples

```
## Not run:
#Full example (from scratch):
########################################
#Set up example data:
decoder.file <- system.file(
                  "extdata/annoFiles/decoder.bySample.txt",
                  package="JctSeqData");
decoder <- read.table(decoder.file,
                  header=TRUE,
                  stringsAsFactors=FALSE);
gff.file <- system.file(
            "extdata/cts/withNovel.forJunctionSeq.gff.gz",
            package="JctSeqData");
countFiles <- system.file(paste0("extdata/cts/",
     decoder$sample.ID,
     "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
     package="JctSeqData");
########################################
#Advanced Analysis:

#Make a "design" dataframe:
design <- data.frame(condition = factor(decoder$group.ID));
#Read the QoRTs counts.
jscs = readJunctionSeqCounts(countfiles = countFiles,
           samplenames = decoder$sample.ID,
           design = design,
           flat.gff.file = gff.file
);
#Generate the size factors and load them into the JunctionSeqCountSet:
jscs <- estimateJunctionSeqSizeFactors(jscs);
```

```
#Estimate feature-specific dispersions:
jscs <- estimateJunctionSeqDispersions(jscs);
#Fit dispersion function and estimate MAP dispersion:
jscs <- fitJunctionSeqDispersionFunction(jscs);
#Test for differential usage:
jscs <- testForDiffUsage(jscs);
#Estimate effect sizes and expression estimates:
jscs <- estimateEffectSizes( jscs);


## End(Not run)
```

`estimateJunctionSeqSizeFactors` *Estimate Size Factors*

Description

Estimate size factors, which are scaling factors used as "offsets" by the statistical model to make the different samples comparable. This is necessary because the different samples may have been sequenced to slightly different depths. Additionally, the presence of differentially expressed genes may cause the apparent depth of many genes to appear different.

This function uses the "geometric" size factor normalization method, which is identical to the one used by DESeq, DESeq2, DEXSeq, and the default method used by CuffDiff.

This function is called internally by the `runJunctionSeqAnalyses` function, and thus for most purposes users should not need to call this function directly. It may be useful to advanced users performing non-standard analyses.

Usage

```
estimateJunctionSeqSizeFactors(jscs,
        method.sizeFactors = c("byGenes","byCountbins"),
        replicateDEXSeqBehavior.useRawBaseMean = FALSE,
        calcAltSF = TRUE,
        verbose = FALSE);

writeSizeFactors(jscs, file);
```

Arguments

`jscs`	A `JunctionSeqCountSet`. Usually initially created by `readJunctionSeqCounts`. Size factors must be set, usually using functions `estimateSizeFactors` and `estimateJunctionSeqDispersions`.
`method.sizeFactors`	Character string. Can be used to apply alternative methodologies or implementations. Intended for advanced users who have strong opinions about the underlying statistical methodologies. Determines the method used to calculate normalization size factors. By default JunctionSeq uses gene-level expression. As an alternative, feature-level counts can be used as they are in DEXSeq. In practice the difference is almost always negligible.

`replicateDEXSeqBehavior.useRawBaseMean`
: USED ONLY FOR INTERNAL TESTING! NOT INTENDED FOR ACTUAL USE!

  This variable activates an alternative mode in which a (very minor) bug in DEXSeq v1.14.0 and earlier is replicated. If `TRUE`, the baseMean and baseVar variables will be computed using raw counts rather than normalized counts. This is used for internal tests in which DEXSeq functionality is replicated precisely and the results are compared against equivalent DEXSeq results. Without this option the results would differ slightly (generally by less than 1 hundreth of a percent).

  USED ONLY FOR INTERNAL TESTING! NOT INTENDED FOR ACTUAL USE!

`calcAltSF`
: Logical. Determines whether both types of size factor calculations should be generated, and placed in the `jscs@altSizeFactors` slot.

`verbose`
: if `TRUE`, send debugging and progress messages to the console / stdout.

`file`
: A file path to write the size factor table.

`...`
: If using the (depreciated) estimateSizeFactors command, use the same syntax as above.

Value

A JunctionSeqCountSet, with size factors included.

Examples

```
data(exampleDataSet,package="JctSeqData");
jscs <- estimateJunctionSeqSizeFactors(jscs);

## Not run:
########################################
#Set up example data:
decoder.file <- system.file(
                  "extdata/annoFiles/decoder.bySample.txt",
                  package="JctSeqData");
decoder <- read.table(decoder.file,
                  header=TRUE,
                  stringsAsFactors=FALSE);
gff.file <- system.file(
             "extdata/cts/withNovel.forJunctionSeq.gff.gz",
             package="JctSeqData");
countFiles <- system.file(paste0("extdata/cts/",
     decoder$sample.ID,
     "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
     package="JctSeqData");
########################################
#Advanced Analysis:

#Make a "design" dataframe:
design <- data.frame(condition = factor(decoder$group.ID));
#Read the QoRTs counts.
jscs = readJunctionSeqCounts(countfiles = countFiles,
           samplenames = decoder$sample.ID,
           design = design,
           flat.gff.file = gff.file
```

```
);
#Generate the size factors and load them into the JunctionSeqCountSet:
jscs <- estimateJunctionSeqSizeFactors(jscs);
#Estimate feature-specific dispersions:
jscs <- estimateJunctionSeqDispersions(jscs);
#Fit dispersion function and estimate MAP dispersion:
jscs <- fitJunctionSeqDispersionFunction(jscs);
#Test for differential usage:
jscs <- testForDiffUsage(jscs);
#Estimate effect sizes and expression estimates:
jscs <- estimateEffectSizes( jscs);


## End(Not run)
```

`fitJunctionSeqDispersionFunction` *Fit Shared Dispersion Function*

Description

Fit dispersion function to share dispersion information between features across the genome.

This function is called internally by the `runJunctionSeqAnalyses` function, and thus for most purposes users should not need to call this function directly. It may be useful to advanced users performing non-standard analyses.

Usage

```
fitJunctionSeqDispersionFunction(jscs,
    method.GLM = c(c("advanced","DESeq2-style"),
                   c("simpleML","DEXSeq-v1.8.0-style")),
    method.dispFit = c("parametric", "local", "mean"),
    method.dispFinal = c("shrink","max","fitted","noShare"),
    fitDispersionsForExonsAndJunctionsSeparately = TRUE,
    verbose = TRUE)
```

Arguments

`jscs`	A `JunctionSeqCountSet`. Usually initially created by `readJunctionSeqCounts`. Size factors must be set, usually using functions `estimateSizeFactors` and `estimateJunctionSeqDispersions`.
`method.GLM`	Character string. Can be used to apply alternative methodologies or implementations. Intended for advanced users who have strong opinions about the underlying statistical methodologies. The default is "advanced" or, equivalently, "DESeq2-style". This uses the dispersion estimation methodology used by DESeq2 and DEXSeq v1.12.0 or higher to generate the initial (feature-specific) dispersion estimates. The alternative method is "simpleML" or, equivalently, "DEXSeq-v1.8.0-style". This uses a simpler maximum-likelihood-based method used by the original DESeq and by DEXSeq v1.8.0 or less.

`method.dispFit` Character string. Can be used to apply alternative methodologies or implementations. Intended for advanced users who have strong opinions about the underlying statistical methodologies.

Determines the method used to generated "fitted" dispersion estimates. One of "parametric" (the default), "local", or "mean". See the DESeq2 documentation for more information.

`method.dispFinal`

Character string. Can be used to apply alternative methodologies or implementations. Intended for advanced users who have strong opinions about the underlying statistical methodologies.

Determines the method used to arrive at a "final" dispersion estimate. The default, "shrink" uses the maximum a posteriori estimate, combining information from both the fitted and feature-specific dispersion estimates. This is the method used by DESeq2 and DEXSeq v1.12.0 and above.

`fitDispersionsForExonsAndJunctionsSeparately`

When running a "junctionsAndExons" type analysis in which both exons and splice junctions are being tested simultaniously, this parameter determines whether a single fitted dispersion model should be fitted for both exons and splice junctions, or if separate fitted dispersions should be calculated for each. By default the dispersions are run separately.

`verbose` if TRUE, send debugging and progress messages to the console / stdout.

`...` If using the depreciated fitDispersionFunction command, use the same syntax as above.

Value

A JunctionSeqCountSet, with dispersion results included.

Examples

```
data(exampleDataSet,package="JctSeqData");
jscs <- fitJunctionSeqDispersionFunction(jscs);

## Not run:
#Full example (from scratch):
########################################
#Set up example data:
decoder.file <- system.file(
                  "extdata/annoFiles/decoder.bySample.txt",
                  package="JctSeqData");
decoder <- read.table(decoder.file,
                  header=TRUE,
                  stringsAsFactors=FALSE);
gff.file <- system.file(
            "extdata/cts/withNovel.forJunctionSeq.gff.gz",
            package="JctSeqData");
countFiles <- system.file(paste0("extdata/cts/",
     decoder$sample.ID,
     "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
     package="JctSeqData");
########################################
#Advanced Analysis:

#Make a "design" dataframe:
```

```
design <- data.frame(condition = factor(decoder$group.ID));
#Read the QoRTs counts.
jscs = readJunctionSeqCounts(countfiles = countFiles,
           samplenames = decoder$sample.ID,
           design = design,
           flat.gff.file = gff.file
);
#Generate the size factors and load them into the JunctionSeqCountSet:
jscs <- estimateJunctionSeqSizeFactors(jscs);
#Estimate feature-specific dispersions:
jscs <- estimateJunctionSeqDispersions(jscs);
#Fit dispersion function and estimate MAP dispersion:
jscs <- fitJunctionSeqDispersionFunction(jscs);
#Test for differential usage:
jscs <- testForDiffUsage(jscs);
#Estimate effect sizes and expression estimates:
jscs <- estimateEffectSizes( jscs);


## End(Not run)
```

`JunctionSeqCountSet-class`
Class `"JunctionSeqCountSet"`

Description

A `JunctionSeqCountSet` is a container class that contains all information pertaining to a JunctionSeq analysis and dataset. In general, these methods and slots will not be used by the end-users. In general, JunctionSeqCountSet objects will be created by `readJunctionSeqCounts` or `runJunctionSeqAnalyses` and are to be manipulated byhigh-level JunctionSeq functions such as `estimateEffectSizes` or `fitJunctionSeqDispersionFunction`.

The methods documented here are for use by advanced users only.

Details

Slots:

`designColumns` A character vector with the column names in the `design` data.frame.

`dispFitCoefs` The dispersion fit coefficients.

`fittedMu` Fitted mu values generated by DESeq2 code.

`dispFunctionType` A list of various variables defining the dispersion functon used.

`dispFunction` A function that converts a base mean to a fitted dispersion based on all included count bins.

`dispFunctionJct` A function that converts a base mean to a fitted dispersion based only on the splice junction bins.

`dispFunctionExon` A function that converts a base mean to a fitted dispersion based only on the exon bins.

`formulas` A list of formulas used.

`annotationFile` The annotation file.

`geneCountData` A matrix of the gene-level counts

`countVectors` A matrix of the count vectors.

`altSizeFactors` (Not currently used)

`plottingEstimates` A list of fitted estimates, for plotting.

`plottingEstimatesVST` (Not currently used)

`geneLevelPlottingEstimates` A list of gene-level fitted estimates, for plotting.

`modelFitForHypothesisTest` (Not currently used)

`modelFitForEffectSize` (Not currently used)

`flatGffData` A `data.frame` representation of the flattened gff annotation for each countbin.

`flatGffGeneData` A `data.frame` representation of the flattened gff annotation for each gene.

`analysisType` The type of analysis. Character string.

`DESeqDataSet` The specially-constructed `DESeqDataSet`, to be passed to the internally-loaded DESeq2 code.

`modelCoefficientsSample`**:** Object of class `"list"`. Placeholder slot for model coefficients (used for diagnostic testing of code).

`modelCoefficientsGene`**:** Object of class `"list"`. Placeholder slot for model coefficients (used for diagnostic testing of code).

`assayData`**:** Object of class `"AssayData"`. Contains various data.

`phenoData`**:** Object of class `"AnnotatedDataFrame"`. Phenotype data.

`featureData`**:** Object of class `"AnnotatedDataFrame"`. Counting bin data.

`experimentData`**:** Object of class `"MIAxE"`. Information on the experiment.

`annotation`**:** Object of class `"character"`. Not used.

`protocolData`**:** Object of class `"AnnotatedDataFrame"`. Information on the code.

`.__classVersion__`**:** Object of class `"Versions"`. The version of the JunctionSeqCountSet.

Constructor

`newJunctionSeqCountSet(countData,geneCountData,design,geneIDs,countbinIDs,featureIntervals=NULL`
Creates a new JunctionSeqCountSet

`countData` A matrix of junction-level count data of non-negative integer values. The rows correspond to counts for each splice-junction counting bin, the columns correspond to samples. Note that biological replicates should each get their own column, while the counts of technical replicates (i.e., several sequencing runs/lanes from the same sample) should be summed up into a single column.

`geneCountData` A matrix of gene-level count data of non-negative integer values. The rows correspond to counts for each gene, the columns correspond to samples. Note that biological replicates should each get their own column, while the counts of technical replicates (i.e., several sequencing runs/lanes from the same sample) should be summed up into a single column. Must have the same dimensions as countData.

`design` A data frame consisting of all factors to be included in the analysis. All columns should be factors. Each column should represent a different variable, each row should represent a different sample. The number of rows must equal the number of columns in geneCountData and countData.

`geneIDs` A character vector of gene indentifiers for each splice junction. The length must equal the number of rows in countData.

`countbinIDs` A character vector of splice-junction-locus indentifiers for each splice junction. The length must equal the number of rows in countData.

`featureIntervals` Optional. A data.frame with 4 columns: "chr", "start", "end", and "strand". chr and strand should be character vectors or factors, start and end must be integers.

`transcripts` Optional. Character vector listing the transcripts that each splice junction belongs to. Some junctions may belong to more than one transcripts. In this case, transcripts should be separated with the "+" character.

This constructor function SHOULD NOT BE USED in normal operation. Instead you should use the `readJunctionSeqCounts` function, which returns a new `JunctionSeqCountSet`.

Extends

Class "`eSet`", directly. Class "`VersionedBiobase`", by class "eSet", distance 2. Class "`Versioned`", by class "eSet", distance 3.

Note

End-users generally will not use any of these slots or methods directly. However, they may be useful for model fit diagnostics and similar statistical experimentation.

You can access method-mode information using the "AltMethods" attribute, and a list of all calls using the "callStack" attribute.

Author(s)

Stephen Hartley

See Also

The proper way to create a JunctionSeqCountSet is to use `readJunctionSeqCounts` or `runJunctionSeqAnalyses`.

Examples

```
showClass("JunctionSeqCountSet")
```

`plotDispEsts` *Plot Fitted and Test-wise Dispersion*

Description

Plots the countbin-specific estimated dispersion and the fitted dispersion curve.

Usage

```
plotDispEsts( jscs, ylim, xlim,
              linecol=c("#0000FF","#FF0000"),
              pointcol = c("#00009980","#99000080"),
              title.main = "Dispersion Estimates",
              xlab = "Mean Normalized Coverage",
              ylab = "Dispersion",
              miniTicks = TRUE,
              pch.MLE = 46, pch.MAP = 1, lwd.fitted = 2,
```

```
par.cex = 1, points.cex = 1, text.cex = 1, lines.cex = 8,
use.smoothScatter = FALSE, smooth.nbin = 512, nrpoints = 100,
plot.exon.results = TRUE,
plot.junction.results = TRUE,
anno.lwd = 2,
mar = c(4.1,4.1,3.1,1.1),
show.legends = TRUE,
verbose = TRUE, debug.mode = FALSE,
... )
```

Arguments

jscs	A JunctionSeqCountSet. Usually created by runJunctionSeqAnalyses. Alternatively, this can be created manually by readJunctionSeqCounts. Dispersions and size factors must then be set, usually using functions estimateSizeFactors and estimateJunctionSeqDispersions. Hypothesis tests must be performed by testForDiffUsage.
ylim	The plotting range for the y-axis.
xlim	The plotting range for the x-axis.
linecol	Character vector of length 2. The line color to use for the fit line. If the fits were performed separately for exons and junctions, the junction line will be drawn with the second color.
pointcol	Character vector of length 2. The point color to use for the final dispersions. If the fits were performed separately for exons and junctions, the junction points will be drawn with the second color.
title.main	The main title of the plot.
xlab	The label for the x-axis.
ylab	The label for the y-axis.
miniTicks	Whether or not to plot smaller ticks at the tenth-decades.
par.cex	The base cex value to be passed to par() immediately before all plots are created. See par.
points.cex	The character expansion value for the plotted points.
text.cex	The character expansion value for the annotation text (labels, etc).
lines.cex	The character expansion value for lines. What this means seems to vary depending on the plotting device.
pch.MLE	Numeric. The pch code for the MLE (ie single-feature) dispersion estimate. The default is a small point.
pch.MAP	Numeric. The pch code for the MAP (ie. final) dispersion estimate. The default is a circle.
lwd.fitted	Numeric. The width of the dispersion fit line(s).
use.smoothScatter	Logical. If TRUE, features will be ploted with density shading rather than having each point plotted.
smooth.nbin	The number of bins to smooth, for the density plot, if use.smoothScatter is TRUE.
nrpoints	The number of extra points to plot, if use.smoothScatter is TRUE.

plot.exon.results
: Logical. If TRUE, plot results for exons. Technically speaking, JunctionSeq can be used to do DEXSeq-style analyses on exon partitions. However this functionality is for advanced users only.

plot.junction.results
: Logical. If TRUE, plot results for splice junctions. For advanced users only.

anno.lwd
: The lwd value to be passed to lines, box, axis, and similar.

mar
: The margin sizes, expressed in lines. see link{par}.

show.legends
: Logical. If TRUE, display legends.

verbose
: if TRUE, send debugging and progress messages to the console / stdout.

debug.mode
: if TRUE, send even more debugging and progress messages to the console / stdout.

...
: Additional options to pass to plotting functions, particularly graphical parameters.

Value

This is a side-effecting function, and does not return a value.

Examples

```
data(exampleDataSet,package="JctSeqData");
plotDispEsts(jscs);

## Not run:
########################################
#Set up example data:
decoder.file <- system.file(
                  "extdata/annoFiles/decoder.bySample.txt",
                  package="JctSeqData");
decoder <- read.table(decoder.file,
                  header=TRUE,
                  stringsAsFactors=FALSE);
gff.file <- system.file(
            "extdata/cts/withNovel.forJunctionSeq.gff.gz",
            package="JctSeqData");
countFiles <- system.file(paste0("extdata/cts/",
     decoder$sample.ID,
     "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
     package="JctSeqData");
######################
#Run example analysis:
jscs <- runJunctionSeqAnalyses(sample.files = countFiles,
           sample.names = decoder$sample.ID,
           condition=factor(decoder$group.ID),
           flat.gff.file = gff.file,
           analysis.type = "junctionsAndExons"
);
########################################

#Plot dispersions:
plotDispEsts(jscs);


## End(Not run)
```

plotJunctionSeqResultsForGene
Generate a JunctionSeq expression plot.

Description

Creates one results plot for one gene. Note that this function does not call a plotting device, so it will simply plot to the "current" device. If you want to automatically save images to file, use buildAllPlotsForGene, which internally calls this function.

Note that this function has MANY parameters, allowing the user to tweak the appearance of the plots to suit their particular needs and preferences. Don't be daunted: the default parameters are probably fine for most purposes.

Usage

```
plotJunctionSeqResultsForGene(geneID, jscs,
    colorRed.FDR.threshold=0.01,
    plot.type = c("expr","normCounts","rExpr","rawCounts"),
    sequencing.type = c("paired-end","single-end"),
    displayTranscripts = FALSE,
    colorList = list(),
    use.vst = FALSE, use.log = TRUE,
    exon.rescale.factor = 0.3,
    exonRescaleFunction = c("sqrt","log","linear","34root"),
    label.p.vals = TRUE,
    plot.lwd = 3, axes.lwd = plot.lwd,
    anno.lwd = plot.lwd, gene.lwd = plot.lwd / 2,
    par.cex = 1, anno.cex.text = 1,
    anno.cex.axis=anno.cex.text, anno.cex.main = anno.cex.text * 1.2,
    cex.arrows = "auto",
    fit.countbin.names = TRUE, fit.genomic.axis = TRUE, fit.labels = TRUE,
    plot.gene.level.expression = TRUE,
    plot.exon.results, plot.junction.results, plot.novel.junction.results,
    plot.untestable.results = FALSE, draw.untestable.annotation = TRUE,
    show.strand.arrows = 1,
    sort.features = TRUE,
    drawCoordinates = TRUE,
    yAxisLabels.inExponentialForm = FALSE,
    title.main, title.ylab, title.ylab.right,
    graph.margins = c(2, 3, 3, 3),
    GENE.annotation.relative.height = 0.15,
    TX.annotation.relative.height = 0.05,
    CONNECTIONS.relative.height = 0.1,
    SPLICE.annotation.relative.height = 0.1,
    TX.margins = c(0,0.5),
    condition.legend.text = NULL, include.TX.names = TRUE,
    draw.start.end.sites = TRUE,
    label.chromosome = TRUE,
    splice.junction.drawing.style = c("hyperbola","ellipse",
                                      "triangular","line"),
```

```
    draw.nested.SJ = TRUE, merge.exon.parts = TRUE,
    verbose=TRUE, debug.mode = FALSE,
    INTERNAL.VARS = list(),
    ...)
```

Arguments

geneID	Character string. The gene to the plotted.
jscs	A JunctionSeqCountSet. Usually created by runJunctionSeqAnalyses. Alternatively, this can be created manually by readJunctionSeqCounts. However in this case a number of additional steps will be necessary: Dispersions and size factors must then be set, usually using functions estimateSizeFactors and estimateJunctionSeqDispersions. Hypothesis tests must be performed by testForDiffUsage. Effect sizes and parameter estimates must be created via estimateEffectSizes.
colorRed.FDR.threshold	The adjusted-p-value threshold used to determine whether a feature should be marked as "significant" and colored pink. By default this will be the same as the FDR.threshold.
plot.type	Character string. Determines which plot to produce. Options are: "expr" for "expression", or mean normalized read counts by experimental condition, "rExpr" for "relative" expression relative to gene-level expression, "normCounts" for normalized read counts for each sample, and "rawCounts" for raw read counts for each sample.
sequencing.type	The type of sequencing used, either "paired-end" or "single-end". This only affects the labelling of the y-axis, and does not affect the actual plots in any way.
displayTranscripts	Logical. If true, then the full set of annotated transcripts will be displayed below the expression plot (to a maximum of 42 different TX).
colorList	A named list of R colors, setting the colors used for various things. See junctionSeqColors for more information.
use.vst	Logical. If TRUE, all plots will be scaled via a variance stabilizing transform.
use.log	Logical. If TRUE, all plots will be log-scaled.
exon.rescale.factor	Numeric. Exons will be proportionately scaled-up so that the exonic regions make up this fraction of the horizontal plotting area. If negative, exons and introns will be plotted to a common scale.
exonRescaleFunction	Character string. Exonic and intronic regions will be rescaled to be proportional to this transformation of their span. By default the square-root function is used, which shrinks long features and extends short features so that they are all still readable and destinguishable against one another. This default option seems to behave well on mammalian genomes. This parameter does nothing if exon.rescale.factor is negative.
label.p.vals	Logical. If TRUE, then statistically significant p-values will be labelled.
plot.lwd	the line width for the plotting lines.
axes.lwd	the line width for the axes.

`anno.lwd` the line width for the various other annotation lines.

`gene.lwd` the line width used for the gene annotation lines.

`par.cex` The base cex value to be passed to par() immediately before all plots are created. See `par`.

`anno.cex.text` The font size multiplier for most annotation text. This will be multiplied by a factor of the par.cex value. More specifically: The cex value to be passed to all function calls that take graphical parameters. See `par`.

`anno.cex.axis` The font size multiplier for the axis text. This will be multiplied by a factor of the par.cex value. More specifically: The cex.axis value to be passed to all function calls that take graphical parameters. See `par`.

`anno.cex.main` The font size multiplier for the main title text. This will be multiplied by a factor of the par.cex value. More specifically: The cex.main value to be passed to all function calls that take graphical parameters. See `par`.

`cex.arrows` The font size for the strand-direction arrows in the gene annotation region. The arrows will be sized to equal the dimensions of the letter "M" at this font size.

`fit.countbin.names`
Logical. If `TRUE`, then splice-junction-locus labels should be rescaled to fit in whatever horizontal space is available.

`fit.genomic.axis`
Logical. If `TRUE`, then the genomic coordinate labels will be auto-scaled down to fit, if needed.

`fit.labels` Logical. If `TRUE`, then y-axis labels will be auto-scaled down to fit, if needed. Note this only applies to the text labels, not the numeric scales.

`plot.gene.level.expression`
Logical value. If `TRUE`, gene-level expression (when applicable) will be plotted beside the sub-element-specific expression in a small seperate plotting box. For the "relative expression" plots the simple mean normalized expression will be plotted (since it doesn't make sense to plot something relative to itself).

`plot.exon.results`
Logical. If `TRUE`, plot results for exons. By default everything that was tested will be plotted.

`plot.junction.results`
Logical. If `TRUE`, plot results for splice junctions. By default everything that was tested will be plotted.

`plot.novel.junction.results`
Logical. If `TRUE`, plot results for novel splice junctions. If false, novel splice junctions will be ignored. By default everything that was tested will be plotted.

`plot.untestable.results`
Logical. If TRUE, plots the expression of splice junctions that had coverage that was too low to be tested.

`draw.untestable.annotation`
Logical. If TRUE, draws the annotation for splice junctions that had coverage that was too low to be tested.

`show.strand.arrows`
The number of strand-direction arrows to display. If equal to `1` (the default) then the arrow will extend from the end of the gene drawing, if it is greater than `1` then arrows will be drawn along the gene length. If it is `0` or `NA` then arrows will not be drawn.

`sort.features` Logical. If `TRUE`, sort features by genomic position.

`drawCoordinates`
: Whether to label the genomic coordinates at the bottom of the plot.

`yAxisLabels.inExponentialForm`
: Logical. If `TRUE`, then the y-axis will be labelled in exponential form.

`graph.margins`
: Numeric vector of length 4. These margins values used (as if for par("mar")) for the main graph. The lower part of the plot uses the same left and right margins.

`GENE.annotation.relative.height`
: The height of the "gene track" displayed underneath the main graph, relative to the height of the main graph. By default it is 20 percent.

`TX.annotation.relative.height`
: For all plots that draw the annotated-transcript set (when the `with.TX` parameter is `TRUE`), this sets the height of each transcript, as a fraction of the height of the main graph. By default it is 2.5 percent.

`CONNECTIONS.relative.height`
: The height of the panel that connects the plotting area to the gene annotation area, relative to the height of the plotting area. This panel has the lines that connects the counting bin columns to their actual loci on the gene. By default it is 10 percent.

`SPLICE.annotation.relative.height`
: The height of the area that shows the splice junction loci, relative to the size of the plotting area.

`TX.margins`
: A numeric vector of length 2. The size of the blank space between the gene plot and the transcript list and then beneath the transcript list, relative to the size of each transcript line.

`title.main`
: Character string. Overrides the default main plot title.

`title.ylab`
: Character string. Overrides the default y-axis label for the left y-axis.

`title.ylab.right`
: Character string. Overrides the default y-axis label for the right y-axis.

`condition.legend.text`
: List or named vector of character strings. This optional parameter can be used to assign labels to each condition variable values. It should be a list or named vector with length equal to `factor(condition)`. Each element should be named with one of the values from `factor(condition)`, and should contain the label. They will be listed in this order in the figure legend.

`include.TX.names`
: Logical value. If `TRUE`, then for the plots that include the annotated transcript, the transcript names will be listed. The labels will be drawn at half the size of `anno.cex.text`.

`draw.start.end.sites`
: Logical value. If `TRUE`, then transcript start/end sites will be marked on the main gene annotation.

`label.chromosome`
: Logical. If `TRUE`, label the chromosome in the left margin. If the text is too long it will be auto-fitted into the available margin.

`splice.junction.drawing.style`
: The visual style of the splice junctions drawn on the gene annotation. The default uses paired hyperbolas with the ends straightened out. A number of other styles are available.

draw.nested.SJ	Logical. If TRUE, overlapping splice junctions will be drawn layered under one another. This can vastly improve readability when there are a large number of overlapping splice junctions. Default is TRUE.
merge.exon.parts	Logical. If TRUE, in the gene annotation plot merge connected exon-fragments and delineate them with dotted lines.
verbose	if TRUE, send debugging and progress messages to the console / stdout.
debug.mode	Logical. If TRUE, print additional debugging information during execution.
INTERNAL.VARS	NOT FOR GENERAL USE. Intended only for use by JunctionSeq itself, internally. This is used for passing pre-generated data (when generating many similar plots, for example), and for internally-generated parameters. DO NOT USE.
...	Additional options to pass to plotting functions, particularly graphical parameters.

Value

This is a side-effecting function, and does not return a value.

Examples

```
data(exampleDataSet,package="JctSeqData");

plotJunctionSeqResultsForGene(geneID = "ENSRNOG00000009281", jscs);


## Not run:
########################################
#Set up example data:
decoder.file <- system.file(
                  "extdata/annoFiles/decoder.bySample.txt",
                  package="JctSeqData");
decoder <- read.table(decoder.file,
                  header=TRUE,
                  stringsAsFactors=FALSE);
gff.file <- system.file(
            "extdata/cts/withNovel.forJunctionSeq.gff.gz",
            package="JctSeqData");
countFiles <- system.file(paste0("extdata/cts/",
     decoder$sample.ID,
     "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
     package="JctSeqData");
######################
#Run example analysis:
jscs <- runJunctionSeqAnalyses(sample.files = countFiles,
           sample.names = decoder$sample.ID,
           condition=factor(decoder$group.ID),
           flat.gff.file = gff.file,
           analysis.type = "junctionsAndExons"
);
########################################

#Make an expression plot for a given gene:
plotJunctionSeqResultsForGene(geneID = "ENSRNOG00000009281", jscs);

#Plot normalized read counts for a given gene:
```

```
plotJunctionSeqResultsForGene(geneID = "ENSRNOG00000009281", jscs,
                    plot.type = "normCounts");

#Plot relative expression for a given gene:
plotJunctionSeqResultsForGene(geneID = "ENSRNOG00000009281", jscs,
                    plot.type = "rExpr");

#Plot raw read counts for a given gene:
plotJunctionSeqResultsForGene(geneID = "ENSRNOG00000009281", jscs,
                    plot.type = "rawCounts");

#Same thing, but with isoforms shown:
plotJunctionSeqResultsForGene(geneID = "ENSRNOG00000009281", jscs,
                    plot.type = "rawCounts",
                    displayTranscripts = TRUE);


## End(Not run)
```

---

plotMA *Generate a MA-Plot*

Description

Generates an MA-plot, which graphs the fold change versus the mean normalized expression. Statistically significant features are colored red.

Usage

```
plotMA(jscs,
       FDR.threshold = 0.01,
       fc.name = NULL,
       fc.thresh = 1,
       use.pch = 20,
       smooth.nbin = 256,
       ylim = c( 1 / 1000,1000),
       use.smoothScatter = TRUE,
       label.counts = TRUE,
       label.axes = c(TRUE,TRUE,FALSE,FALSE),
       show.labels = TRUE,
       par.cex = 1, points.cex = 1, text.cex = 1,
       lines.cex = 8,
       anno.lwd = 2, mar = c(4.1,4.1,3.1,1.1),
       miniTicks = TRUE,
       verbose = TRUE, debug.mode = FALSE,
       ...)
```

Arguments

jscs	A JunctionSeqCountSet. Usually created by runJunctionSeqAnalyses. Alternatively, this can be created manually by readJunctionSeqCounts. However in this case a number of additional steps will be necessary: Dispersions and

	size factors must then be set, usually using functions estimateSizeFactors and estimateJunctionSeqDispersions. Hypothesis tests must be performed by testForDiffUsage. Effect sizes and parameter estimates must be created via estimateEffectSizes.
FDR.threshold	The FDR threshold used to color dots. Tests with an adjusted-p-value more significant than this threshold will be marked in red.
fc.name	The name of the column to take from fData(jscs).
fc.thresh	The fold-change threshold required to count a significant locus in the count labels. It will also draw horizontal lines at this threshold.
use.pch	The value of pch to pass to the points call.
use.smoothScatter	Logical. If TRUE, non-significant genes will be ploted with density shading.
smooth.nbin	The number of bins to smooth, for the density plot, if use.smoothScatter is TRUE.
ylim	The y-axis limits.
label.counts	Logical. If TRUE, include labels showing the number of loci that pass both the statistical-significance and fold-change threshold in each direction.
label.axes	Logical vector. Whether to label each axis. Must have length 4; each corresponds to the bottom, left, top, and right axes respectively.
show.labels	Logical. If TRUE, include all titles and axes labels.
par.cex	The cex value to be passed to par.
points.cex	The cex value to be passed to points.
text.cex	The cex value to be passed to text.
lines.cex	The cex value to be passed to lines, box, and similar.
anno.lwd	The lwd value to be passed to lines, box, axis, and similar.
mar	The margin sizes, expressed in lines. see link{par}.
miniTicks	Logical. If TRUE, then include "mini tick marks" on the x and y axes.
verbose	if TRUE, send debugging and progress messages to the console / stdout.
debug.mode	if TRUE, send even more debugging and progress messages to the console / stdout.
...	Additional graphical parameters.

Value

This is a side-effecting function, and does not return a value.

Examples

```
data(exampleDataSet,package="JctSeqData");
plotMA(jscs);


## Not run:
########################################
#Set up example data:
decoder.file <- system.file(
                 "extdata/annoFiles/decoder.bySample.txt",
                 package="JctSeqData");
```

```
decoder <- read.table(decoder.file,
                  header=TRUE,
                  stringsAsFactors=FALSE);
gff.file <- system.file(
            "extdata/cts/withNovel.forJunctionSeq.gff.gz",
            package="JctSeqData");
countFiles <- system.file(paste0("extdata/cts/",
     decoder$sample.ID,
     "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
     package="JctSeqData");
######################
#Run example analysis:
jscs <- runJunctionSeqAnalyses(sample.files = countFiles,
           sample.names = decoder$sample.ID,
           condition=factor(decoder$group.ID),
           flat.gff.file = gff.file,
           analysis.type = "junctionsAndExons"
);
########################################

#Plot M-A:
plotMA(jscs);


## End(Not run)
```

`readAnnotationData` *Read junctionSeq annotation files produced by QoRTs.*

Description

This function reads the "flattened" gff annotation file created by QoRTs. This annotation file contains all the gene, transcript, exon, and junction ID's and their loci. In general this function is not used by the end-user, but is called internally by `runJunctionSeqAnalyses` or `readJunctionSeqCounts`.

Usage

```
readAnnotationData(flat.gff.file)
```

Arguments

`flat.gff.file`	Character string. The filename of the "flat" gff annotation file. The file may be gzip-compressed. This "flat" gff file must be produced by the QoRTs jar utility using the makeFlatGtf or mergeNovelSplices functions (depending on whether inclusion of novel splice junctions is desired).

Value

A `data.frame` object, containing the annotation information from the flat gff file.

Examples

```
gff.file <- system.file("extdata/cts/withNovel.forJunctionSeq.gff.gz",
                        package="JctSeqData");

#Parse the GFF file:
annoData <- readAnnotationData(gff.file);
head(annoData);
```

`readJunctionSeqCounts` *Read junctionSeq count files*

Description

This function loads read-count data (usually produced by QoRTs) and compiles them into a `JunctionSeqCountSet` object.

This function is called internally by the `runJunctionSeqAnalyses` function, and thus for most purposes users should not need to call this function directly. It may be useful to advanced users performing non-standard analyses.

Usage

```
readJunctionSeqCounts(countfiles, countdata,
    samplenames,  design,
    flat.gff.file,
    test.formula1 = formula(~ sample + countbin + condition : countbin),
    analysis.type = c("junctionsAndExons","junctionsOnly","exonsOnly"),
    nCores = 1,
    use.exons, use.junctions,
    use.known.junctions = TRUE, use.novel.junctions = TRUE,
    use.multigene.aggregates = FALSE,
    gene.names,
    verbose = TRUE,
    method.countVectors = c("geneLevelCounts","sumOfAllBinsForGene",
                            "sumOfAllBinsOfSameTypeForGene"),
    noDESeqMatrix = FALSE
)
```

Arguments

`countfiles`	Character vector. The filenames of the count files generated by QoRTs. The counts must all be generated using equivalent QoRTs parameters. The strandedness must be the same, as well as the inclusion of novel junctions.
`countdata`	List. An alternative parameterization. Instead of supplying count files using the countfiles parameter, you can pass a list of data frames, one for each sample. Each data frame should contain two columns: the first should be the feature id and the second should be the counts. This list must have the same length as the samplenames parameter.
`samplenames`	Character vector. A vector of full sample names, in the same order as the countfiles parameter.

`design` A data frame containing the condition variable and all desired covariates. All variables should be factors.

`flat.gff.file` Character string. The filename of the "flat" gff annotation file. Can be gzip-compressed. This "flat" gff file must be produced by the QoRTs jar utility using the makeFlatGtf or mergeNovelSplices functions (depending on whether inclusion of novel splice junctions is desired).

NOTE: This option is technically optional, but strongly recommended. If it is not included, then attempts to plot the results will crash unless (non-default) options are used to deactivate the plotting of genomic coordinates and transcript information

`test.formula1` For advanced users. The base formula for the alternate hypothesis model used in the hypothesis tests.

NOTE: the biological condition to be tested must be named "condition".

`analysis.type` Character string. One of "junctionsAndExons", "junctionsOnly", or "exonsOnly". This parameter determines what type of analysis is to be performed. By default JunctionSeq tests both splice junction loci and exonic regions for differential usage (a "hybrid" analysis). This parameter can be used to limit analyses specifically to either splice junction loci or exonic regions.

`nCores` The number of cores to use. Note that multicore functionality may not be available on all platforms.

`use.exons` Logical value. This is an alternate parameterization of the `analysis.type` parameter. If `TRUE`, then exonic region loci will be included in the analyses and will be tested for differential usage. If this parameter is set, then parameter `use.junctions` must also be set.

`use.junctions` Logical value. This is an alternate parameterization of the `analysis.type` parameter. If `TRUE`, then splice junction loci will be included in the analyses and will be tested for differential usage. If this parameter is set, then parameter `use.exons` must also be set.

`use.known.junctions`
Logical value. If `TRUE`, then known splice junctions will not be filtered out prior to analysis. Note: this is overidden if use.junctions is `FALSE` or if `analysis.type` is set to "exonsOnly".

`use.novel.junctions`
Logical value. If `TRUE`, then novel splice junctions will not be filtered out prior to analysis. Note: this is overidden if use.junctions is `FALSE` or if `analysis.type` is set to "exonsOnly".

`use.multigene.aggregates`
Logical value. Whether to attempt to test "aggregate genes" which consist of multiple genes that overlap with one another. Note that inclusion of aggregate genes may affect the false discovery rate, since by their very nature aggregate genes will often show differential splice junction usage, as the two genes will often be regulated independently.

`gene.names` data.frame. This optional parameter can be used to decoder the gene id's used in the actual analysis into gene symbols or gene names for general readability. This must be a text file name or data.frame with two columns of character strings. The first must be the gene ID's, and the second must be the gene names (as you wish them to appear in the plots). Genes are allowed to have multiple gene names, in which case they will be separated by commas. The gene names will be used in the plots and figures.

`verbose` if TRUE, send debugging and progress messages to the console / stdout.

`method.countVectors`
: Character string. Can be used to apply alternative methodologies or implementations. Intended for advanced users who have strong opinions about the underlying statistical methodologies.

  Determines the type of count vectors to be used in the model framework. By default JunctionSeq compares the counts for a specific feature against the counts across the rest of the gene minus the counts for the specific feature. Alternatively, the sum of all other features on the gene can be used, like in DEXSeq. The advantage to the default JunctionSeq behavior is that no read or read-pair is ever counted more than once in any model. Under DEXSeq, some reads may cover many exonic segments and thus be counted repeatedly.

`noDESeqMatrix`
: Suppresses the internal generation of a DESeq2 object. Depending on the options used this will break many downstream steps, and is for advanced usage only.

Value

A JunctionSeqCountSet.

Examples

```
########################################
#Set up example data:
decoder.file <- system.file(
                  "extdata/annoFiles/decoder.bySample.txt",
                  package="JctSeqData");
decoder <- read.table(decoder.file,
                  header=TRUE,
                  stringsAsFactors=FALSE);
gff.file <- system.file(
             "extdata/tiny/withNovel.forJunctionSeq.gff.gz",
             package="JctSeqData");
countFiles <- system.file(paste0("extdata/tiny/",
     decoder$sample.ID,
     "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
     package="JctSeqData");
########################################
#Advanced Analysis:

#Make a "design" dataframe:
design <- data.frame(condition = factor(decoder$group.ID));
#Read the QoRTs counts.
jscs = readJunctionSeqCounts(countfiles = countFiles,
           samplenames = decoder$sample.ID,
           design = design,
           flat.gff.file = gff.file
);
```

`runJunctionSeqAnalyses` *Run a JunctionSeq analysis.*

Description

This function runs a complete analysis from start to finish. It internally calls functions readAnnotationData, readJunctionSeqCounts, estimateJunctionSeqSizeFactors, estimateJunctionSeqDispersions, fitJunctionSeqDispersionFunction, testForDiffUsage, and estimateEffectSizes.

Usage

```
runJunctionSeqAnalyses(sample.files, sample.names, condition,
   flat.gff.file,
   analysis.type = c("junctionsAndExons","junctionsOnly","exonsOnly"),
   meanCountTestableThreshold = "auto",
   nCores = 1,
   use.covars,
   test.formula0 = formula(~ sample + countbin),
   test.formula1 = formula(~ sample + countbin + condition : countbin),
   effect.formula = formula(~ condition + countbin + condition : countbin),
   geneLevel.formula = formula(~ condition),
   use.exons, use.junctions,
   use.known.junctions = TRUE,
   use.novel.junctions = TRUE,
   use.multigene.aggregates = FALSE,
   gene.names,
   method.GLM = c(c("advanced","DESeq2-style"),
                  c("simpleML","DEXSeq-v1.8.0-style")),
   method.dispFit = c("parametric", "local", "mean"),
   method.dispFinal = c("shrink","max","fitted","noShare"),
   method.sizeFactors = c("byGenes","byCountbins"),
   method.countVectors = c("geneLevelCounts","sumOfAllBinsForGene",
                           "sumOfAllBinsOfSameTypeForGene"),
   method.expressionEstimation = c("feature-vs-gene",
                                   "feature-vs-otherFeatures"),
   method.cooksFilter = TRUE,
   optimizeFilteringForAlpha = 0.01,
   fitDispersionsForExonsAndJunctionsSeparately = TRUE,
   keep.hypothesisTest.fit = FALSE,
   keep.estimation.fit = FALSE,
   replicateDEXSeqBehavior.useRawBaseMean = FALSE,
   verbose = TRUE, debug.mode = FALSE)
```

Arguments

sample.files	Character vector. The filenames of the count files generated by QoRTs. The counts must all be generated using equivalent QoRTs parameters. The strandedness must be the same, as well as the inclusion of novel junctions.
sample.names	A character vector of sample names. This must have the same length as sample.files, and should be in the same order.
condition	A factor vector of condition values. This must have the same length as sample.files and sample.names, and should be listed in the same order.
flat.gff.file	A flattened gff-formatted annotation file from which the gene counts were generated. Technically optional, but STRONGLY RECOMMENDED, as the annotation data WILL be required by plotting functions.

`analysis.type` Character string. One of "junctionsAndExons", "junctionsOnly", or "exonsOnly". This parameter determines what type of analysis is to be performed. By default JunctionSeq tests both splice junction loci and exonic regions for differential usage (a "hybrid" analysis). This parameter can be used to limit analyses specifically to either splice junction loci or exonic regions.

`meanCountTestableThreshold`
"auto" or Numeric value. Features with a total mean normalized count of less than this value will be excluded from the analyses. If left as the default ("auto"), then the cutoff threshold will be determined automatically using the DESeq2 independent filtering method.

`nCores` Either an integer or a BiocParallelParam object. Either way, this determines The number of cores to use. Note that multicore functionality may not be available on all platforms. If parallel execution is not available then JunctionSeq will automatically fallback to single-core execution. See the BiocParallel package for more information.

`use.covars` Optional: for advanced users. A data frame containing covariate factors. The names must be included in the model formulas.

`test.formula0` For advanced users. The base formula for the null hypothesis model used in the hypothesis tests.
NOTE: the biological condition to be tested must be named "condition".

`test.formula1` For advanced users. The base formula for the alternate hypothesis model used in the hypothesis tests.
NOTE: the biological condition to be tested must be named "condition".

`effect.formula` For advanced users. The base formula for the model used for effect size estimation.
NOTE: the biological condition to be tested must be named "condition".

`geneLevel.formula`
For advanced users. The base formula for the model used to estimate total gene-level expression.
NOTE: the biological condition to be tested must be named "condition".

`use.exons` Logical value. This is an alternate parameterization of the `analysis.type` parameter. If `TRUE`, then exonic region loci will be included in the analyses and will be tested for differential usage. If this parameter is set, then parameter `use.junctions` must also be set.

`use.junctions` Logical value. This is an alternate parameterization of the `analysis.type` parameter. If `TRUE`, then splice junction loci will be included in the analyses and will be tested for differential usage. If this parameter is set, then parameter `use.exons` must also be set.

`use.known.junctions`
Logical value. If `TRUE`, then known splice junctions will not be filtered out prior to analysis. Note: this is overidden if use.junctions is `FALSE` or if `analysis.type` is set to "exonsOnly".

`use.novel.junctions`
Logical value. If `TRUE`, then novel splice junctions will not be filtered out prior to analysis. Note: this is overidden if use.junctions is `FALSE` or if `analysis.type` is set to "exonsOnly".

`use.multigene.aggregates`
Logical value. Whether to attempt to test "aggregate genes" which consist of multiple genes that overlap with one another. Note that inclusion of aggregate

genes may affect the false discovery rate, since by their very nature aggregate genes will often show differential splice junction usage, as the two genes will often be regulated independently.

`gene.names` data.frame. This optional parameter can be used to decoder the gene id's used in the actual analysis into gene symbols or gene names for general readability. This must be a text file name or data.frame with two columns of character strings. The first must be the gene ID's, and the second must be the gene names (as you wish them to appear in the plots). Genes are allowed to have multiple gene names, in which case they will be separated by commas. The gene names will be used in the plots and figures.

`method.GLM` Character string. Can be used to apply alternative methodologies or implementations. Intended for advanced users who have strong opinions about the underlying statistical methodologies.

The default is "advanced" or, equivalently, "DESeq2-style". This uses the dispersion estimation methodology used by DESeq2 and DEXSeq v1.12.0 or higher to generate the initial (feature-specific) dispersion estimates. The alternative method is "simpleML" or, equivalently, "DEXSeq-v1.8.0-style". This uses a simpler maximum-likelihood-based method used by the original DESeq and by DEXSeq v1.8.0 or less.

`method.dispFit` Character string. Can be used to apply alternative methodologies or implementations. Intended for advanced users who have strong opinions about the underlying statistical methodologies.

Determines the method used to generated "fitted" dispersion estimates. One of "parametric" (the default), "local", or "mean". See the DESeq2 documentation for more information.

`method.dispFinal`

Character string. Can be used to apply alternative methodologies or implementations. Intended for advanced users who have strong opinions about the underlying statistical methodologies.

Determines the method used to arrive at a "final" dispersion estimate. The default, "shrink" uses the maximum a posteriori estimate, combining information from both the fitted and feature-specific dispersion estimates. This is the method used by DESeq2 and DEXSeq v1.12.0 and above.

`method.sizeFactors`

Character string. Can be used to apply alternative methodologies or implementations. Intended for advanced users who have strong opinions about the underlying statistical methodologies.

Determines the method used to calculate normalization size factors. By default JunctionSeq uses gene-level expression. As an alternative, feature-level counts can be used as they are in DEXSeq. In practice the difference is almost always negligible.

`method.countVectors`

Character string. Can be used to apply alternative methodologies or implementations. Intended for advanced users who have strong opinions about the underlying statistical methodologies.

Determines the type of count vectors to be used in the model framework. By default JunctionSeq compares the counts for a specific feature against the counts across the rest of the gene minus the counts for the specific feature. Alternatively, the sum of all other features on the gene can be used, like in DEXSeq. The advantage to the default JunctionSeq behavior is that no read or read-pair

is ever counted more than once in any model. Under DEXSeq, some reads may cover many exonic segments and thus be counted repeatedly.

`method.expressionEstimation`

Character string. Can be used to apply alternative methodologies or implementations. Intended for advanced users who have strong opinions about the underlying statistical methodologies.

Determines the methodology used to generate feature expression estimates and relative fold changes. By default each feature is modeled separately. Under the default count-vector method, this means that the resultant relative fold changes will be a measure of the relative fold change between the feature and the gene as a whole.

Alternatively, the "feature-vs-otherFeatures" method builds a large, complex model containing all features belonging to the gene. The coefficients for each feature are then "balanced" using linear contrasts weighted by the inverse of their variance. In general we have found this method to produce very similar results but less efficiently and less consistently. Additionally, this alternative method "multi-counts" reads that cover more than one feature. This can result in over-weighting of exonic regions with a large number of annotated variations in a small genomic area, as each individual read or read-pair may be counted many times in the model.

Under the default option, no read or read-pair is ever counted more than once in a given model.

`method.cooksFilter`

Logical value. if `TRUE`, use the cook's filter to detect and remove outliers.

`optimizeFilteringForAlpha`

Numeric value between 0 and 1. If `meanCountTestableThreshold` is set to "auto" then this sets the adjusted-p-value threshold to optimize against.

`fitDispersionsForExonsAndJunctionsSeparately`

When running a "junctionsAndExons" type analysis in which both exons and splice junctions are being tested simultaniously, this parameter determines whether a single fitted dispersion model should be fitted for both exons and splice junctions, or if separate fitted dispersions should be calculated for each. By default the dispersions are run separately.

`keep.hypothesisTest.fit`

Logical value. If `TRUE`, save both complete hypothesis test model fits for every gene. This will require a lot of memory, but may be useful for statistical diagnostics. Default: `FALSE`.

`keep.estimation.fit`

Logical value. If `TRUE`, save the complete model fits for every gene. This will require a lot of memory, but may be useful for statistical diagnostics. Default: `FALSE`.

`verbose` if TRUE, send debugging and progress messages to the console / stdout.

`debug.mode` if TRUE, send even more debugging and progress messages to the console / stdout.

`replicateDEXSeqBehavior.useRawBaseMean`

USED ONLY FOR INTERNAL TESTING! NOT INTENDED FOR ACTUAL USE!

This variable activates an alternative mode in which a (very minor) bug in DEXSeq v1.14.0 and earlier is replicated. If `TRUE`, the baseMean and baseVar variables will be computed using raw counts rather than normalized counts. This is used

for internal tests in which DEXSeq functionality is replicated precisely and the results are compared against equivalent DEXSeq results. Without this option the results would differ slightly (generally by less than 1 hundreth of a percent).

USED ONLY FOR INTERNAL TESTING! NOT INTENDED FOR ACTUAL USE!

Value

A JunctionSeqCountSet object, containing the complete analysis dataset and results.

Examples

```
## Not run:
########################################
#Set up example data:
decoder.file <- system.file(
                  "extdata/annoFiles/decoder.bySample.txt",
                  package="JctSeqData");
decoder <- read.table(decoder.file,
                  header=TRUE,
                  stringsAsFactors=FALSE);
gff.file <- system.file(
            "extdata/cts/withNovel.forJunctionSeq.gff.gz",
            package="JctSeqData");
countFiles <- system.file(paste0("extdata/cts/",
     decoder$sample.ID,
     "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
     package="JctSeqData");
########################################

jscs <- runJunctionSeqAnalyses(sample.files = countFiles,
           sample.names = decoder$sample.ID,
           condition=factor(decoder$group.ID),
           flat.gff.file = gff.file,
           analysis.type = "junctionsAndExons"
);


## End(Not run)
```

---

setJunctionSeqCompiledSourcePackage

Set the source of the internal compiled C++ binaries.

Description

This function sets the package that contains the compiled C++ binaries needed by JunctionSeq.

This is intended for advanced users, or for users attempting to install testing versions of JunctionSeq without a C++ compiler.

Usage

```
setJunctionSeqCompiledSourcePackage( PACKAGE = c("JunctionSeq","DESeq2") )
```

Arguments

PACKAGE	The name of the package where the compiled binaries are stored.

---

testForDiffUsage *Test Junctions for Differential Junction Usage*

Description

This function runs the hypothesis tests for differential junction usage.

This function is called internally by the runJunctionSeqAnalyses function, and thus for most purposes users should not need to call this function directly. It may be useful to advanced users performing non-standard analyses.

Usage

```
testForDiffUsage( jscs,
    test.formula0 = formula(~ sample + countbin),
    test.formula1 = formula(~ sample + countbin + condition : countbin),
    method.GLM = c(c("advanced","DESeq2-style"),
                   c("simpleML","DEXSeq-v1.8.0-style")),
    dispColumn="dispersion", nCores=1,
    keep.hypothesisTest.fit = FALSE,
    meanCountTestableThreshold = "auto",
    optimizeFilteringForAlpha = 0.01,
    method.cooksFilter = TRUE,
    cooksCutoff,
    pAdjustMethod = "BH",
    verbose = TRUE)
```

Arguments

jscs	A JunctionSeqCountSet. Usually initially created by readJunctionSeqCounts. Dispersions and size factors must be set, usually using functions estimateJunctionSeqSizeFactors and estimateJunctionSeqDispersions.
test.formula0	The formula for the null hypothesis. Note that the condition to be tested must be named "condition".
test.formula1	The formula for the alternative hypothesis. Note that the condition to be tested must be named "condition".
method.GLM	Character string. Can be used to apply alternative methodologies or implementations. Intended for advanced users who have strong opinions about the underlying statistical methodologies. The default is "advanced" or, equivalently, "DESeq2-style". This uses the model test methodology used by DESeq2 and DEXSeq v1.12.0 or higher. The alternative method is "simpleML" or, equivalently, "DEXSeq-v1.8.0-style". This uses a simpler maximum-likelihood-based method used by the original DESeq and by some earlier versions of DEXSeq (v1.8.0 or less).
dispColumn	Character value. The name of the fData(jscs) column in which the model dispersion is stored.

nCores — Either an integer or a BiocParallelParam object. Either way, this determines The number of cores to use. Note that multicore functionality may not be available on all platforms. If parallel execution is not available then JunctionSeq will automatically fallback to single-core execution. See the BiocParallel package for more information.

keep.hypothesisTest.fit — Logical value. If TRUE, save both complete hypothesis test model fits for every gene. This will require a lot of memory, but may be useful for statistical diagnostics. Default: FALSE.

meanCountTestableThreshold — "auto" or Numeric value. Features with a total mean normalized count of less than this value will be excluded from the analyses. If left as the default ("auto"), then the cutoff threshold will be determined automatically using the DESeq2 independent filtering method.

optimizeFilteringForAlpha — Numeric value between 0 and 1. If meanCountTestableThreshold is set to "auto" then this sets the adjusted-p-value threshold to optimize against.

method.cooksFilter — Logical value. if TRUE, use the cook's filter to detect and remove outliers.

cooksCutoff — The cook's cutoff threshold to use.

pAdjustMethod — The p-adjustment method to use with the p.adjust function.

verbose — if TRUE, send debugging and progress messages to the console / stdout.

Value

A JunctionSeqCountSet, with hypothesis test results included.

Examples

```
data(exampleDataSet,package="JctSeqData");
jscs <- testForDiffUsage(jscs);

## Not run:
########################################
#Set up example data:
decoder.file <- system.file(
                  "extdata/annoFiles/decoder.bySample.txt",
                  package="JctSeqData");
decoder <- read.table(decoder.file,
                  header=TRUE,
                  stringsAsFactors=FALSE);
gff.file <- system.file(
            "extdata/cts/withNovel.forJunctionSeq.gff.gz",
            package="JctSeqData");
countFiles <- system.file(paste0("extdata/cts/",
     decoder$sample.ID,
     "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
     package="JctSeqData");
########################################
#Advanced Analysis:

#Make a "design" dataframe:
design <- data.frame(condition = factor(decoder$group.ID));
```

```
#Read the QoRTs counts.
jscs = readJunctionSeqCounts(countfiles = countFiles,
           samplenames = decoder$sample.ID,
           design = design,
           flat.gff.file = gff.file
);
#Generate the size factors and load them into the JunctionSeqCountSet:
jscs <- estimateJunctionSeqSizeFactors(jscs);
#Estimate feature-specific dispersions:
jscs <- estimateJunctionSeqDispersions(jscs);
#Fit dispersion function and estimate MAP dispersion:
jscs <- fitJunctionSeqDispersionFunction(jscs);
#Test for differential usage:
jscs <- testForDiffUsage(jscs);
#Estimate effect sizes and expression estimates:
jscs <- estimateEffectSizes( jscs);


## End(Not run)
```

`writeBedTrack` *Write splice junction browser tracks*

Description

This function saves the JunctionSeq results in the form of a set of "bed" files designed for use with the UCSC genome browser.

Usage

```
writeExprBedTrack(file, jscs,
     trackLine,
     only.with.sig.gene = FALSE,
     only.sig = FALSE,
     only.testable = TRUE,
     plot.exons = TRUE, plot.junctions = TRUE, plot.novel.junctions = TRUE,
     group.RGB,
     use.score = FALSE,
     FDR.threshold = 0.05,
     count.digits = 1,
     includeGeneID = FALSE,
     includeLocusID = TRUE,
     includeGroupID = TRUE,
     output.format = c("BED", "GTF", "GFF3"),
     use.gzip = TRUE,
     verbose = TRUE)

writeSigBedTrack(file,
     jscs,
     trackLine,
     only.sig = TRUE,
     only.testable = TRUE,
```

```
    plot.exons = TRUE, plot.junctions = TRUE, plot.novel.junctions = TRUE,
    sig.RGB = "255,0,0",
    nonsig.RGB = "0,0,0",
    use.score = TRUE,
    FDR.threshold = 0.05,
    pval.digits = 4,
    includeGeneID = FALSE,
    includeLocusID = TRUE,
    output.format = c("BED", "GTF", "GFF3"),
    use.gzip = TRUE,
    verbose = TRUE)
```

Arguments

file	Character string. File path for the output bed file.
jscs	A JunctionSeqCountSet. Usually created by runJunctionSeqAnalyses. Alternatively, this can be created manually by readJunctionSeqCounts. However in this case a number of additional steps will be necessary: Dispersions and size factors must then be set, usually using functions estimateSizeFactors and estimateJunctionSeqDispersions. Hypothesis tests must be performed by testForDiffUsage. Effect sizes and parameter estimates must be created via estimateEffectSizes.
trackLine	The "track line" of the bed file. In other words, the first line of the file. By default JunctionSeq will attempt to automatically generate a reasonable track line.
only.with.sig.gene	Logical. If TRUE, only genes containing statistically significant results will be included.
only.sig	Logical. If TRUE, only statistically significant loci will be included.
only.testable	Logical. If TRUE, only loci with sufficiently high expression to be tested will be included.
plot.exons	Logical. If TRUE, exons will be plotted.
plot.junctions	Logical. If TRUE, splice junctions will be plotted.
plot.novel.junctions	Logical. If TRUE, novel splice junctions will be plotted (if plot.junctions is also TRUE).
sig.RGB	Character string. The RGB color for significant genes. Must be in the format "r,g,b", with each value ranging from 0 to 255.
nonsig.RGB	Character string. The RGB color for non-significant loci. Must be in the format "r,g,b", with each value ranging from 0 to 255.
group.RGB	Character string. The RGB color used for each experimental group. Must be in the format "r,g,b", with each value ranging from 0 to 255. Must have a length equal to the number of experimental condition values.
use.score	Logical. If TRUE, score each locus based on the p-value.
FDR.threshold	Numeric. The FDR-adjusted p-value threshold to use to assign statistical significance.
count.digits	Numeric. The number of digits after the decimal point to include for the mean normalized counts.

pval.digits	Numeric. The number of digits after the decimal point to include for the p-values.
includeGeneID	Logical. If TRUE, include the ID of the gene in the "name" field of each line.
includeLocusID	Logical. If TRUE, include the ID of the locus in the "name" field of each line.
includeGroupID	Logical. If TRUE, include the ID of the group in the "name" field of each line.
output.format	Character string. The format to use.
use.gzip	Logical. Whether or not to gzip the bed file.
verbose	Logical. if TRUE, output debugging/progress information.

Value

This is a side-effecting function, and does not return a value.

Examples

```
data(exampleDataSet,package="JctSeqData");
writeExprBedTrack("test.exonCoverage.bed.gz", jscs,
                  plot.exons = TRUE, plot.junctions = FALSE)

## Not run:
########################################
#Set up example data:
decoder.file <- system.file(
                  "extdata/annoFiles/decoder.bySample.txt",
                  package="JctSeqData");
decoder <- read.table(decoder.file,
                  header=TRUE,
                  stringsAsFactors=FALSE);
gff.file <- system.file(
            "extdata/cts/withNovel.forJunctionSeq.gff.gz",
            package="JctSeqData");
countFiles <- system.file(paste0("extdata/cts/",
     decoder$sample.ID,
     "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
     package="JctSeqData");
######################
#Run example analysis:
jscs <- runJunctionSeqAnalyses(sample.files = countFiles,
           sample.names = decoder$sample.ID,
           condition=factor(decoder$group.ID),
           flat.gff.file = gff.file,
           analysis.type = "junctionsAndExons"
);
########################################

#Exon coverage:
writeExprBedTrack("test.exonCoverage.bed.gz", jscs,
                  plot.exons = TRUE, plot.junctions = FALSE)
#Junction coverage:
writeExprBedTrack("test.jctCoverage.bed.gz", jscs,
                  plot.exons = FALSE, plot.junctions = TRUE)
#Both Exon and Junction coverage:
writeExprBedTrack("test.featureCoverage.bed.gz", jscs)
```

```
#p-values of significant features:
writeSigBedTrack("test.pvals.bed.gz", jscs)


## End(Not run)
```

`writeCompleteResults` *Produce output data files, given annotation files and DEXSeq exonCountSet object and DEXSeq results data.*

Description

This function takes the raw DEXSeq results and merges in feature annotations, as well as calculating and merging in a number of different normalized and fitted values for each level of the condition variable.

Usage

```
writeCompleteResults(jscs, outfile.prefix,
                         gzip.output = TRUE,
                         FDR.threshold = 0.01,
                         save.allGenes = TRUE, save.sigGenes = TRUE,
                         save.fit = FALSE, save.VST = FALSE,
                         save.bedTracks = TRUE,
                         save.jscs = FALSE,
                         bedtrack.format = c("BED", "GTF", "GFF3"),
                         verbose = TRUE)
```

Arguments

`jscs`	A `JunctionSeqCountSet`. Usually created by `runJunctionSeqAnalyses`. Alternatively, this can be created manually by `readJunctionSeqCounts`. However in this case a number of additional steps will be necessary: Dispersions and size factors must then be set, usually using functions `estimateSizeFactors` and `estimateJunctionSeqDispersions`. Hypothesis tests must be performed by `testForDiffUsage`. Effect sizes and parameter estimates must be created via `estimateEffectSizes`.
`outfile.prefix`	A string indicating the filename prefix where output files should be saved.
`gzip.output`	Logical. If `TRUE`, then all ".txt" text files should be gzip-compressed to save space.
`FDR.threshold`	The adjusted-p-value threshold used to determine statistical significance.
`save.allGenes`	Logical. Whether to save files containing data for all genes.
`save.sigGenes`	Logical. Whether to save a separate set of files containing data for only the significant genes. If this and save.allGenes are both true then two sets of files will be generated.
`save.fit`	Logical. Whether to save model fit data.
`save.VST`	Logical. Whether to save VST-transformed data.
`save.bedTracks`	Logical. Whether to save "bed" junction coverage tracks.
`save.jscs`	Logical. Whether to the entire `JunctionSeqCountSet` object. Default is FALSE.

bedtrack.format
: Character string. The format to use for the browser tracks.

verbose
: A boolean flag indicating whether or not to print progress information during execution. (Default=FALSE)

Details

Saves a wide variety of data from the analyses.

Value

This is a side-effecting function, and does not return a value.

Examples

```
data(exampleDataSet,package="JctSeqData");
#Write results tables and browser track files:
writeCompleteResults(jscs, outfile.prefix = "./results.");

## Not run:
########################################
#Set up example data:
decoder.file <- system.file(
                  "extdata/annoFiles/decoder.bySample.txt",
                  package="JctSeqData");
decoder <- read.table(decoder.file,
                  header=TRUE,
                  stringsAsFactors=FALSE);
gff.file <- system.file(
            "extdata/cts/withNovel.forJunctionSeq.gff.gz",
            package="JctSeqData");
countFiles <- system.file(paste0("extdata/cts/",
     decoder$sample.ID,
     "/QC.spliceJunctionAndExonCounts.withNovel.forJunctionSeq.txt.gz"),
     package="JctSeqData");
######################
#Run example analysis:
jscs <- runJunctionSeqAnalyses(sample.files = countFiles,
           sample.names = decoder$sample.ID,
           condition=factor(decoder$group.ID),
           flat.gff.file = gff.file,
           analysis.type = "junctionsAndExons"
);
########################################

#Write results tables and browser track files:
writeCompleteResults(jscs, outfile.prefix = "./results.");


## End(Not run)
```

Index