Package 'EventPointer'

May 15, 2026

Type Package

Title An effective identification of alternative splicing events using junction arrays and RNA-Seq data

Version 3.21.0

Description EventPointer is an R package to identify alternative splicing events that involve either simple (case-control experiment) or complex experimental designs such as time course experiments and studies including paired-samples. The algorithm can be used to analyze data from either junction arrays (Affymetrix Arrays) or sequencing data (RNA-Seq).
In the latter, EventPointer can work with annotated splicing events or can build a splicing graph from the RNA-Seq reads and then identify new and specific alternative splicing events.
The software returns a data.frame with the detected alternative splicing events: gene name, type of event (cassette, alternative 3',...,etc), genomic position, statistical significance and increment of the percent spliced in (Delta PSI) for all the events.
The algorithm can generate a series of files to visualize the detected alternative splicing events in IGV. This eases the interpretation of results and the design of primers for standard PCR validation.

Depends R (>= 3.4), SGSeq, Matrix, SummarizedExperiment

Imports txdbmaker, stringr, GenomeInfoDb, igraph, MASS, nnls, limma, matrixStats, RBGL, prodlim, graph, methods, utils, stats, doParallel, foreach, affxparser, GenomicRanges, GenomicAlignments, Rsamtools, S4Vectors, IRanges, qvalue, cobs, rhdf5, BSgenome, Biostrings, glmnet, abind, aroma.light, iterators, lpSolve, poibin, speedglm, tximport, fgsea

Suggests knitr, rmarkdown, BiocStyle, RUnit, BiocGenerics, dplyr, kableExtra

License Artistic-2.0

LazyData true

RoxygenNote 7.3.3

Encoding UTF-8

biocViews AlternativeSplicing, DifferentialSplicing, mRNAMicroarray, RNASeq, Transcription, Sequencing, TimeCourse, ImmunoOncology

VignetteBuilder knitr

Url https://github.com/jpromeror/EventPointer

BugReports https://github.com/jpromeror/EventPointer/issues

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

git_branch devel

git_last_commit afef15f

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-05-15

Author Juan Pablo Romero [aut],
Juan A. Ferrer-Bonsoms [aut, cre],
Pablo Sacristan [aut],
Ander Muniategui [aut],
Fernando Carazo [aut],
Ander Aramburu [aut],
Angel Rubio [aut]

Maintainer Juan A. Ferrer-Bonsoms <jafhernandez@tecnun.es>

Contents

`AllEvents_RNASeq` *Alternative splicing events detected by EventPointer*

Description

Alternative splicing events detected by EventPointer

Usage

```
data(AllEvents_RNASeq)
```

Format

A `list` object `AllEvents_RNASeq[[i]][[j]]` displays the jth splicing event for the ith gene.

Value

AllEvents_RNASeq object contains all the detected alternativesplicing events using EventPointer-methodology. The splicing events where detected using the BAM files from the dataset published in Seshagiri et al. 2012 andused in the `SGSeq` R package vignette.

`AllEvents_RNASeq_MP` *Alternative splicing multi-path events detected by EventPointer*

Description

Alternative splicing multi-path events detected by EventPointer

Usage

```
data(AllEvents_RNASeq_MP)
```

Format

A `list` object `AllEvents_RNASeq[[i]][[j]]` displays the jth splicing event for the ith gene.

Value

AllEvents_RNASeq_MP object contains all the detected alternative splicing events using Event-Pointer methodology for multi-path events. The splicing events where detected using the BAM files from the dataset published in Seshagiri et al. 2012 and used in the `SGSeq` R package vignette.

ArrayDatamultipath *Preprocessed arrays data with multi-path events*

Description

Preprocessed arrays data with multi-path events

Usage

```
data(ArrayDatamultipath)
```

Format

A `data.frame` with preprocessed arrays data. The preprocessing was done using `aroma.affymetrix`. See the package vignette for the preprocessing pipeline

Value

ArrayDatamultipath object contains preprocessed junction arrays data. The preprocessing was done using `aroma.affymetrix` R package, refer to EventPointer vignette for the pipeline used for the preprocessing. The data corresponds to 4 samples from the SUM149 Cell line hybridized to the HTA 2.0 Affymetrix array. The first two samples are control and the second ones are treated.

ArraysData *Preprocessed arrays data*

Description

Preprocessed arrays data

Usage

```
data(ArraysData)
```

Format

A `data.frame` with preprocessed arrays data. The preprocessing was done using `aroma.affymetrix`. See the package vignette for the preprocessing pipeline.

Value

ArraysData object contains preprocessed junction arrays data. The preprocessing was done using `aroma.affymetrix` R package, refer to EventPointer vignette for the pipeline used for the preprocessing. The data corresponds to 4 samples from the SUM149 Cell line hybridized to the HTA 2.0 Affymetrix array. The first two samples are control and the second ones are treated.

CDFfromGTF *CDF file creation for EventPointer*

Description

Generates the CDF file to be used under the aroma.affymetrix framework

Usage

```
CDFfromGTF(
  input = "Ensembl",
  inputFile = NULL,
  PSR,
  Junc,
  PathCDF,
  microarray = NULL
)
```

Arguments

input	Reference transcriptome used to build the CDF file. Must be one of: 'Ensembl', 'UCSC' , 'AffyGTF' or 'CustomGTF'.
inputFile	If input is 'AffyGTF' or 'CustomGTF', inputFile should point to the GTF file to be used.
PSR	Path to the Exon probes txt file.
Junc	Path to the Junction probes txt file.
PathCDF	Directory where the output will be saved.
microarray	Microarray used to create the CDF file. Must be one of: HTA-2_0, ClariomD, RTA or MTA.

Value

The function displays a progress bar to show the user the progress of the function. However, there is no value returned in R as the function creates three files that are used later by other EventPointer functions. 1) EventsFound.txt : Tab separated file with all the information of all the alternative splcing events found. 2) .flat file : Used to build the corresponding CDF file. 3) .CDF file: Output required for the aroma.affymetrix preprocessing pipeline. Both the .flat and .CDF file take large ammounts of memory in the hard drive, it is recommended to have at least 1.5 GB of free space.

Examples

```
## Not run:
   PathFiles<-system.file('extdata',package='EventPointer')
   DONSON_GTF<-paste(PathFiles,'/DONSON.gtf',sep='')
   PSRProbes<-paste(PathFiles,'/PSR_Probes.txt',sep='')
   JunctionProbes<-paste(PathFiles,'/Junction_Probes.txt',sep='')
   Directory<-tempdir()
   microarray<-'HTA-2_0'

  # Run the function
```

```
CDFfromGTF(input='AffyGTF',inputFile=DONSON_GTF,PSR=PSRProbes,Junc=JunctionProbes,
           PathCDF=Directory,microarray=microarray)

## End(Not run)
```

CDFfromGTF_Multipath *CDF file creation for EventPointer (MultiPath)*

Description

Generates the CDF file to be used under the aroma.affymetrix framework.

Usage

```
CDFfromGTF_Multipath(
  input = "Ensembl",
  inputFile = NULL,
  PSR,
  Junc,
  PathCDF,
  microarray = NULL,
  paths = 2
)
```

Arguments

input	Reference transcriptome used to build the CDF file. Must be one of Ensembl, UCSC or GTF.
inputFile	If input is GTF, inputFile should point to the GTF file to be used.
PSR	Path to the Exon probes txt file.
Junc	Path to the Junction probes txt file.
PathCDF	Directory where the output will be saved.
microarray	Microarray used to create the CDF file. Must be one of: HTA-2_0, ClariomD, RTA or MTA.
paths	Maximum number of paths of the events to find.

Value

The function displays a progress bar to show the user the progress of the function. However, there is no value returned in R as the function creates three files that are used later by other EventPointer functions. 1) EventsFound.txt : Tab separated file with all the information of all the alternative splcing events found. 2) .flat file : Used to build the corresponding CDF file. 3) .CDF file: Output required for the aroma.affymetrix preprocessing pipeline. Both the .flat and .CDF file take large ammounts of memory in the hard drive, it is recommended to have at least 1.5 GB of free space.

Examples

```
## Not run:
  PathFiles<-system.file('extdata',package='EventPointer')
  DONSON_GTF<-paste(PathFiles,'/DONSON.gtf',sep='')
  PSRProbes<-paste(PathFiles,'/PSR_Probes.txt',sep='')
  JunctionProbes<-paste(PathFiles,'/Junction_Probes.txt',sep='')
  Directory<-tempdir()
  microarray<-'HTA-2_0'

 # Run the function

 CDFfromGTF_Multipath(input='AffyGTF',inputFile=DONSON_GTF,PSR=PSRProbes,Junc=JunctionProbes,
                      PathCDF=Directory,microarray=microarray,paths=3)

## End(Not run)
```

`CreateExSmatrix` *Events X RBPS matrix creation*

Description

Generates the Events x RBP matrix for the splicing factor enrichment analysis.

Usage

```
CreateExSmatrix(
  pathtoeventstable,
  SG_List,
  nt = 400,
  Peaks,
  POSTAR,
  EventsRegions = NULL,
  cores = 1
)
```

Arguments

`pathtoeventstable`	Path to eventsFound.txt with the information of all the events.
`SG_List`	List with the information of the splicing graph of the genes. Returned by the function EventDetection_transcriptome.
`nt`	Number of nt up and down for the splicing regions of each event.
`Peaks`	Table with the peaks.
`POSTAR`	Table with peaks of POSTAR
`EventsRegions`	Events regions if calculated previously. Not need to calculated again.
`cores`	Number of cores if user want to run in parallel.

Value

The function returns a list with the ExS matrix and with the splicing regions of the events. If the Splicing regions is an input of the function then only the ExS matrix will be returned. The ExS matrix is the input for the Splicing Factor enrichment analysis.

EventDetection *Detect splicing events using EventPointer methodology*

Description

Identification of all the alternative splicing events in the splicing graphs

Usage

```
EventDetection(Input, cores)
```

Arguments

Input	Output of PrepareBam_EP function.
cores	Number of cores used for parallel processing.

Value

list with all the events found for all the genes present in the experiment. It also generates a file called EventsFound_RNASeq.txt with the information of each event.

Examples

```
## Not run:
  # Run EventDetection function
   data(SG_RNASeq)
   TxtPath<-tempdir()
   AllEvents_RNASeq<-EventDetection(SG_RNASeq,cores=1)

## End(Not run)
```

EventDetectionAnn *Detect splicing events using EventPointer methodology*

Description

Identification of all the alternative splicing events in the splicing graphs

Usage

```
EventDetectionAnn(Input, cores)
```

Arguments

Input	Output of PrepareBam_EP function.
cores	Number of cores used for parallel processing.

Value

list with all the events found for all the genes present in the experiment. It also generates a file called EventsFound_RNASeq.txt with the information of each event.

EventDetectionMultipath

Detect splicing multipath events using EventPointer methodology

---

Description

Identification of all the multipath alternative splicing events in the splicing graphs.

Usage

```
EventDetectionMultipath(Input, cores, Path, paths = 2)
```

Arguments

Input	Output of the PrepareBam_EP function.
cores	Number of cores used for parallel processing.
Path	Directory where to write the EventsFound_RNASeq.txt file.
paths	Maximum number of paths of the events to find.

Value

List with all the events found for all the genes present in the experiment and a file called EventsFound_RNASeq.txt with the information each event.

Examples

```
## Not run:
  # Run EventDetection function
   data(SG_RNASeq)
   TxtPath<-tempdir()
   AllEvents_RNASeq_MP<-EventDetectionMultipath(SG_RNASeq,cores=1,Path=TxtPath,paths=3)

## End(Not run)
```

EventDetection_transcriptome
EventDetection_transcriptome

Description

Finds all the possible alternative splicing (AS) events given a reference transcriptome. This function use parallel foreach. User must set the value of cores (by default equal to one). Moreover, it will create a .txt file with the relative information of all the AS events found. Besides, it will return a list with main information of the splicing graph of each event. This list will be used as an input in downstream functions (Get_PSI_FromTranRef, FindPrimers, and EventPointer_RNASeq_TranRef_IGV)

Usage

```
EventDetection_transcriptome(inputFile = NULL, Pathtxt = NULL, cores = 1)
```

Arguments

inputFile	Path to the GTF file of the reference transcriptome.
Pathtxt	Directory to save the .txt of the events found
cores	Number of cores using in the parallel processing (by default = 1)

Value

a list is returned with the following information:

ExTP1 a sparce matrix of Events x Transcripts that relates which isoform build up the path1 of each event.

ExTP2 a sparce matrix of Events x Transcripts that relates which isoform build up the path2 of each event.

ExTPRef a sparce matrix of Events x Transcripts that relates which isoform build up the pathRef of each event.

transcritnames a vector with the annotation names of the isoforms.

SG_List A list containing the information of the splicing graph of each gene.

Examples

```
## Not run:
      PathFiles<-system.file("extdata",package="EventPointer")
      inputFile <- paste(PathFiles,"/gencode.v24.ann_2genes.gtf",sep="")
      Pathtxt <- tempdir()


      # Run the function

      EventXtrans <- EventDetection_transcriptome(inputFile = inputFile,
                                                  Pathtxt=Pathtxt,
                                                  cores=1)

## End(Not run)
```

EventPointer *EventPointer*

Description

Statistical analysis of alternative splicing events

Usage

```
EventPointer(
  Design,
  Contrast,
  ExFit,
  Eventstxt,
  Filter = TRUE,
  Qn = 0.25,
  Statistic = "LogFC",
  PSI = FALSE
)
```

Arguments

Design	A matrix defining the linear model. Each row corresponds to an array, and each column corresponds to a coefficient (such as the baseline and treatment effects).
Contrast	A numeric matrix with contrasts to be tested. Rows correspond to coefficients in the design matrix, and columns correspond to contrasts.
ExFit	aroma.affymetrix pre-processed variable after using `extractDataFrame(affy, addNames=TRUE)`
Eventstxt	Path to the EventsFound.txt file generated by CDFfromGTF function.
Filter	Boolean variable to indicate if an expression filter is applied.
Qn	Quantile used to filter the events (Bounded between 0-1, Q1 would be 0.25).
Statistic	Statistical test to identify differential splicing events, must be one of : LogFC, Dif_LogFC or DRS.
PSI	Boolean variable to indicate if Delta PSI should be calculated for every splicing event.

Value

Data.frame ordered by the splicing p-value. The object contains the different information for each splicing event such as Gene name, event type, genomic position, p-value, z.value and delta PSI.

Examples

```
data(ArraysData)

Dmatrix<-matrix(c(1,1,1,1,0,0,1,1),nrow=4,ncol=2,byrow=FALSE)
Cmatrix<-t(t(c(0,1)))
EventsFound<-paste(system.file('extdata',package='EventPointer'),'/EventsFound.txt',sep='')

Events<-EventPointer(Design=Dmatrix,
```

```
                    Contrast=Cmatrix,
                    ExFit=ArraysData,
                    Eventstxt=EventsFound,
                    Filter=TRUE,
                    Qn=0.25,
                    Statistic='LogFC',
                    PSI=TRUE)
```

EventPointerBAM_IGV *EventPointer RNASeq IGV Visualization*

Description

Generates files to be loaded in IGV for visualization and interpretation of events

Usage

```
EventPointerBAM_IGV(SG_RNASeq, EventsCSV, PathGTF)
```

Arguments

SG_RNASeq	Output from EventDetection_BAM function. Contains splicing graphs components.
EventsCSV	Path to EventsFound.txt file generated with EventDetection_BAM function
PathGTF	Directory where to write the GTF files.

Value

Creates paths_RNASeq.gtf file that represents the alternative splicing events.

Examples

```
## Not run:
  data(AllEvents_RNASeq)
  data(SG_RNASeq)

   # Run EventPointer

   Dmatrix<-matrix(c(1,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1),ncol=2,byrow=FALSE)
   Cmatrix<-t(t(c(0,1)))
  Events <- EventPointer_RNASeq(AllEvents_RNASeq,Dmatrix,Cmatrix,Statistic='LogFC',PSI=TRUE)

   # IGV Visualization

  EventsCSV<-paste(system.file('extdata',package='EventPointer'),'/EventsFound_RNASeq.txt',sep='')
   PathGTF<-tempdir()
   EventPointerBAM_IGV(Events,SG_RNASeq,EventsCSV,PathGTF)

## End(Not run)
```

`EventPointerStats_BAM` *Statistical analysis of alternative splicing events from BAM files*

Description

Performs statistical testing on PSI values derived from BAM files to identify differential splicing events between conditions. Uses bootstrap resampling for robust statistical inference.

Usage

```
EventPointerStats_BAM(
  PSI_boots,
  Design,
  Contrast,
  Threshold = 0,
  nbootstraps = 1000,
  cores = 1,
  ram = 0.1,
  pathResult = "./"
)
```

Arguments

`PSI_boots`	PSI_boots.RData obtained from `EventsDetection_BAM` function. Contains PSI values and bootstrap information for each event.
`Design`	A matrix defining the linear model. Each row corresponds to a sample, and each column corresponds to a coefficient (such as the baseline and treatment effects).
`Contrast`	A numeric matrix with contrasts to be tested. Rows correspond to coefficients in the design matrix, and columns correspond to contrasts.
`Threshold`	Numeric. Threshold value for computing p-values. Default is 0.
`nbootstraps`	Integer. Number of bootstrap iterations to use for statistical testing. Higher numbers increase computational time but improve statistical power. Default is 1000.
`cores`	Integer. Number of cores to use for parallel processing. Default is 1.
`ram`	Numeric. Amount of RAM (in GB) to use for computations. Default is 0.1.
`pathResult`	Path where results will be saved. A subdirectory "EventPointerStatsResult" will be created containing a table with the results of the differential Ψ analysis for each contrast. The table presents the $\Delta\Psi$ associated with each event and its corresponding significance parameters. Note that a separate table will be generated for each contrast specified in the contrast matrix. Default is current directory ("./").

Value

Creates result files in the specified path containing statistical analysis results, including p-values and effect sizes for each contrast. Files are saved in CSV format in the "bootstrapResult" subdirectory.

Examples

```
data(PSI_boots)

Design <- cbind(rep(1, 9), rep(c(1, 0, 0), 3), rep(c(0, 1, 0), 3))
Contrast <- cbind(c(0, 1, 0), c(0, 0, 1))

PathSGResult <- tempdir()

EventPointerStats_BAM(
  PSI_boots,
  Design,
  Contrast,
  Threshold = 0,
  nbootstraps = 1000,
  cores = 1,
  ram = 0.1,
  pathResult = PathSGResult
)
```

EventPointer_Bootstraps

EventPointer_Bootstraps

Description

Statistical analysis of alternative splicing events with bootstrap technique.

Usage

```
EventPointer_Bootstraps(
  PSI,
  Design,
  Contrast,
  nbootstraps = 10000,
  UsePseudoAligBootstrap = TRUE,
  Threshold = 0,
  cores = 1,
  ram = 0.1
)
```

Arguments

PSI	Array or matrix that contains the values of PSI calculated in the function GetPSIFromTranRef. If bootstrap option was selected in GetPSIFromTranRef, input must be an array. If not, input must be a matrix.
Design	A matrix defining the experimental design. Rows represent samples and columns conditions.
Contrast	A numeric matrix with contrasts to be tested. Rows correspond to coefficients in the design matrix, and columns correspond to contrasts.

nbootstraps	How many layers, Bootstraps or samplings are going to be used. Caution, high numbers increase computational time.
UsePseudoAligBootstrap	TRUE (default) if bootstrap data from pseudoalignment want to be used or FALSe if not.
Threshold	it assigns a threshold to compute the pvalues. Default value is 0.
cores	The number of cores desired to use.
ram	How many ram memory is used,in Gb.

Value

A list containing the summary of the Bootstrap analysis: DeltaPSI, Pvalues, FDR. This info can be obtained in a simple table with the function ResulTable.

Examples

```
data(PSIss)
PSI <- PSIss$PSI

Dmatrix <- cbind(1,rep(c(0,1),each=2))
Cmatrix <- matrix(c(0,1),nrow=2)

Fit <- EventPointer_Bootstraps(PSI = PSI,
                               Design = Dmatrix,
                               Contrast = Cmatrix,
                               nbootstraps = 10,
                               UsePseudoAligBootstrap = TRUE,
                               Threshold = 0,
                               cores = 1,
                               ram = 1)
```

EventPointer_IGV *EventPointer IGV Visualization*

Description

Generates of files to be loaded in IGV for visualization and interpretation of events

Usage

```
EventPointer_IGV(
  Events,
  input,
  inputFile = NULL,
  PSR,
  Junc,
  PathGTF,
  EventsFile,
  microarray = NULL
)
```

Arguments

Events	Data.frame generated by EventPointer with the events to be included in the GTF file.
input	Reference transcriprome. Must be one of: 'Ensembl', 'UCSC' , 'AffyGTF' or 'CustomGTF'.
inputFile	If input is 'AffyGTF' or 'CustomGTF', inputFile should point to the GTF file to be used.
PSR	Path to the Exon probes txt file.
Junc	Path to the Junction probes txt file.
PathGTF	Directory where to write the GTF files.
EventsFile	Path to EventsFound.txt file generated with CDFfromGTF function.
microarray	Microarray used to create the CDF file. Must be one of: HTA-2_0, ClariomD, RTA or MTA.

Value

The function displays a progress bar to show the user the progress of the function. Once the progress bar reaches 100 in PathGTF. The created files are: 1) paths.gtf: GTF file representing the alternative splicing events. 2) probes.gtf: GTF file representing the probes that measure each event and each path.

Examples

```
## Not run:
   PathFiles<-system.file('extdata',package='EventPointer')
   DONSON_GTF<-paste(PathFiles,'/DONSON.gtf',sep='')
   PSRProbes<-paste(PathFiles,'/PSR_Probes.txt',sep='')
   JunctionProbes<-paste(PathFiles,'/Junction_Probes.txt',sep='')
   Directory<-tempdir()

   data(ArraysData)

   Dmatrix<-matrix(c(1,1,1,1,0,0,1,1),nrow=4,ncol=2,byrow=FALSE)
   Cmatrix<-t(t(c(0,1)))
  EventsFound<-paste(system.file('extdata',package='EventPointer'),'/EventsFound.txt',sep='')

   Events<-EventPointer(Design=Dmatrix,
                        Contrast=Cmatrix,
                        ExFit=ArraysData,
                        Eventstxt=EventsFound,
                        Filter=TRUE,
                        Qn=0.25,
                        Statistic='LogFC',
                        PSI=TRUE)

 EventPointer_IGV(Events=Events[1,,drop=FALSE],
                  input='AffyGTF',
                  inputFile=DONSON_GTF,
                  PSR=PSRProbes,
                  Junc=JunctionProbes,
                  PathGTF=Directory,
                 EventsFile= EventsFound,
                 microarray='HTA-2_0')
```

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

`EventPointer_RNASeq` *Statistical analysis of alternative splcing events for RNASeq data*

Description

Statistical analysis of all the alternative splicing events found in the given bam files.

Usage

```
EventPointer_RNASeq(Events, Design, Contrast, Statistic = "LogFC", PSI = FALSE)
```

Arguments

`Events`	Output from EventDetection function
`Design`	The design matrix for the experiment.
`Contrast`	The contrast matrix for the experiment.
`Statistic`	Statistical test to identify differential splicing events, must be one of : LogFC, Dif_LogFC and DRS.
`PSI`	Boolean variable to indicate if PSI should be calculated for every splicing event.

Value

Data.frame ordered by the splicing p.value . The object contains the different information for each splicing event such as Gene name, event type, genomic position, p.value, z.value and delta PSI.

Examples

```
data(AllEvents_RNASeq)
Dmatrix<-matrix(c(1,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1),ncol=2,byrow=FALSE)
Cmatrix<-t(t(c(0,1)))
Events <- EventPointer_RNASeq(AllEvents_RNASeq,Dmatrix,Cmatrix,Statistic='LogFC',PSI=TRUE)
```

`EventPointer_RNASeq_TranRef`
EventPointer_RNASeq_TranRef

Description

Statistical analysis of alternative splicing events with the output of GetPSI_FromTranRef

Usage

```
EventPointer_RNASeq_TranRef(
  Count_Matrix,
  Statistic = "LogFC",
  Design,
  Contrast
)
```

Arguments

Count_Matrix	The list containing the expression data taken from the ouput of GetPSI_FromTranRef
Statistic	The type of statistic to apply. Default = 'LogFC' (can be 'logFC, 'Dif_LogFC','DRS')
Design	The design matrix of the experiment.
Contrast	The Contrast matrix of the experiment.

Value

a data.frame with the information of the names of the event, its p.values and the corresponding z.value. If there is more than one contrast, the function returns as many data.frames as number of contrast and all these data.frame are sotred in an unique list.

Examples

```
## Not run:
   data(EventXtrans)
   data(PSIss)
   # Design and contrast matrix:

   Design <- matrix(c(1,1,1,1,0,0,1,1),nrow=4)
   Contrast <- matrix(c(0,1),nrow=2)

   # Statistical analysis:


   Fit <- EventPointer_RNASeq_TranRef(Count_Matrix =  PSIss$ExpEvs,
                                      Statistic = 'LogFC',Design = Design,
                                      Contrast = Contrast)

## End(Not run)
```

EventPointer_RNASeq_TranRef_IGV

EventPointer RNASeq from reference transcriptome IGV Visualization

---

Description

Generates of files to be loaded in IGV for visualization and interpretation of events detected from a reference transcriptome (see EventDetection_transcriptome).

Usage

```
EventPointer_RNASeq_TranRef_IGV(SG_List, pathtoeventstable, PathGTF)
```

Arguments

SG_List	List with the Splicing Graph information of the events. This list is created by EventDetection_transcriptome function.
pathtoeventstable	Complete path to the table returned by EventDetection_transcriptome that contains the information of each event, or table with specific events that the user want to load into IGV to visualize.
PathGTF	Directory where to write the GTF files.

Value

The function displays a progress bar to show the user the progress of the function. Once the progress bar reaches 100 file is written to the specified directory in PathGTF. The created file is named 'paths_RNASeq.gtf'.

Examples

```
###### example using all the events found in a reference transcriptome
data("EventXtrans")
SG_List <- EventXtrans$SG_List
PathEventsTxt<-system.file('extdata',package='EventPointer')
PathEventsTxt <- paste0(PathEventsTxt,"/EventsFound_Gencode24_2genes.txt")
PathGTF <- tempdir()

EventPointer_RNASeq_TranRef_IGV(SG_List = SG_List,pathtoeventstable = PathEventsTxt,PathGTF = PathGTF)
```

EventsDetection_BAM *Detection of alternative splicing events from BAM alignment files*

Description

Identifies alternative splicing events directly from BAM alignment files using a splicing graph approach. This function builds a splicing graph from read alignments, detects events, and calculates PSI values with bootstrap resampling.

Usage

```
EventsDetection_BAM(
  PathSamplesAbundance,
  PathTranscriptomeGTF = NULL,
  region = NULL,
  min_junction_count = 2,
  max_complexity = 30,
  min_n_sample = NULL,
```

```
  min_anchor = 6,
  nboot = 20,
  lambda = NULL,
  cores = 1,
  PathSGResult = ".",
  verbose = FALSE
)
```

Arguments

PathSamplesAbundance	Path to BAM and BAI files or path to folder containing BAM and BAI files.
PathTranscriptomeGTF	Path to file containing the regions to be analysed from the BAM files in GTF format.
region	Numerical vector indicating the index of positions (at chromosomal level) to be analysed from the GTF. Default is NULL so that all regions are analysed.
min_junction_count	Minimum number of junctions detected in the alignment to be considered in the splicing graph. Default is 2.
max_complexity	Maximum allowed complexity. If a locus exceeds this threshold, it is skipped with a warning. Complexity is defined as the maximum number of unique predicted splice junctions overlapping a given position. High complexity regions are often due to spurious read alignments and can slow down processing. Default is 30.
min_n_sample	Minimum number of samples that a junction must have to be considered. Default is NULL (automatically set to minimum of 3 or total number of samples).
min_anchor	Minimum number of aligned bases at one end of an exon to consider a junction. Default is 6.
nboot	Number of resamples of the quantification of the samples to perform bootstrap. Default is 20.
lambda	Lambda parameter for PSI calculation. Default is NULL.
cores	Number of cores to use for parallel processing. Default is 1.
PathSGResult	Path where results will be saved. The following 4 files are generated: • TotalEventsFound.csv: General data for total events detected in CSV format. • EventsDetection_EPBAM.RData: Raw data per event, paths of splicing graph and counts. • SgFC.RData: Contains the splicing graph in RData format. • PSI_boots.RData: Ψ per event and sample in RData format. Default is current directory (".")
verbose	Logical indicating whether to show warnings and messages. If FALSE, warnings from internal functions (e.g., makeTxDbFromGRanges) will be suppressed. Default is FALSE

Value

Invisibly returns NULL. Results are saved to files in PathSGResult.

Examples

```
## Not run:
PathSamplesAbundance <- system.file("extdata/bams", package = "EventPointer")
PathTranscriptomeGTF <- list.files(PathSamplesAbundance, "*.gtf", full.names = TRUE)

PathSGResult <- tempdir()

EventsDetection_BAM(
  PathSamplesAbundance,
  PathTranscriptomeGTF,
  region = 16,
  min_junction_count = 2,
  max_complexity = 30,
  min_n_sample = NULL,
  min_anchor = 6,
  nboot = 20,
  lambda = NULL,
  cores = 1,
  PathSGResult = PathSGResult
)

## End(Not run)
```

`Events_ReClassification` *Events_ReClassification*

Description

EventPointer can detect splicing events that cannot be cataloged in any of the canonical types (Cassette Exon, Alternative 3' or 5' splice site, retained intron and mutually exclusive exon). These events are classified as "Complex Events". With this function, EventPointer reclassifies these complex events according to how similar the event is to the canonical events. The same complex event can have several types. Further, EP adds a new type of event: "multiple skipping exon". These events are characterized by presenting several exons in a row as alternative exons. If there is only one alternative exon we would be talking about a "Cassette Exon".

Usage

```
Events_ReClassification(EventTable, SplicingGraph)
```

Arguments

`EventTable`	Table returned by EventDetection_transcriptome. Can be easily loaded using the function read.delim as data.frame.
`SplicingGraph`	A list with the splicing graph of all the genes of a reference transcriptome. This data is returned by the function EventDetection_transcriptome.

Value

A data.frame containing a new column with the new classification ('EventType_new'):

Examples

```
#load splicing graph
data("SG_reclassify")

#load table with info of the events
PathFiles<-system.file("extdata",package="EventPointer")
inputFile <- paste(PathFiles,"/Events_found_class.txt",sep="")
EventTable <- read.delim(file=inputFile)
#this table has the information of 5 complex events.

EventTable_new <- Events_ReClassification(EventTable = EventTable,
                                          SplicingGraph = SG_reclassify)
```

`EventXtrans` *Relationship between isoforms and events.*

Description

Relationship between isoforms and events.

Usage

```
data(EventXtrans)
```

Format

A `list` object `EventXtrans[[1]]` displays the isoform that build up the path1 of each event.

Value

EventXtrans object contains the relationship between the isoforms and the events. It is a list of 4 elements. the first three stored sparse matrices relating the isoforms with the events. The fourth element stores de names of the reference annotation used (isoforms names).

`FindPrimers` *FindPrimers*

Description

FindPrimers is the main function of the primers design option. The aim of this function is the design of PCR primers and TaqMan probes for detection and quantification of alternative splicing.

Depending on the assay we want to carry out the the algorithm will design the primers for a conventional PCR or the primers and TaqMan probes if we are performing a TaqMan assay.

In the case of a conventional PCR we will be able to detect the alternative splicing event. Besides, the algorithm gives as an output the length of the PCR bands that are going to appear. In the case of a TaqMan assay, we will not only detect but also quantify alternative splicing.

Usage

```
FindPrimers(
  SG,
  EventNum,
  Primer3Path,
  Dir,
  mygenomesequence,
  taqman = NA,
  nProbes = 1,
  nPrimerstwo = 3,
  ncommonForward = 3,
  ncommonReverse = 3,
  nExons = 5,
  nPrimers = 15,
  shortdistpenalty = 2000,
  maxLength = 1000,
  minsep = 100,
  wminsep = 200,
  valuethreePenalty = 1000,
  minexonlength = 25,
  wnpaths = 200,
  qualityfilter = 5000
)
```

Arguments

SG	Information of the graph of the gene where the selected event belongs. This information is avaible in the output of EventDetection_transcriptome function.
EventNum	The "EventNum" variable can be found in the returned .txt file from the EventDetection_transcriptome function in the column "EventNumber" or in the output of EventPointer_RNASeq_TranRef, the number after the "_" character of the 'Event_ID'.
Primer3Path	Complete path where primer3_core.exe is placed.
Dir	Complete path where primer3web_v4_0_0_default_settings.txt file and primer3_config directory are stored.
mygenomesequence	genome sequence of reference
taqman	TRUE if you want to get probes and primers for taqman. FALSE if you want to get primers for conventional PCR.
nProbes	Number of probes for Taqman experiments. By default 1.
nPrimerstwo	Number of potential exon locations for primers using two primers (one forward and one reverse). By default 3.
ncommonForward	Number of potential exon locations for primers using one primer in forward and two in reverse. By default 3.
ncommonReverse	Number of potential exon locations for primers using two primer in forward and one in reverse. By default 3.
nExons	Number of combinations of ways to place primers in exons to interrogate an event after sorting. By default 5.
nPrimers	Once the exons are selected, number of primers combination sequences to search within the whole set of potential sequences. By default 5.

`shortdistpenalty`
: Penalty for short exons following an exponential funciton(A * exp(-dist * shortdistpenalty)). By defautl 2000.

`maxLength`
: Max length of exons that are between primers and for paths once we have calculated the sequence. By default 1000.

`minsep`
: Distance from which it is penalized primers for being too close By default 100.

`wminsep`
: Weigh of the penalization to primers for being too close By default 200.

`valuethreePenalty`
: penalization for cases that need three primers instead of 2. By default 1000.

`minexonlength`
: Minimum length that a exon has to have to be able to contain a primer. By default 25.

`wnpaths`
: Penalty for each existing path By default 200.

`qualityfilter`
: Results will show as maximum 3 combinations with a punctuation higher than qualityfilter By default 5000.

Value

The output of the function is a 'data.frame' whose columns are:

For1Seq: Sequence of the first forward primer.

For2Seq: Sequence of the second forward primer in case it is needed.

Rev1Seq: Sequence of the first reverse primer.

Rev2Seq: Sequence of the second reverse primer in case it is needed.

For1Exon: Name of the exon of the first forward primer.

For2Exon: Name of the exon of the second forward primer in case it is needed.

Rev1Exon: Name of the exon of the first reverse primer.

Rev2Exon: Name of the exon of the second reverse primer in case it is needed.

FINALvalue: Final punctuation for that combination of exons and sequences. The lower it is this score, the better it is the combination.

DistPath1: Distances of the bands, in base pairs, that interrogate Path1 when we perform the conventional PCR experiment.

DistPath2: Distances of the bands, in base pairs, that interrogate Path2 'when we perform the conventional PCR experiment.

DistNoPath: Distances of the bands, in base pairs, that they do not interrogate any of the two paths when we perform the conventional PCR experiment.

SeqProbeRef: Sequence of the TaqMan probe placed in the Reference.

SeqProbeP1: Sequence of the TaqMan probe placed in the Path1.

SeqProbeP2: Sequence of the TaqMan probe placed in the Path2.

Examples

```
## Not run:

data("EventXtrans")
#From the output of EventsGTFfromTranscriptomeGTF we take the splicing graph information
SG_list <- EventXtrans$SG_List
#SG_list contains the information of the splicing graphs for each gene
```

```
#Let's supone we want to design primers for the event 1 of the gene ENSG00000254709.7

#We take the splicing graph information of the required gene
SG <- SG_list$ENSG00000254709.7

#We point the event number
EventNum <- 1

#Define rest of variables:
Primer3Path <- Sys.which("primer3_core")
Dir <- "C:\\PROGRA~2\\primer3\\"

MyPrimers <- FindPrimers(SG = SG,
                         EventNum = EventNum,
                         Primer3Path = Primer3Path,
                         Dir = Dir,
                         mygenomesequence = BSgenome.Hsapiens.UCSC.hg38::Hsapiens,
                         taqman = 1,
                         nProbes=1,
                         nPrimerstwo=4,
                         ncommonForward=4,
                         ncommonReverse=4,
                         nExons=10,
                         nPrimers =5,
                         maxLength = 1200)



## End(Not run)
```

`Fit` *Result of EventPointer_Bootstrap*

Description

Result of EventPointer_Bootstrap

Usage

```
data(Fit)
```

Format

A `list` object.

Value

A list containing the summary of the Bootstrap analysis: DeltaPSI, Pvalues, FDR. This info can be obtained in a simple table with the function ResulTable.

getbootstrapdata *getbootstrapdata*

Description

Function to load the values of the bootstrap returned by kallisto or salmon pseudoaligners.

Usage

```
getbootstrapdata(PathSamples, type)
```

Arguments

PathSamples	A vector with the complete directory to the folder of the output of kallisto/salmon.
type	'kallisto' or 'salmon'.

Value

A list containing the quantification data with the bootstrap information.

Examples

```
PathSamples<-system.file("extdata",package="EventPointer")
PathSamples <- paste0(PathSamples,"/output")
PathSamples <- dir(PathSamples,full.names = TRUE)

data_exp <- getbootstrapdata(PathSamples = PathSamples,type = "kallisto")
```

GetPSI_FromTranRef *GetPSI_FromTranRef*

Description

Get the values of PSI. A filer expression is applied if the user select the option of filter.

Usage

```
GetPSI_FromTranRef(
  Samples,
  PathsxTranscript,
  Bootstrap = FALSE,
  Filter = TRUE,
  Qn = 0.25
)
```

Arguments

Samples	matrix or list containing the expression of the samples.
PathsxTranscript	the output of EventDetection_transcriptome.
Bootstrap	Boolean variable to indicate if bootstrap data from pseudo-alignment is used.
Filter	Boolean variable to indicate if an expression filter is applied. Default TRUE.
Qn	Quartile used to filter the events (Bounded between 0-1, Qn would be 0.25 by default).

Value

The output is a list containing two elements: a matrix with the values of PSI and a list containing as many matrices as number of events. In each matrix is stored the expression of the different paths of an event along the samples.

Examples

```
data(EventXtrans)

PathSamples <- system.file("extdata",package="EventPointer")
PathSamples <- paste0(PathSamples,"/output")
PathSamples <- dir(PathSamples,full.names = TRUE)

data_exp <- getbootstrapdata(PathSamples = PathSamples,type = "kallisto")

#same annotation
rownames(data_exp[[1]]) <- gsub("\\|.*","",rownames(data_exp[[1]]))

#Obtain values of PSI
PSI_List <- GetPSI_FromTranRef(PathsxTranscript = EventXtrans,Samples = data_exp,Bootstrap = TRUE, Filter =
PSI <- PSI_List$PSI
Expression_List <- PSI_List$ExpEvs
```

---

getPSI_RNASeq_boot *getPSI_RNASeq_boot*

Description

Calculates PSI (Percent Spliced In) values with bootstrap resampling for RNA-Seq data. This function estimates PSI values using a median polish approach and generates bootstrap samples to assess the variability of PSI estimates.

Usage

```
getPSI_RNASeq_boot(Result, lambda = NULL, cores = 1, nboot = 20)
```

Arguments

Result	An EventPointer result object from RNA-Seq analysis (e.g., from EventsDetection_BAM). Must contain count matrices that can be extracted with getCountMatrix.
lambda	Regularization parameter for PSI estimation. Default is 0.1 if NULL. Controls the smoothness of PSI estimates.
cores	The number of cores to use for parallel computation. Default is 1.
nboot	Number of bootstrap iterations to perform. Default is 20. Higher values provide more stable estimates but increase computation time.

Details

The function performs the following steps:

1. Extracts count matrices from the Result object
2. Applies median polish to log-transformed count ratios to estimate normalization factors
3. Performs parallel bootstrap resampling to generate multiple PSI estimates
4. Combines original PSI with bootstrap samples into a single array

The bootstrap procedure uses parallel processing to distribute computations across multiple cores.

Value

A 3-dimensional array with dimensions (events, bootstrap samples + 1, samples). The first layer (bootstrap sample 1) contains the original PSI estimates, followed by nboot bootstrap replicates.

Examples

```
## Not run:
# Assuming 'eventsResult' is an EventPointer RNA-Seq result object
PSI_boot <- getPSI_RNASeq_boot(Result = eventsResult,
                               lambda = 0.1,
                               cores = 4,
                               nboot = 20)

# Access original PSI estimates
original_PSI <- PSI_boot[, 1, ]

# Access bootstrap samples
bootstrap_PSI <- PSI_boot[, 2:(nboot+1), ]

## End(Not run)
```

InternalFunctions *EventPointer Internal Functions*

Description

Internal functions used by EventPointer in the different steps of the algorithm

Usage

```
annotateEvents(Events, PSR_Gene, Junc_Gene, Gxx)

annotateEventsMultipath(Events, PSR_Gene, Junc_Gene, Gxx, paths)

AnnotateEvents_RNASeq(Events)

AnnotateEvents_RNASeq_MultiPath(Events, paths)

AnnotateEvents_KLL(Events, Gxx, GenI)

reClassificationIntern(SG, Event)

ClassifyEvents(SG, Events, twopaths)

estimateAbsoluteConc(Signal1, Signal2, SignalR, lambda)

estimateAbsoluteConcmultipath(datos, lambda = 0.1)

findTriplets(randSol, tol = 1e-08)

findTriplets2(Incidence, paths = 2, randSol)

GetCounts(Events, sg_txiki, type = "counts")

getPathCounts(x, readsC, widthinit)

getPathFPKMs(x, readsC, widthinit)

GetCountsMP(Events, sg_txiki, type = "counts")

getPathCountsMP(x, readsC, widthinit)

getPathFPKMsMP(x, readsC, widthinit)

getEventPaths(Events, SG)

getEventMultiPaths(Events, SG, twopaths, paths)

GetIGVPaths(EventInfo, SG_Edges)

getPSI(ExFit, lambda = 0.1)
```

```
getPSImultipath(ExFit, lambda = 0.1)

getPSI_RNASeq(Result, lambda = 0.1)

getCountMatrix(Result, modeFill = "FPKM")

estimateAbsoluteConc_boot(
  Signal1,
  Signal2,
  SignalR,
  lambda = NULL,
  l1 = 1,
  l2 = 1,
  lR = 1
)

estimatePSI(CountMatrix, l1eq, l2eq, lReq, lambda = NULL)

calcBootstrapPSI(
  rowProcess,
  PSI_boot,
  CountMatrix,
  l1eq,
  l2eq,
  lReq,
  lambda,
  nboot
)

getPSI_RNASeq_MultiPath(Result, lambda = 0.1)

getRandomFlow(Incidence, ncol = 1)

IHsummarization(Pv1, t1, Pv2, t2, coherence = "Opposite")

pdist2(X, Y)

PrepareCountData(Result)

PrepareProbes(Probes, Class)

PrepareOutput(Result, Final)

SG_Info(SG_Gene)

SG_creation(SG_Gene)

SG_creation_RNASeq(SG_Gene)

SG_creation_fast(SG_Gene)

WriteGTF(PATH, Data, Probes, Paths)
```

```
WriteGTF_RNASeq(PATH, Data, Paths)

flat2Cdf(
  file,
  chipType,
  tags = NULL,
  rows = 2560,
  cols = 2560,
  verbose = 10,
  xynames = c("X", "Y"),
  gcol = 5,
  ucol = 6,
  splitn = 4,
  col.class = c("integer", "character")[c(1, 1, 1, 2, 2, 2)],
  Directory = getwd(),
  ...
)

uniquefast(X)

filterimagine(Info, paths)

transfromedge(SG, SG_Gene)

sacartranscritos(edgetr, events)

convertToSGFeatures2(x, coerce = FALSE, merge = FALSE)

processFeatures2(features, coerce = FALSE, merge = FALSE)

annotate2(query, subject)

annotateFeatures2(query, subject)

mergeExonsTerminal2(features, min_n_sample = 1)

PrimerSequenceGeneral(
  taqman,
  FinalExons,
  generaldata,
  SG,
  Dir,
  nPrimers,
  Primer3Path = Sys.which("primer3_core"),
  maxLength,
  minsep,
  wminsep,
  valuethreePpenalty,
  wnpaths,
  qualityfilter,
  mygenomesequence
```

```
)

PrimerSequenceTwo(
  FinalExons,
  SG,
  generaldata,
  n,
  thermo.param,
  Primer3Path,
  settings,
  mygenomesequence
)

ProbesSequence(
  SG,
  FinalSeq,
  generaldata,
  Dir,
  Primer3Path = Sys.which("primer3_core"),
  nProbes,
  mygenomesequence
)

sort_exons(namesPath, decreasing = FALSE)

all_simple_paths2(wg, from, to, ...)

callPrimer3(
  seq,
  threeprimers = FALSE,
  pr,
  reverse = FALSE,
  size_range = "150-500",
  Tm = c(57, 59, 62),
  name = "Primer1",
  Primer3Path = "primer3-2.3.7/bin/primer3_core",
  thermo.param = "primer3-2.3.7/src/primer3_config/",
  sequence_target = NULL,
  settings = "primer3-2.3.7/primer3web_v4_0_0_default_settings.txt"
)

callPrimer3probes(
  seq,
  name = "Primer1",
  Primer3Path = "primer3-2.3.7/bin/primer3_core",
  thermo.param = "primer3-2.3.7/src/primer3_config/",
  sequence_target = NULL,
  settings = "primer3-2.3.7/primer3web_v4_0_0_default_settings.txt"
)

CreateSequenceforProbe(SG, Exons, FinalSeq, n, mygenomesequence)
```

```
findPotencialExons(D, namesPath, maxLength, SG, minexonlength)

fullExons(namesPath)

includeaexons(Forward)

genreverse(FinalInfo, taqman)

getDistanceseachPath(Exon1, Exon2, generaldata, distinPrimers, SG)

getDominants2(
  PrimersTwo,
  Primers1,
  commonForward,
  commonReverse,
  namesRef,
  D,
  numberOfPaths,
  nprimerstwo,
  ED,
  wNpaths = 1000,
  wP12inRef = 1000
)

getDominantsFor(
  Primers1,
  Primers2,
  commonForward,
  namesRef,
  D,
  numberOfPaths,
  Event,
  ncommonForward,
  ED,
  wNpaths = 1000,
  wP12inRef = 1000
)

getDominantsRev(
  Primers1,
  Primers2,
  commonReverse,
  namesRef,
  D,
  numberOfPaths,
  Event,
  ncommonReverse,
  ED,
  wNpaths = 1000,
  wP12inRef = 1000
)
```

```
getExonsFullSignal(namesPath, SG)

getFinalExons(
  generaldata,
  maxLength,
  nPrimerstwo,
  ncommonForward,
  ncommonReverse,
  nExons,
  minsep,
  wminsep,
  valuethreePpenalty,
  minexonlength
)

getgeneraldata(SG, Event, shortdistpenalty)

getrankexons(
  SG,
  Dominants,
  nt,
  wg,
  items,
  minsep,
  wminsep,
  valuethreePpenalty,
  D
)

getranksequence(
  taqman,
  Fdata,
  maxLength,
  minsep,
  wminsep,
  valuethreePpenalty,
  wnpaths,
  qualityfilter
)

PrimerSequenceCommonFor(
  FinalExons,
  SG,
  generaldata,
  n,
  thermo.param,
  Primer3Path,
  settings,
  mygenomesequence
)

PrimerSequenceCommonRev(
```

```
  FinalExons,
  SG,
  generaldata,
  n,
  thermo.param,
  Primer3Path,
  settings,
  mygenomesequence
)

get_beta(combboots, incrPSI_original, ncontrastes)

get_table(
  PSI_arrayP,
  nevents,
  totchunk,
  chunk,
  nsamples,
  incrPSI_original,
  V,
  nboot,
  nbootin,
  ncontrastes
)

get_YB(PSI_arrayS, l, nsamples, I, J, CTEind)

getInfo(table, ncontrast)

checkContrastDesignMatrices(C, D)

mclapplyPSI_Bootstrap(
  PSI_boots,
  Design,
  Contrast,
  cores,
  ram,
  nbootstraps,
  KallistoBootstrap,
  th,
  verbose = 0
)

call_get_table_Bootstrap(
  chunklist,
  Design,
  Contrast,
  nbootstraps,
  KallistoBootstrap,
  th,
  cores
)
```

```
get_table_Bootstrap(
  PSI_arrayP,
  Design,
  Contrast,
  nbootstraps,
  KallistoBootstrap,
  th
)

pvalue_incr_PSI(incr_PSI, th = 0, verbose = 0)

calculateCorrelationTest(A, B, method = c("pearson", "spearman"))

x %in2% table

callGRseq_parallel(EventsFound, SG_List, cores, typeA, nt)

getpij(A)

speedglm.wfit2(
  y,
  X,
  intercept = TRUE,
  weights = NULL,
  row.chunk = NULL,
  family = gaussian(),
  start = NULL,
  etastart = NULL,
  mustart = NULL,
  offset = NULL,
  acc = 1e-08,
  maxit = 25,
  k = 2,
  sparselim = 0.9,
  camp = 0.01,
  eigendec = TRUE,
  tol.values = 1e-07,
  tol.vectors = 1e-07,
  tol.solve = .Machine$double.eps,
  sparse = NULL,
  method = c("eigen", "Cholesky", "qr"),
  trace = FALSE,
  ...
)

dgl(x, med = 0, iqr = 1, chi = 0, xi = 0.6, maxit = 1000L)

fitgl(
  x,
  start,
  inc = FALSE,
```

```
  na.rm = FALSE,
  method = c("mle", "hist", "prob", "quant", "shape"),
  ...
)

pgl(q, med = 0, iqr = 1, chi = 0, xi = 0.6, maxit = 1000L)

qdgl(p, med = 0, iqr = 1, chi = 0, xi = 0.6)

qgl(p, med = 0, iqr = 1, chi = 0, xi = 0.6)

rgl(n, med = 0, iqr = 1, chi = 0, xi = 0.6)

callGRseq_parallel(EventsFound, SG_List, cores, typeA, nt)

getpij(A)

speedglm.wfit2(
  y,
  X,
  intercept = TRUE,
  weights = NULL,
  row.chunk = NULL,
  family = gaussian(),
  start = NULL,
  etastart = NULL,
  mustart = NULL,
  offset = NULL,
  acc = 1e-08,
  maxit = 25,
  k = 2,
  sparselim = 0.9,
  camp = 0.01,
  eigendec = TRUE,
  tol.values = 1e-07,
  tol.vectors = 1e-07,
  tol.solve = .Machine$double.eps,
  sparse = NULL,
  method = c("eigen", "Cholesky", "qr"),
  trace = FALSE,
  ...
)

hyperGeometricApproach(ExS, nSel, P_value_PSI, significance, resPred, N)

poissonBinomialApproach(ExS, nSel, P_value_PSI, significance, resPred, N)

significanceFunction(P_value_PSI, cSel, nSel, significance)

hyperMatrixRes(cSel, nSel, ExS, P_value_PSI, significance, N)

GseaApproach(P_value_PSI, ExS, significance, resPred, PSI_table = NULL)
```

```
WilcoxonApproach(
  P_value_PSI,
  ExS,
  significance,
  resPred,
  PSI_table = NULL,
  nSel,
  N
)

Wilcoxon.z.matrix(
  ExprT,
  GeneGO,
  alternative = c("two.sided", "less", "greater"),
  mu = 0,
  paired = FALSE,
  exact = NULL,
  correct = TRUE,
  conf.int = FALSE,
  conf.level = 0.95
)

MatrixRes(cSel, nSel, ExS, P_value_PSI, significance, N, nmTopEv)

myphyper(p, m, n, k, lower.tail = TRUE, log.p = FALSE)

reclasify_intern(SG, mievento, pp1, pp2, ppref)

sampleWhichCount(sample_info, list_features)

listCHRFileCount(which, valSample, list_features)

getSGFeatureCounts(
  sample_info,
  features,
  min_anchor = 6,
  counts_only = FALSE,
  retain_coverage = FALSE,
  verbose = FALSE,
  cores = 1
)

getSGFeatureCountsTotal(mainCount, min_anchor, retain_coverage, verbose)

processCounts(
  gap,
  features,
  strand,
  sample_name,
  min_anchor,
  retain_coverage,
```

```
  verbose
)

junctionCompatible(
  junctions,
  frag_exonic,
  frag_intron,
  min_anchor,
  counts = TRUE
)

filterIntrons(frag_intron, frag_exonic, min_anchor)

dimMatrixFix(matrix1, matrix2)

splicesiteOverlap(
  splicesites,
  side,
  frag_exonic,
  frag_intron,
  min_anchor,
  include = c("all", "spliced", "unspliced"),
  counts = TRUE
)

exonCompatible(
  exons,
  spliceL,
  spliceR,
  frag_exonic,
  frag_intron,
  counts = TRUE
)

findOverlapsRanges(query, subject, type = "any", out = "list")

modFindOverlap(
  query_unlisted,
  subject_unlisted,
  subject_togroup,
  lenQuery,
  lenSubject,
  out
)

splicesiteCounts(
  x,
  frag_exonic,
  frag_intron,
  min_anchor,
  option = c("junction", "exon"),
  include
```

```
)

exonCoverage(exons, exons_i_frag, frag_exonic)

predictTxFeatures(
  sample_info,
  which = NULL,
  alpha = 2,
  psi = 0,
  beta = 0.2,
  gamma = 0.2,
  include_counts = FALSE,
  retain_coverage = FALSE,
  junctions_only = FALSE,
  min_junction_count = NULL,
  min_anchor = 6,
  max_complexity = 20,
  min_n_sample = 1,
  min_overhang = NA,
  verbose = FALSE,
  cores = 1
)

expandUnstrandedRanges(x)

validationParameters(sample_info, alpha, min_junction_count, which)

globalWhich(sample_info)

sampleWhichPredict(sample_info, alpha, min_junction_count, which, novo)

listCHRFilePredict(range, valSample, bam_index)

predictTxFeaturesTotal(
  sWhich,
  file_bam,
  paired_end,
  which,
  min_junction_count,
  psi,
  beta,
  gamma,
  min_anchor,
  include_counts,
  retain_coverage,
  junctions_only,
  max_complexity,
  sample_name,
  verbose
)

predictJunctions(
```

```
  frag_exonic,
  frag_intron,
  min_junction_count,
  psi,
  min_anchor,
  retain_coverage
)

readGapPair(file, paired_end, which, sample_name, verbose)

generateWarningMessage(fun_name, item, msg)

fragExonIntron(gap, strand, verbose)

togroup0(x)

constructPaired(
  frag_exonic,
  frag_intron,
  junctions_df,
  min_junction_count,
  psi,
  beta,
  gamma,
  min_anchor,
  include_counts,
  retain_coverage,
  junctions_only,
  max_complexity,
  sample_name,
  seqlevel,
  strand,
  si
)

co2str(seqlevel, start, end, strand)

predictSpliced(
  frag_exonic,
  frag_intron,
  junctions_df,
  min_junction_count,
  psi,
  beta,
  gamma,
  min_anchor,
  include_counts,
  retain_coverage,
  junctions_only,
  max_complexity,
  sample_name,
  seqlevel,
```

```
  strand
)

extractSplicesitesFromJunctions(junctions, type = c("L", "R"))

predictExonsTerminal(
  candidates,
  frag_exonic,
  frag_intron,
  relCov,
  min_anchor,
  type = c("exon_L", "exon_R"),
  include_counts,
  retain_coverage
)

predictCandidatesTerminal(islands, splicesites, type = c("exon_L", "exon_R"))

predictExonsInternal(
  candidates,
  frag_exonic,
  frag_intron,
  relCov,
  min_anchor,
  include_counts,
  retain_coverage
)

predictCandidatesInternal(islands, splicesites, frag_coverage, relCov)

constructGRangesFromRanges(x, seqname, strand, seqinfo)

AnnEventsFunc(EventsDetection_pred, EventsDetection_ann, cores)

getBamInfo(PathSamplesAbundance, region, cores = 1)

getBamInfoPerSample(sample_info, region)
```

Value

Internal outputs

`MyPrimers` *Data frame with primers design for conventional PCR*

Description

Data frame with primers design for conventional PCR

Usage

```
data(MyPrimers)
```

Format

A `data.frame` object displays the relative information for primers design for conventional PCR

Value

MyPrimers object contains a data.frame with the information of the design primers for conventioanl PCR.

`MyPrimers_taqman` *Data frame with primers design for taqman PCR*

Description

Data frame with primers design for taqman PCR

Usage

```
data(MyPrimers_taqman)
```

Format

A `data.frame` object displays the relative information for primers design for taqman PCR

Value

MyPrimers_taqman object contains a data.frame with the information of the design primers for taqman PCR.

`PrepareBam_EP` *Bam files preparation for EventPointer*

Description

Prepares the information contained in .bam files to be analyzed by EventPointer

Usage

```
PrepareBam_EP(
  PathSamplesAbundance,
  PathTranscriptomeGTF = NULL,
  region = NULL,
  min_junction_count = 2,
  max_complexity = 30,
  min_n_sample = NULL,
  min_anchor = 6,
  cores = 1,
  PathSGResult = ".",
  verbose = TRUE
)
```

Arguments

`PathSamplesAbundance`
: Path to BAM and BAI files or path to folder containing BAM and BAI files.

`PathTranscriptomeGTF`
: Path to file containing the regions to be analysed from the BAM files in GTF format.

`region`
: Numerical vector indicating the index of positions (at chromosomal level) to be analysed from the GTF. Default is NULL so that all regions are analysed.

`min_junction_count`
: Minimum number of junctions detected in the alignment to be considered in the splicing graph. Default is 2.

`max_complexity`
: Maximum allowed complexity. If a locus exceeds this threshold, it is skipped with a warning. Complexity is defined as the maximum number of unique predicted splice junctions overlapping a given position. High complexity regions are often due to spurious read alignments and can slow down processing. Default is 30.

`min_n_sample`
: Minimum number of samples that a junction must have to be considered. Default is NULL (automatically set to minimum of 3 or total number of samples).

`min_anchor`
: Minimum number of aligned bases at one end of an exon to consider a junction. Default is 6.

`cores`
: Number of cores to use for parallel processing. Default is 1.

`PathSGResult`
: Path where results will be saved. The following 4 files are generated:

  - TotalEventsFound.csv: General data for total events detected in CSV format.
  - EventsDetection_EPBAM.RData: Raw data per event, paths of splicing graph and counts.
  - SgFC.RData: Contains the splicing graph in RData format.
  - PSI_boots.RData: Ψ per event and sample in RData format.

  Default is current directory (".").

`verbose`
: Logical indicating whether to show warnings and messages. If FALSE, warnings from internal functions (e.g., makeTxDbFromGRanges) will be suppressed. Default is TRUE.

Value

SGFeaturesCounts object. It contains a GRanges object with the corresponding elements to build the different splicing graphs found and the counts related to each of the elements.

`Protein_Domain_Enrichment` *Protein_Domain_Enrichment*

Description

Analyze whether the presence of a protein domain increases or decreases in the condition under study.

Usage

```
Protein_Domain_Enrichment(PathsxTranscript, TxD, Diff_PSI, method = "spearman")
```

Arguments

PathsxTranscript	the output of EventDetection_transcriptome.
TxD	matrix that relates transcripts with Protein domain. Users can get it from BioMart
Diff_PSI	matrix with the difference of psi of the condition under study. Can get it from the output of EventPointer_Bootstraps
method	a character string indicating which correlation coeffcient is to be calculated. "spearman" (default) or "pearson" can be selected.

Value

A list containing the results of the protein domain enrichment anaylisis. This list contains 3 matrices in which the rows indicate the protein domains and the columns the number of contrasts. The 3 matrices are the following:

-mycor: correlation value between the deltaPSI and the DifProtDomain matrix (see more details in vignette)

-STATISTIC: the values of the test statistic

-PVAL: the pvalues of the test statistic

Examples

```
## Not run:
   data("EventXtrans")
   data("TxD")
   data("Fit")

   #same annotation in TxD and EventXtrans
   transcriptnames <- EventXtrans$transcritnames
   transcriptnames <- gsub("\\..*","",transcriptnames)
   EventXtrans$transcritnames <- transcriptnames

   Result_PDEA <- Protein_Domain_Enrichment(PathsxTranscript = EventXtrans,
                                            TxD = TxD,
                                            Diff_PSI = Fit$deltaPSI)

## End(Not run)
```

PSIss *relationship between isoforms and events*

Description

relationship between isoforms and events

Usage

```
data(PSIss)
```

Format

A object `PSIss[[1]]` displays the values of PSI and `PSIss[[2]]` the valeus of expression.

Value

PSIss object the values of PSI calculated by the funcion GetPSI_FromTranRef and also the values of expression.

`PSI_boots` *Bootstrap PSI values from BAM files*

Description

Example dataset containing PSI (Percent Spliced In) values with bootstrap replicates obtained from RNA-Seq BAM files analysis.

Usage

```
PSI_boots
```

Format

A 3-dimensional array with dimensions (events, bootstrap samples, samples). The first layer contains original PSI estimates, followed by bootstrap replicates.

Source

Generated from EventsDetection_BAM function using example BAM files.

Examples

```
data(PSI_boots)
dim(PSI_boots)
```

PSI_Statistic *PSI_Statistic*

Description

Statistical analysis of the alternative splicing events. This function takes as input the values of PSI. Perform a statistical analysis based on permutation test

Usage

```
PSI_Statistic(PSI, Design, Contrast, nboot)
```

Arguments

PSI	A matrix with the values of the PSI.
Design	The design matrix for the experiment.
Contrast	The contrast matrix for the experiment.
nboot	The number of random analysis.

Value

The output of these functions is a list containing: two data.frame (deltaPSI and Pvalues) with the values of the deltaPSI and the p.values for each contrast, and a third element (LocalFDR) with the information of the local false discovery rate.

Examples

```
## Not run:
      data(ArraysData)
      PSI_Arrays_list<-EventPointer:::getPSI(ArraysData)
      PSI_Arrays <- PSI_Arrays_list$PSI
      Design <- matrix(c(1,1,1,1,0,0,1,1),nrow=4)
      Contrast <- matrix(c(0,1),nrow=1)

      # Statistical analysis:

     table <- PSI_Statistic(PSI_Arrays,Design = Design, Contrast = Contrast, nboot = 50)

## End(Not run)
```

ResulTable *ResulTable*

Description

Extract a table of the top-ranked events from the output of EventPointer_Bootstraps.

Usage

```
ResulTable(EP_Result,coef = 1,number = Inf)
```

Arguments

EP_Result	The output of the function EventPointer_Bootstraps
coef	Number specifying which coefficient or contrast of the model is of interest.
number	Maximum number of events to list

Value

A dataframe with a row for the number of top events and the following columns:

deltaPSI: the difference of PSI between conditions

pvalue: raw p-value

lfdr: local false discovery rate

qvalue: adjusted p-value or q-value

Examples

```
data(PSIss)
PSI <- PSIss$PSI

Dmatrix <- cbind(1,rep(c(0,1),each=2))
Cmatrix <- matrix(c(0,1),nrow=2)

Fit <- EventPointer_Bootstraps(PSI = PSI,
                               Design = Dmatrix,
                               Contrast = Cmatrix,
                               cores = 1,
                               ram = 1,
                               nbootstraps = 10,
                               UsePseudoAligBootstrap = TRUE)

ResulTable(EP_Result = Fit,coef = 1,number = 5)
```

SF_Prediction *Splicing Factor Prediction*

Description

Methodology to predict context-specific splicing factors

Usage

```
SF_Prediction(
  P_value_PSI,
  ExS,
  nSel = 1000,
  significance = NULL,
  method = "Fisher"
)
```

Arguments

P_value_PSI	A data.frame with the p.values of the experiment.
ExS	The ExS matrix biuldt in CreateExSmatrix function.
nSel	Top ranked events to be considered as spliced events.
significance	Threshold of P.value to consider which events are deferentially spliced. A vector of length equal to the number of contrasts. If null it will consider the nSel top ranked events.
method	methodology to apply: "Fisher" for Fisher's exact test (default), "PoiBin" for Poisson Binomial test, "Wilcoxon" for a wilcoxon test or "Gsea" for a test of kolmogorov smirnov

Value

The function returs a list. This list has for each contrast a data.frame containing the results of the prediction.

SG_reclassify *Splicing graph example for Events_ReClassification function*

Description

Splicing graph example for Events_ReClassification function

Usage

```
data(SG_reclassify)
```

Format

A `list` object `SG_reclassify[[i]]` displays the splicing graph of the ith gene.

Value

A list with the splicing graph of the 5 genes corresponding to the alternative splicing events depicted in the example of the function Events_ReClassification.

SG_RNASeq *Splicing graph elements predicted from BAM files*

Description

Splicing graph elements predicted from BAM files

Usage

```
data(SG_RNASeq)
```

Format

A `SGFeatureCounts` objects with predicted splicing graph features and counts

Value

SG_RNASeq object displays the predicted features found in the BAM files from the dataset published in Seshagiri et al. 2012 and used in the `SGSeq` R package vignette.

TxD *Transcript x Protein Domain matrix: small matrix for examples*

Description

Transcript x Protein Domain matrix: small matrix for examples

Usage

```
data(TxD)
```

Format

A `matrix` object

Value

A matrix containing the relates Transcripts with Protein Domains

Index