Package 'casper'

April 5, 2014

Version 1.4.0

Date 2013-10-11

Title Characterization of Alternative Splicing based on Paired-End Reads

Author David Rossell, Camille Stephan-Otto, Manuel Kroiss, Miranda Stobbe

Maintainer David Rossell <rosselldavid@gmail.org>

Depends R (>= 2.14.1), Biobase, IRanges, methods, gtools,GenomicRanges, Rsamtools, plyr, gaga

Imports VGAM, mgcv, GenomicFeatures, survival, sqldf

Enhances parallel

Description Infer alternative splicing from paired-end RNA-seq data. The model is based on counting paths across exons, rather than pairwise exon connections, and estimates the fragment size and start distributions non-parametrically, which improves estimation precision.

License GPL (>=2)

LazyLoad yes

Collate GenericDefs.R ClassDefinitions.R calcDenovo.R calcExp.R createDenovoGenome.R genePlot.R getDistrs.R getRoc.R denovoExpr.R mergeBatches.R mergeExp.R modelPrior.R pathCounts.R probNonEquiv.R procBam.R procGenome.R rmShortInserts.R simMultSamples.R simPost.R simReads.R splitGenomeByLength.R truncnorm.R wrapKnown.R

biocViews Bioinformatics, GeneExpression, DifferentialExpression,Transcription, RNASeq, HighThroughputSequencing

R topics documented:

`annotatedGenome-class` *Class "annotatedGenome"*

Description

Stores an annotated genome, either known or build de novo by combining a known genome with observed RNA-seq data.

Objects from the Class

Objects are typically created with a call to `procGenome` (for known genomes) or to `createDenovoGenome` (for de novo genomes).

Slots

islands `GRangesList` object with elements corresponding to gene islands. It indicates the start/end/name of each exon contained in the island

transcripts Each element in the list corresponds to a gene island. It indicates the exons contained in each known variant.

exon2island `data.frame` indicating the chromosome, start and end of each exon, and its corresponding gene island.

exonsNI `GRanges` indicating the chromosome, start/end and id of each exon

aliases `data.frame` indicating the aliases for each known transcript, i.e. transcripts having the exact same sequence of exons (as indicated in the UCSC database).

genomeVersion Character indicating the genome version from which the object was build, e.g. "hg19"

dateCreated Character indicating the date when the object was created. UCSC genomes chance from time to time, so that an "hg19" genome from Jan 2012 may not be exactly the same as in Dec 2012.

denovo Logical variable. `FALSE` indicates that the object was created using available annotation only. `TRUE` indicates that new exons/islands were added based on the data observed in a particular RNA-seq experiment.

txLength Numeric vector storing transcript lengths.

Methods

show `signature(object = "annotatedGenome")`: Displays general information about the object.

Author(s)

Camille Stephan-Otto Attolini

See Also

`procGenome` and `createDenovoGenome` to create `annotatedGenome` objects.

Examples

```
showClass("annotatedGenome")
```

calcDenovo *Estimate expression of gene splicing variants de novo.*

Description

calcDenovo estimates expression of gene splicing variants, considering both known variants and variants that have not been previously described.

relativeExpr computes relative expression estimates from calcDenovo output via Bayesian model selection or averaging.

Usage

```
calcDenovo(distrs, genomeDB, pc, readLength, islandid, priorq=3, mprior,
minpp=0.001, selectBest=FALSE, method="auto", niter, exactMarginal=TRUE,
verbose=TRUE, mc.cores=1)

relativeExpr(expr, summarize="modelAvg", minProbExpr=0.5, minExpr=0.05)
```

Arguments

distrs	List of fragment distributions as generated by the getDistrs function
genomeDB	denovoGenome object containing annotated genome, as returned by the createDenovoGenome function.
pc	Named vector of exon path counts as returned by pathCounts
readLength	Read length in bp, e.g. in a paired-end experiment where 75bp are sequenced on each end one would set readLength=75.
islandid	Name of the gene island to be analyzed. If not specified, all gene islands are analyzed.
priorq	Parameter of the prior distribution on the proportion of reads coming from each variant. The prior is Dirichlet with prior sample size for each variant equal to priorq. We recommend priorq=3 as this defines a non-local prior that penalizes including variants with low estimated expression unless they are warranted by the data.
mprior	Prior on the model space, as returned by modelPrior. This argument is used to favor splicing variant configurations that are consistent with the annotated genome. If left missing a uniform prior on the model space is used, i.e. all splicing variants configurations are equally likely a priori. We strongly recommend setting mprior via modelPrior. See details.
minpp	Models (i.e. splicing configurations) with posterior probability less than minpp are not reported. This argument can help reduce substantially the amount of required memory to store the results.
selectBest	If set to TRUE only the model with highest posterior probability is reported. While this can save memory, we do not recommend this option as it may ignore a substantial amount of uncertainty.

method	Method used to perform the model search. "allmodels" enumerates all possible models (warning: this is not feasible for genes with >5 exons). "rwmcmc" uses a random-walk MCMC scheme to focus on models with high posterior probability. "submodels" considers that some isoforms in genomeDB may not be expressed, but does not search for new variants. "auto" uses "allmodels" for genes with up to 5 exons and "rwmcmc" for longer genes. See details.
niter	Number of MCMC iterations.
exactMarginal	Set to FALSE to estimate posterior model probabilities as the proportion of MCMC visits. Set to TRUE to use the integrated likelihoods (default). See details.
integrateMethod	Method to compute integrated likelihoods. The default (plugin) evaluates likelihood*prior at the posterior mode and is the faster option. Set Laplace for Laplace approximations and IS for Importance Sampling. The latter increases computation cost very substantially.
verbose	Set to TRUE to display progress information.
mc.cores	Number of processors to be used for parallel computation. Can only be used if the package multicore is available for your system. Warning: using multiple processors substantially increases the memory requirements, so set this value carefully.
expr	Object of class denovoGenomeExpr, as returned by calcDenovo.
summarize	Set to "modelAvg" to average estimates across models according to the posterior probability of each model. Set to "bestModel" to report the posterior mode, i.e. the estimate under the model with highest posterior probabiliby.
minProbExpr	Variants with marginal posterior probability of being expressed below minProbExpr are not reported.
minExpr	Variants with estimated expression below minExpr are not reported.

Details

calcDenovo attempts to consider all possible splicing variant configurations for each gene. For example, a gene with 5 exons has 31 possible variants (2^5 - 1), each of which may be either expressed or not expressed. This gives rise to 2^31 - 1 possible configurations of expressed variants.

The routines computes the posterior probability of each model (i.e. configuration of expressed variants) via Bayes theorem.

P(modelly) "proportional to" m(y|model) P(model)

where m(y|model) is the integrated likelihood and P(model) is the prior probability of the model. Importantly, P(model) can be set by analyzing available genome annotations. For instance, genes with 20 exons have variants that tend to have close to 20 exons. They also tend to have more expressed variants than genes with 5 exons. The function modelPrior analyzes an annotated genome to set reasonable values for P(model).

Given that the number of models grows super-exponentially with the number of exons, an exhaustive enumeration of all possible models is not feasible unless the gene is very short (in practice, we set the limit at 5 exons). For longer genes, we use a random walk MCMC scheme that incorporates Shotgun Stochastic Search. This random exploration scheme focuses on models with large posterior probability. The longer the number of iterations (niter) the more stable the results, but this can come at a substantial computational cost.

In order to compute P(modelly) one can either use the computed m(y|model) P(model) (option `exactMarginal==TRUE`) or the proportion of MCMC visits (option `exactMarginal==FALSE`). Unless `niter` is large the former option typically provides more precise estimates.

Value

Expression set with expression estimates. `featureNames` identify each transcript via RefSeq ids, and the `featureData` contains Entrez identifiers and the length of each transcript (in bp). If `citype` was set to a value other than `"none"`, the `featureData` also contains the 95% credibility intervals (i.e. intervals that contain the true parameter value with 95% posterior probability).

Author(s)

Camille Stephan-Otto Attolini, Manuel Kroiss, David Rossell

References

Rossell D. QASPER: Quantifying Alternative Splicing from Paired End Reads. Technical report, IRB Barcelona (2010)

See Also

`denovoExpr` to obtain expression estimates from the `calcDenovo` output. `plotExpr` to produce a plot with splicing variants and estimated expression.

Examples

```
## See help(denovoExpr)
```

`calcExp` *Estimate expression of a known set of gene splicing variants.*

Description

Estimate expression of gene splicing variants, assuming that the set of variants is known. When `rpkm` is set to `TRUE`, fragments per kilobase per million are returned. Otherwise relative expression estimates are returned.

Usage

```
calcExp(distrs, genomeDB, pc, readLength, islandid, rpkm=TRUE, priorq=2,
priorqGeneExpr=2, citype="none", niter=10^3, burnin=100, mc.cores=1, verbose=FALSE)
```

Arguments

distrs	List of fragment distributions as generated by the getDistrs function
genomeDB	knownGenome object containing annotated genome, as returned by the procGenome function.
pc	Named vector of exon path counts as returned by pathCounts
readLength	Read length in bp, e.g. in a paired-end experiment where 75bp are sequenced on each end one would set readLength=75.
islandid	Name of the gene island to be analyzed. If not specified, all gene islands are analyzed.
rpkm	Set to FALSE to return relative expression levels, i.e. the proportion of reads generated from each variant per gene. These proportions add up to 1 for each gene. Set to TRUE to return fragments per kilobase per million (RPKM).
priorq	Parameter of the prior distribution on the proportion of reads coming from each variant. The prior is Dirichlet with prior sample size for each variant equal to priorq. We recommend priorq=2 for estimation, as it pools the estimated expression away from 0 and 1 and returned lower estimation errors than priorq=1 in our simulated experiments.
priorqGeneExpr	Parameter for prior distribution on overall gene expression. Defaults to 2, which ensures non-zero estimates for all genes
citype	Set to "none" to return no credibility intervals. Set to "asymp" to return approximate 95% CIs (obtained via the delta method). Set to "exact" to obtain exact CIs via Monte Carlo simulation. Options "asymp" and especially "exact" can increase the computation time substantially.
niter	Number of Monte Carlo iterations. Only used when citype=="exact".
burnin	Number of burnin Monte Carlo iterations. Only used when citype=="exact".
mc.cores	Number of processors to be used for parallel computation. Can only be used if the package multicore is available for your system.
verbose	Set to TRUE to display progress information.

Value

Expression set with expression estimates. featureNames identify each transcript via RefSeq ids, and the featureData contains further information. If citype was set to a value other than "none", the featureData also contains the 95% credibility intervals (i.e. intervals that contain the true parameter value with 95% posterior probability).

Author(s)

Camille Stephan-Otto Attolini, Manuel Kroiss, David Rossell

References

Rossell D, Stephan-Otto Attolini C, Kroiss M, Stocker A. Quantifying Alternative Splicing from Paired-End RNA-sequencing data. COBRA (http://biostats.bepress.com/cobra/art97)

Examples

```
data(K562.r1l1)
data(hg19DB)

#Pre-process
bam0 <- rmShortInserts(K562.r1l1, isizeMin=100)
pbam0 <- procBam(bam0)
head(getReads(pbam0))

#Estimate distributions, get path counts
distrs <- getDistrs(hg19DB,bam=bam0,readLength=75)
pc <- pathCounts(pbam0, DB=hg19DB)

#Get estimates
eset <- calcExp(distrs=distrs, genomeDB=hg19DB, pc=pc, readLength=75, rpkm=FALSE)
head(exprs(eset))
head(fData(eset))

#Add fake sample by permuting and combine
eset2 <- eset[sample(1:nrow(eset),replace=FALSE),]
esetall <- mergeExp(eset,eset2)

#After merge samples are correctly matched
head(exprs(esetall))
```

---

denovoExpr *Estimate expression for de novo splicing variants.*

Description

Obtains expression estimates from `denovoGenomeExpr` objects, as returned by `calcDenovo`. When `rpkm` is set to `TRUE`, fragments per kilobase per million are returned. Otherwise relative expression estimates are returned.

The estimates can be obtained by Bayesian model averaging (default) or by selecting the model with highest posterior probability. See details.

Usage

```
denovoExpr(x, pc, rpkm = TRUE, summarize = "modelAvg", minProbExpr = 0.5, minExpr = 0)
```

Arguments

`x`	`denovoGenomeExpr` object returned by `calcExp`
`pc`	Named vector of exon path counts as returned by `pathCounts`
`rpkm`	Set to `FALSE` to return relative expression levels, i.e. the proportion of reads generated from each variant per gene. These proportions add up to 1 for each gene. Set to `TRUE` to return fragments per kilobase per million (RPKM).

summarize	Set to "modelAvg" to obtain model averaging estimates, or to "bestModel" to select the model with highest posterior probability. We recommend the former, as even the best model may have low posterior probability.
minProbExpr	Variants with (marginal posterior) probability of being expressed below minProbExpr are omitted from the results. This argument is useful to eliminate variants that are not at least moderately supported by the data.
minExpr	Variants with relative expression minExpr are omitted from the results. This is useful to eliminate variants to which few reads are assigned, e.g. due to read miss-alignments or biases.

Value

Expression set with expression estimates. The featureData indicates the gene island id, posterior probability that each variant is expressed (column "probExpressed"), the number of aligned reads per gene island (column "explCnts") and the exons in each variant.

Author(s)

David Rossell

References

Rossell D, Stephan-Otto Attolini C, Kroiss M, Stocker A. Quantifying Alternative Splicing from Paired-End RNA-sequencing data. COBRA (http://biostats.bepress.com/cobra/art97)

Examples

```
## NOTE: toy example with few reads & genes to illustrate code usage
##       Results with complete data are much more interesting!

data(K562.r1l1)
data(hg19DB)

#Pre-process
bam0 <- rmShortInserts(K562.r1l1, isizeMin=100)
pbam0 <- procBam(bam0)

#Estimate distributions, get path counts
distrs <- getDistrs(hg19DB,bam=bam0,readLength=75)
pc <- pathCounts(pbam0, DB=hg19DB)

#Set prior distrib on model space
mprior <- modelPrior(hg19DB, maxExons = 40, smooth = TRUE, verbose=TRUE)

#Fit model
denovo <- calcDenovo(distrs, genomeDB=hg19DB, pc=pc, readLength=75, priorq = 3, mprior=mprior, minpp=0)

head(names(denovo))
denovo[[6499]]
posprob(denovo[[6499]])
```

```
#Get estimates
eset <- denovoExpr(denovo, pc=pc, rpkm=TRUE, minProbExpr=0.5)

head(exprs(eset))
head(fData(eset))
```

`denovoGeneExpr-class` *Class "denovoGeneExpr"*

Description

`denovoGeneExpr` stores inferred expression for de novo splicing variants for a single gene. `denovoGenomeExpr` stores the information for several genes (typically, the whole genome).

Objects from the Class

Objects are returned by `calcDenovo`. When running `calcDenovo` on multiple genes results are returned in a `denovoGenomeExpr` object. Results for a single gene can be retrieved using the [[operator as usual, which returns a `denovoGeneExpr` object.

Slots

posprob `data.frame` containing the posterior probability of each model

expression `data.frame` with the estimated expression of each variant under each model

variants `matrix` indicating the exons contained in each variant.

integralSum Sum of the log(integrated likelihood) + log(model prior probability) across all considered models.

npathDeleted Number of paths that had 0 probability under all considered variants and had to be excluded for model fitting purposes.

priorq Input parameter to `calcDenovo`

txLength Length of transcripts in bp (including new isoforms found by casper)

Methods

show `signature(object = "denovoGeneExpr")`: Displays general information about the object.

names Show names (island ids)

"[" Selects a subset of genes

"[[" Selects a single gene

posprob Accesses the posterior probabilities of each model (slot posprob)

variants Accesses the variant names and their respective exons

variants<- Replaces the value of the slot variants (can be useful for renaming variants, for instance)

Author(s)

David Rossell

See Also

`calcDenovo` to create objects from the class. `denovoExpr` to obtain expression estimates from `denovoGenomeExpr` objects.

Examples

```
showClass("denovoGeneExpr")
```

---

denovoGenomeExpr-class

Class "denovoGenomeExpr"

Description

`denovoGeneExpr` stores inferred expression for de novo splicing variants for a single gene. `denovoGenomeExpr` stores the information for several genes (typically, the whole genome).

Objects from the Class

Objects are returned by `calcDenovo`.

Slots

islands A list of `denovoGeneExpr` objects, with each element containing results for an individual gene.

Methods

show `signature(object = "denovoGenomeExpr")`: Displays general information about the object.

as.list Coerces the object to a list

"[" Selects a subset of genes

"[[" Selects a single gene

Author(s)

Camille Stephan-Otto Attolini

See Also

`procGenome` and `createDenovoGenome` to create `denovoGenomeExpr` objects.

Examples

```
showClass("denovoGeneExpr")
showClass("denovoGenomeExpr")
```

---

genePlot *Plot exon structure for each transcript of a given gene.*

Description

Plot exon structure for each transcript of a given gene. Optionally, aligned reads can be added to the plot.

Usage

```
genePlot(generanges, islandid, genomeDB, reads, exp, names.arg, xlab=, ylab=, xlim, cex=1, yaxt=n, col,
```

Arguments

generanges	Object containing the ranges with start/end of each exon.
islandid	If generanges is not specified, transcripts are obtained from island islandid from the annotated genome genomeDB.
genomeDB	Annotated genome produced with the "procGenome" function
reads	pbam object with aligned reads. This is an optional argument.
exp	ExpressionSet object with expression values, as returned by calcExp. This is an optional argument.
names.arg	Optionally, indicate the names of each transcript.
xlab	x-axis label
ylab	y-axis label
xlim	x-axis limits, defaults to start of 1st exon and end of last exon
cex	Character expansion
yaxt	The y-axis in the plot has no interpretation, hence by default it is not displayed.
col	Either single color or vector of colors to be used to draw each transcript. Defaults to rainbow colors.
...	Other arguments to be passed on to plot.

Value

A plot is produced.

Methods

signature(generanges="CompressedIRangesList", islandid="ANY", genomeDB="ANY", reads="ANY", exp="ANY
: Plots a set of transcripts. Each element in the generanges corresponds to a transcript. Each transcript should contain exon start/end positions.

signature(generanges="IRanges", islandid="ANY", genomeDB="ANY", reads="ANY", exp="ANY")
: Plots a single transcript. Each range indicates the start/end of a single exon.

signature(generanges="IRangesList", islandid="ANY", genomeDB="ANY", reads="ANY", exp="ANY")
: Plots a set of transcripts. Each element in the generanges corresponds to a transcript. Each transcript should contain exon start/end positions.

signature(generanges="GRangesList", islandid="ANY", genomeDB="ANY", reads="ANY", exp="ANY")
: Plots a set of transcripts. Each element in the generanges corresponds to a transcript. Each transcript should contain exon start/end positions.

signature(generanges="GRanges", islandid="ANY", genomeDB="ANY", reads="ANY", exp="ANY")
: Plots a set of transcripts. Each space in generanges corresponds to a transcript. Each transcript should contain exon start/end positions.

signature(generanges="missing", islandid="character", genomeDB="annotatedGenome", reads="GRanges",
: Plots all transcripts stored in genomeDB for island with identifier islandid. Individual reads are added to the plot (reads contains start/end of individual read fragments).

signature(generanges="missing", islandid="character", genomeDB="annotatedGenome", reads="missing",
: Plots all transcripts stored in genomeDB for island with identifier islandid.

signature(generanges="missing", islandid="character", genomeDB="annotatedGenome", reads="procBam",
: Plots all transcripts stored in genomeDB for island with identifier islandid. Individual reads are added to the plot (reads contains start/end of individual read fragments).

signature(generanges="missing", islandid="character", genomeDB="annotatedGenome", reads="procBam",
: Plots all transcripts stored in genomeDB for island with identifier islandid. Individual reads and estimated expression are added to the plot (reads contains start/end of individual read fragments).

Author(s)

Camille Stephan-Otto Attolini, David Rossell

Examples

```
data(hg19DB)

#Plot an IRangesList
txs <- transcripts(entrezid="27",genomeDB=hg19DB)
genePlot(txs)

#Equivalently, indicate islandid
islandid <- getIsland(entrezid="27",genomeDB=hg19DB)
genePlot(islandid=islandid, genomeDB=hg19DB)
```

`getDistrs` *Compute fragment start and fragment length distributions*

Description

Compute fragment start distributions by using reads aligned to genes with only one annotated variant. Estimate fragment length distribution using fragments aligned to long exons (>1000nt). Fragment length is defined as the distance between the start of the left-end read and the end of the right-end read.

Usage

```
getDistrs(DB, bam, pbam, islandid, verbose, nreads=4*10^6, readLength)
```

Arguments

`DB`	Annotated genome. Object of class `knownGenome` as returned by `procGenome`.
`bam`	Aligned reads, as returned by `scanBam`. It must be a list with elements 'qname', 'rname', 'pos' and 'mpos'. Ignored when argument `pbam` is specified.
`pbam`	Processed BAM object of class `procBam`, as returned by function `procBam`. Arguments `bam` and `readLength` are ignored when `pbam` is specified.
`islandid`	Island IDs of islands to be used in the read start distribution calculations (defaults to genes with only one annotated variant)
`verbose`	Set to `TRUE` to print progress information.
`nreads`	To speed up computations, only the first `nreads` are used to obtain the estimates. The default value of 4 milions usually gives highly precise estimates.
`readLength`	Read length in bp, e.g. in a paired-end experiment where 75bp are sequenced on each end one would set `readLength=75`.

Value

An object of class readDistrs with slots:

`lenDis`	Table with number of fragments with a given length
`stDis`	Cumulative distribution function (object of type closure) for relative start position

Author(s)

Camille Stephan-Otto Attolini, David Rossell

Examples

```
data(K562.r1l1)
data(hg19DB)
bam0 <- rmShortInserts(K562.r1l1, isizeMin=100)

distrs <- getDistrs(hg19DB,bam=bam0,readLength=75)

#Fragment length distribution
plot(distrs,fragLength)

#Fragment start distribution (relative to transcript length)
plot(distrs,readSt)
```

`getIsland` *getIsland returns the island id associated to a given entrez or transcript id in an annotatedGenome object. getChr indicates the chromosome for a given Entrez, transcript or island id.*

Description

annotatedGenome objects store information regarding genes and transcripts. When there's an overlap in exons between several genes, these genes are grouped into gene islands. getIsland retrieves the island to which each gene or transcript was assigned, while getChr indicates the chromosome.

Usage

```
getIsland(entrezid, txid, genomeDB)
getChr(entrezid, txid, islandid, genomeDB)
```

Arguments

`entrezid`	Character indicating single Entrez identifier. Can be left missing and specify another identifier instead.
`txid`	Character indicating a single RefSeq transcript identifier. Can be left missing and specify another identifier instead.
`islandid`	Character indicating the gene island indentifier. Can be left missing and specify another identifier instead.
`genomeDB`	Object of class `annotatedGenome`

Value

Character with island identifier

Methods

`signature(entrezid='character',txid='missing',genomeDB='annotatedGenome')` Return island id for given Entrez identifier

`signature(entrezid='missing',txid='character',genomeDB='annotatedGenome')` Return island id for given transcript identifier (RefSeq)

`signature(entrezid='character',txid='missing',islandid='missing',genomeDB='annotatedGenome')` Return chromosome for given Entrez identifier (RefSeq)

`signature(entrezid='missing',txid='character',islandid='missing',genomeDB='annotatedGenome')` Return chromosome for given transcript identifier (RefSeq)

`signature(entrezid='missing',txid='missing',islandid='character',genomeDB='annotatedGenome')` Return chromosome for given island identifier

`signature(entrezid='character',txid='missing',islandid='missing')` Return chromosome for given Entrez identifier

`signature(entrezid='missing',txid='character',islandid='missing')` Return chromosome for given transcript identifier (RefSeq)

`signature(entrezid='missing',txid='character',islandid='missing')` Return chromosome for given island identifier

Examples

```
data(hg19DB)
getIsland(entrezid="27",genomeDB=hg19DB)
getIsland(txid="NM_005158",genomeDB=hg19DB)

getChr(entrezid="27",genomeDB=hg19DB)
getChr(txid="NM_005158",genomeDB=hg19DB)
```

---

getNreads *Get total number of paths in each island from a pathCounts object.*

Description

getNreads returns a numeric vector with the total number of path counts in each island from a pathCounts object.

Usage

```
getNreads(pc)
```

Arguments

pc	pathCounts object generated by pathCounts()

Value

Numeric vector with total number of path counts in each island of pc.

Methods

signature(pathCounts='pathCounts') Returns numeric vector with total number of path counts for each island in the pathCounts object.

Author(s)

Camille Stephan-Otto Attolini

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.
```

---

getReads *getReads returns the reads stored in a* procBam *object.*

Description

procBam objects store reads that have been split according to their CIGAR codes. getReads accesses these reads.

Usage

```
getReads(x)
```

Arguments

x	Object of class procBam

Value

RangedData object with reads stored in x.

Methods

signature(x='procBam') Return reads stored in x.

Examples

```
#See example in calcExp
```

getRoc *Operating characteristics of differential expression analysis*

Description

getRoc compares simulation truth and data analysis results to determine False Positives (FP), False Negatives (FP), True Positives (TP), True Negatives (TN), Positives (FP+TP), False Discovery Proportion (FP/P) and Power (TP/(TP+FN)).

Usage

```
getRoc(simTruth, decision)
```

Arguments

simTruth	Binary vector or matrix indicating simulation truth (FALSE or 0 for non differential expression, TRUE or 1 for differential expression)
decision	Binary vector or matrix with differential expression calls based on some data analysis.

Value

data.frame with TP, FP, TN, FN, P, FDR and Power.

Methods

signature(simTruth='logical',decision='logical') Operating characteristics are computed for a single simulation

signature(simTruth='numeric',decision='numeric') Operating characteristics are computed for a single simulation

signature(simTruth='matrix',decision='matrix') simTruth and decision contain truth and calls for several simulations (in columns). getRoc returns a data.frame with operating characteristics in each simulation.

Author(s)

David Rossell

Examples

```
## See help(probNonEquiv) for an example
```

hg19DB *Subset of human genome (UCSC hg19 version)*

Description

We downloaded the human genome hg19 via procGenome and selected a few genes from chromosome 1 to use as a toy data for the vignette and examples.

Usage

```
data(hg19DB)
```

Format

An annotatedGenome object. See help(procGenome) and help(annotatedGenome-class) for details.

Examples

```
data(hg19DB)
hg19DB
slotNames(hg19DB)
```

K562.r1l1 *Toy RNA-seq data from RGASP project.*

Description

The paired-end RNA-seq data is from the RGASP project sample K562_2x75 (replicate 1, lane 1) and was obtained at ftp://ftp.sanger.ac.uk/pub/gencode/rgasp/RGASP1/inputdata/human_fastq. Reads were aligned against hg19 with tophat 2.0.2 and bowtie 0.12.5, setting the insert size at -r 200, and imported into R using scanBam from package Rsamtools. For illustration purposes, we selected reads mapping to a few genes only (namely, the genes that were also selected for the toy genome annotation in data(hg19DB).

Usage

```
data(K562.r1l1)
```

Format

A list indicating read id, chromosome, start and end locations and the position of the pair, as returned by scanBam.

Source

ftp://ftp.sanger.ac.uk/pub/gencode/rgasp/RGASP1/inputdata/human_fastq

References

C Trapnell, L Pachter, SL Salzberg. TopHat: discovering splice junctions with RNA-Seq. Bioinformatics, 2009, 25, 1105-1111. doi=10.1093/bioinformatics/btp120.

B Langmead, C Trapnell, M Pop, SL Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biology, 2009, 10:R25.

Examples

```
data(K562.r1l1)
names(K562.r1l1)
```

`mergeBatches` *Merge two ExpressionSet objects by doing quantile normalization and computing partial residuals (i.e. substracting group mean expression in each batch). As currently implemented the method is only valid for balanced designs, e.g. each batch has the same number of samples per group.*

Description

`mergeBatches` combines `x` and `y` into an `ExpressionSet`, performs quantile normalization and adjusts for batch effects by subtracting the mean expression in each batch (and then adding the grand mean so that the mean expression per gene is unaltered).

Usage

```
mergeBatches(x, y, mc.cores=1)
```

Arguments

`x`	`ExpressionSet` object with data from batch 1.
`y`	Either `ExpressionSet` object with data from batch 2, or `simulatedSamples` object with data from multiple simulations.
`mc.cores`	Number of processors to be used (ignored when `y` is an `ExpressionSet`)

Value

When `y` is an `ExpressionSet`, `mergeBatches` returns an `ExpressionSet` with combined expressions. Its `featureData` contains a variable `"batch"` indicating the batch that each sample corresponded to.

When `y` is a `simulatedSamples` object, `mergeBatches` is applied to combine `x` with each dataset in `y` and a list of `ExpressionSet` objects is returned.

Author(s)

David Rossell

Examples

```
#Fake data from 2 batches
x <- matrix(rnorm(6),nrow=2)
colnames(x) <- paste(x,1:3,sep=)
y <- matrix(1+rnorm(6),nrow=2)
colnames(y) <- paste(y,1:3,sep=)
x <- new("ExpressionSet",exprs=x)
y <- new("ExpressionSet",exprs=y)
exprs(x)
exprs(y)

#Merge & adjust
z <- mergeBatches(x,y)
exprs(z)
```

---

mergeExp *Merge splicing variant expression from multiple samples*

Description

mergeExp combines the output of calcExp from multiple samples, i.e. multiple ExpressionSet objects, into a single ExpressionSet

Usage

```
mergeExp(..., sampleNames,  keep=c(transcript,gene))
```

Arguments

...	ExpressionSet objects to be combined.
sampleNames	Character vector indicating the name of each sample. Defaults to 'Sample1', 'Sample2', etc.
keep	Variables in the featureData of each individual ExpressionSet to keep in the merged output.

Details

mergeExp runs some checks to ensure that object can be combined (e.g. making sure that measurements are obtained on same set of genes), then sorts and formats each input ExpressionSet.

A label with the sample name is appended to variables in the featureData that appear in multiple samples, e.g. variable 'se' reporting standard errors (obtained by setting citype='asymp' in calcExp).

Value

Object of class ExpressionSet combining the input ExpressionSets. Its featureData contains the columns indicated in the keep argument, plus a column readCount with the total number of reads mapped to each gene (or gene island, when multiple genes have overlapping exons).

Author(s)

David Rossell

See Also

`calcExp` to obtain an `ExpressionSet` for an individual sample.

Examples

```
#See example in calcExp
```

`modelPrior` *Set prior distribution on expressed splicing variants.*

Description

Set prior on expressed splicing variants using the genome annotation contained in a `knownGenome` object.

The prior probability of variants V1,...,Vn being expressed depends on n, on the number of exons in each variant V1,...,Vn and the number of exons in the gene. See the details section.

Usage

```
modelPrior(genomeDB, maxExons=40, smooth=TRUE, verbose=TRUE)
```

Arguments

`genomeDB`	Object of class `knownGenome`
`maxExons`	The prior distribution is estimated for genes with 1 up to `maxExons` exons. As there are fewer genes with many exons, the prior parameters are estimated poorly. To avoid this common estimate is used for all genes with more than `maxExons` exons
`smooth`	If set to `TRUE` the estimated prior distribution parameters for the number of exons in a gene are smoothed using Generalized Additive Models. This step typically improves the precision of the estimates, and is only applied to genes with 10 or more exons.
`verbose`	Set to `TRUE` to print progress information.

Details

The goal is to set a prior that takes into account the number of annotated variants for genes with E exons, as well as the number of exons in each variant.

Suppose we have a gene with E exons. Let V_1,...,V_n be n variants of interest and let |V_1|,...,|V_n| be the corresponding number of exons in each variant. The prior probability of variants V_1,...,V_n being expressed is modeled as

P(V_1,...,V_n|E)= P(n|E) P(|V_1| |E) ... P(|V_n| |E)

where P(n|E)= NegBinom(n; k_E, r_E) I(0 < n < 2^E) and P(|V_i| |E)= BetaBinomial(|V_i|-1; E-1, alpha_E, beta_E).

The parameters k_E, r_E, alpha_E, beta_E depend on E (the number of exons in the gene) and are estimated from the available annotation via maximum likelihood. Parameters are estimated jointly for all genes with E>= `maxExons` in order to improve the precision.

For `smooth==TRUE`, alpha_E and beta_E are modeled as a smooth function of E by calling `gam` and setting the smoothing parameter via cross-validation. Estimates for genes with E>=10 are substituted by their smooth versions, which typically helps improve stability in the estimates.

Value

List with 2 components.

`nvarPrior`	List with prior distribution on the number of expressed variants for genes with 1,2,3... exons. Each element contains the truncated Negative Binomial parameters, observed and predicted frequencies (counting the number of genes with a given number of variants).
`nexonPrior`	List with prior distribution on the number of exons in a variant for genes with 1,2,3... exons. Each element contains the Beta-Binomial parameters, observed and predicted frequencies (counting the number of variants with a given number of exons)

Author(s)

David Rossell, Camille Stephan-Otto Attolini

Examples

```
data(hg19DB)
mprior <- modelPrior(hg19DB, maxExons=10)

##Prior on number of expressed variants
##Genes with 2 exons
##mprior$nvarPrior[[2]]
##Genes with 3 exons
##mprior$nvarPrior[[3]]

##Prior on the number of exons in an expressed variant
##Genes with 2 exons
##mprior$nexonPrior[[2]]
##Genes with 3 exons
##mprior$nexonPrior[[3]]
```

`modelPriorAS-class` *Class "modelPriorAS"*

Description

`modelPriorAS` stores parameters for the prior distribution on all possible alternative splicing configuration (i.e. prior on model space). This information is used for de novo reconstruction of splicing variants.

Objects from the Class

Objects are created by function `modelPrior`.

Slots

nvarPrior Prior on the number of variants per gene. A list with components `"nbpar"` containing the parameters of the Negative Binomial distribution, `"obs"` containing the observed counts and `"pred"` the Negative Binomial predicted counts.

nexonPrior Prior on the number of exons in an expressed variant. A list with components `"bbpar"` containing Beta-Binomial parameters, `"obs"` containing the observed counts and `"pred"` the Beta-Binomial predicted counts.

Methods

show `signature(object = "modelPriorAS")`: Displays general information about the object.

"[" Selects prior parameters for genes with the specified number of exons

coef Selects a single gene

Author(s)

David Rossell

See Also

`procGenome` and `createDenovoGenome` to create `modelPriorAS` objects.

Examples

```
showClass("modelPriorAS")
```

pathCounts *Compute exon path counts*

Description

Compute counts for exon paths visited by aligned reads

Usage

```
pathCounts(reads, DB, mc.cores = 1, verbose=FALSE)
```

Arguments

reads	Object of class procBam containing aligned reads, as returned by procBam.
DB	Object of class annotatedGenome containing either a known or de novo annotated genome.
mc.cores	Number of processors to be used for parallel computing. Requires having package multicore installed and loaded.
verbose	Set to TRUE to print progress information.

Value

Named integer vector with counts of exon paths. Names are character strings built as ".exon1.exon2-exon3.exon4.", with dashes making the split between exons visited by left and right-end reads correspondingly.

Methods

`signature(reads='list')` Computes counts for exon paths from a list of procBam objects (usually reads processed and split by chromosome).

`signature(reads='procBam')` Compute counts for exon paths from a procBam object of processed reads.

Author(s)

Camille Stephan-Otto Attolini

See Also

procGenome to create an annotated genome object, createDenovoGenome to create a de novo annotated genome. See help(getNreads) to get number of fragments mapping to each island.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.
```

pathCounts-class *Class "pathCounts"*

Description

Stores exon path counts.

Objects from the Class

Objects are created with a call to `pathCounts`.

Slots

counts List with one element per gene island. For each island, it contains a named vector with exon path counts. The names indicate the visited exons.

For instance, consider that for gene '1' with 2 exons we observe 10 reads in which the left end falls completely in exon 1 and the right end in exon 2. Suppose that for 5 reads the left end bridges exons 1-2 and the right end falls in exon 2. Then `pc[[1]]` would contain `c(10,5)` and `names(pc[[1]])` would contain `c(".1-2.",".1.2-2.")`

denovo Logical variable. `FALSE` indicates that the counts correspond to a known genome (i.e. created with `procGenome`), and `TRUE` to a de novo annotated genome (i.e. created with `createDenovoGenome`).

stranded Logical variable. `TRUE` indicates that the path counts were obtained from and RNA-seq experiment where strand information was preserved.

Methods

show `signature(object = "pathCounts")`: Displays general information about the object.

Author(s)

Camille Stephan-Otto Attolini

Examples

```
showClass("pathCounts")
```

plot *Plot estimated read start and fragment length distributions.*

Description

Plots the estimated fragment length (insert size) distribution and the relative read start distribution (0 indicating transcription start, 1 transcription end). The former checks that the insert size distribution matches that described in the experimental protocol. The latter checks the extent to which reads are non-uniformly distributed (note: casper does NOT assume reads to be uniformly distributed, so a lack of uniformity is not a problem per se).

Usage

```
plot(x, y, ...)

lines(x, ...)
```

Arguments

x	Object of type readDistrs, as returned by getDistrs.
y	Set to "fragLength" to plot the estimated insert size ditribution. Set to "readSt" to plot a histogram of the estimated read start distribution.
...	Further arguments to be passed on to plot.

Methods

signature(x = "readDistrs") x is an object of type readDistrs, as returned by getDistrs. The plot function allows to visualize the fragment length and read start distributions.

Examples

```
#See getDistrs examples
```

plotExpr *Plot inferred gene structure and expression.*

Description

Plots variants with sufficiently large posterior probability of being expressed along with their (marginal) estimated expression.

Usage

```
plotExpr(gene, minProbExpr = 0.5, minExpr = 0.1,
    xlab = "(kb)", ylab = "", xlim, cex = 1, yaxt = "n", col, ...)
```

Arguments

gene	denovoGeneExpr object containing results for a single gene, as returned by calcDenovo.
minProbExpr	Variants with marginal posterior probability of expression below minProbExpr are not reported
minExpr	Variants with (marginal) estimated expression below minExpr are not reported. Can be useful to remove sequence preference artifacts.
xlab	x-axis label, passed on to plot
ylab	y-axis label, passed on to plot
xlim	x-axis limits, passed on to plot
cex	Character expansion, passed on to plot
yaxt	Type of y-axis, passed on to plot
col	Colors for each variant, defaults to rainbow colors. It is possible to specify a single color.
...	Other arguments to be passed on to plot

Details

The marginal posterior probability that a variant is expressed is the sum of the posterior probabilities of all models containing that variant.

The marginal estimated expression is the average expression across all models (including those where the variant has 0 expression) weighted by the posterior probability of each model.

Methods

signature(gene = "denovoGeneExpr") gene contains the results from a de novo isoform expression analysis for a single gene, as returned by calcDenovo. When calcDenovo is run on multiple genes simultaneously, the desired gene can be selected using the "[[" operator as usual.

Examples

```
#See calcDenovo examples
```

---

plotPriorAS *Plot prior distribution on set of expressed variants (i.e. the model space).*

Description

Plots the prior distribution on the number of expressed variants and the number of exons per variant in genes with exons exons (as returned by function modelPrior). The prior distribution is compared to the observed frequencies to check that the assumed distributional forms are reasonable.

Usage

```
plotPriorAS(object, type = "nbVariants", exons = 1:9, xlab, ylab = "Probability", col = c("red", "blue")
```

Arguments

object	modelPriorAS object with prior distribution on model space.
type	Set to "nbVariants" to plot the prior on the number of variants per gene. Set to "nbExons" to plot the prior on the number of exons.
exons	Vector with integers. The plot is only produced with number of exons indicated in exons.
xlab	x-axis label, passed on to plot
ylab	y-axis label, passed on to plot
col	Colors for bars showing prior probabilities and frequencies in the known genome

Methods

signature(object = "modelPriorAS") object contains the prior distribution on the model space, as returned by function modelPrior

Examples

```
#See modelPrior examples
```

---

probNonEquiv *Posterior probability that differences between group means are greater than a given threshold, i.e. that groups are non-equivalent by a biologically meaningful amount.*

Description

The function computes v_i=P(|theta_i| > logfc | data), where theta_i is the difference between group means for gene i. This posterior probability is based on the NNGCV model from package EBarrays, which has a formulation similar to limma in an empirical Bayes framework. Notice that the null hypothesis here is that |theta_i|<logfc, e.g. isoforms with small fold changes are regarded as uninteresting.

Subsequent differential expression calls are based on selecting large v_i. For instance, selecting v_i >= 0.95 guarantees that the posterior expected false discovery proportion (a Bayesian FDR analog) is below 0.05.

Usage

```
probNonEquiv(x, groups, logfc = log(2), minCount, method = "exact")
```

Arguments

x	ExpressionSet containing expression levels
groups	Variable in fData(x) indicating the two groups to compare (the case with more than 2 groups is not implemented).
logfc	Biologically relevant threshold for the log fold change, i.e. difference between groups means in log-scale
minCount	If specified, probabilities are only computed for rows with `fData(x)$readCount >= minCount`
method	Set to `exact` for exact posterior probabilities (slower), `plugin` for plug-in approximation (much faster). The latter is less conservative than the former, but typically controls the FDR below alpha when setting a threshold 1-alpha on the resulting posterior probabilities.

Value

Vector with posterior probabilities (NA's are returned for rows with less than `minCount` reads).

Author(s)

Victor Pena, David Rossell

References

Rossell D, Stephan-Otto Attolini C, Kroiss M, Stocker A. Quantifying Alternative Splicing from Paired-End RNA-sequencing data. COBRA (http://biostats.bepress.com/cobra/art97)

Examples

```
#Simulate toy data
p <- 50; n <- 10
x <- matrix(rnorm(p*2*n),nrow=p)
x[(p-10):p,1:n] <- x[(p-10):p,1:n] + 1.5
x <- new("ExpressionSet",exprs=x)
x$group <- rep(c(group1,group2),each=n)

#Posterior probabilities
pp <- probNonEquiv(x, groups=group, logfc=0.5)
d <- rowMeans(exprs(x[,1:n])) - rowMeans(exprs(x[,-1:-n]))
plot(d,pp,xlab=Observed log-FC)
abline(v=c(-.5,.5))

#Check false positives
truth <- rep(c(FALSE,TRUE),c(p-11,11))
getRoc(truth, pp>.9)
getRoc(truth, pp>.5)
```

procBam *Process BAM object*

Description

Process paired-end data stored in BAM object generated by scanBam. Outputs GRanges objects for reads and junctions.

Usage

```
   procBam(bam, stranded, seed, verbose=FALSE, rname, keep.junx=FALSE,
keep.flag=FALSE, ...)
```

Arguments

bam	BAM object generated by scanBam
stranded	Set to TRUE to indicate that the RNA-seq experiment preserved the strand information.
seed	Seed for random number generator
verbose	Set to TRUE to print progress information.
rname	Chromosome to process be combined with the \'which\' argument in the scanBam function)
keep.junx	Option to store junction information. Only useful for finding denovo exons and transcripts.
keep.flag	Option to store aligment flag information.
...	Other arguments to be passed on to procBam.

Details

In case of multihits with same start position for both reads but different insertions/deletions patterns only one alignment is chosen at random.

Value

An object of class procBam containing reads with both ends correctly aligned and split according to the corresponding CIGAR. Unique identifiers by fragment are stored. Junctions spanned by reads are also stored in GRanges object if the argument \'keep.junx\' is set to TRUE.

Methods

signature(bam='list',stranded='logical',seed='integer',verbose='logical', rname='character',keep.ju
: Process paired-end data stored in BAM object generated by scanBam. Outputs GRanges objects for reads and (optionally) junctions.

Author(s)

Camille Stephan-Otto Attolini

See Also

scanBam from package Rsamtools, help("procBam-class"), getReads.

Examples

```
##See example in calcExp
```

---

procBam-class *Class "procBam"*

Description

Stores processed bam files in a RangedData format. Each read is split into disjoint ranges according to its cigar code.

Objects from the Class

Objects are created with a call to procBam.

Slots

pbam GRanges indicating chromosome, start and end of each disjoint range. The pair id and read id within the pair are also stored.

junx GRanges indicating chromosome, start and end of junctions spanned by reads.

stranded Logical variable. TRUE indicates that the reads were obtained from and RNA-seq experiment where strand information was preserved.

In the case of stranded experiments:

plus GRanges indicating chromosome, start and end of each disjoint range for fragments originated from the positive strand. The pair id and read id within the pair are also stored.

minus GRanges indicating chromosome, start and end of each disjoint range for fragments originated from the negative strand. The pair id and read id within the pair are also stored.

pjunx GRanges indicating chromosome, start and end of junctions spanned by reads originated from the positive strand.

mjunx GRanges indicating chromosome, start and end of junctions spanned by reads originated from the negative strand.

Methods

show signature(object = "procBam"): Displays general information about the object.

getReads signature(x = "procBam"): Extracts the aligned reads stored in x.

Author(s)

Camille Stephan-Otto Attolini, David Rossell

See Also

getReads

Examples

```
showClass("procBam")
```

---

procGenome *Create an annotatedGenome object that stores information about genes and transcripts*

Description

procGenome processes annotations for a given transcriptome, either from a TranscriptDb object created by GenomicFeatures package (e.g. from UCSC) or from a user-provided GRanges object (e.g. by importing a gtf file output by Cufflinks RABT module with import from rtracklayer package).

createDenovoGenome creates a de novo annotated genome by combining UCSC annotations and observed RNA-seq data.

Usage

```
procGenome(genDB, genome, mc.cores = 1)

createDenovoGenome(reads, DB, minLinks=2,
maxLinkDist=1e+05, maxDist=1000, minConn=2, minJunx=3, minLen=12, mc.cores=1)
```

Arguments

genDB	Either annotations downloaded from UCSC (e.g. genDB<-makeTranscriptDbFromUCSC(genome="hg19 tablename="refGene")) or GRanges object (e.g. genDB <- import('transcripts.gtf')). See details.
genome	Character indicating genome version (e.g. "hg19", "dm3")
mc.cores	Number of cores to use in parallel processing (multicore package required)
DB	annotatedGenome object, as returned by procGenome
minLinks	Minimum number of reads joining two exons to merge their corresponding genes
maxLinkDist	Maximum distance between two exons to merge their correspondin genes. A value of 0 disables this option.
maxDist	Maximum distance between two exons with reads joining them to merge their corresponding genes.

minConn	Minimum number of fragments connecting a new exon to an annotated one to add to denovo genome.
minJunx	Minimum number of junctions needed to redefine an annotated exon's end or start.
minLen	Minimum length of a junction to consider as a putative intron.
reads	Processed reads stored in a RangedData, as returned by procBam

Details

These functions create the annotation objects that are needed for subsequent functions. Typically these objects are created only once for a set of samples.

If interested in quantifying expression for known transcripts only, one would typically use procGenome with a TranscriptDb from the usual Bioconductor annotations. It is also possible to provide procGenome with any desired set of transcripts, so that one can use other genome annotations or even de novo transcripts found by some third-party software. These annotations should be provided as a GRanges object. For instance, one could use the transcripts found by the Cufflinks RABT module. These transcripts are saved in gtf files, which can be imported into Bioconductor as GRanges with function import (package rtracklayer). procGenome is intended to be used in combination with calcExp, which quantifies expression for a set of given transcripts.

createDenovoGenome performs a de novo transcript discovery following our own strategy, and is intended to be used in combination with calcDenovo, which performs a model search additional to quantifying expression. It is convenient to create a single genome by combining information from all samples, as this increases the power to detect new exons and guarantees that the same exons and gene islands are used across samples.

Value

Object of class annotatedGenome.

Methods

signature(genDB = "transcriptDb") genDB is usually obtained with a call to makeTranscriptDbFromUCSC (package GenomicFeatures), e.g. genDB<-makeTranscriptDbFromUCSC(genome="hg19", tablename="refGene")

signature(genDB = "GRanges") genDB stores information about all transcripts and their respective exons. Chromosome, start, end and strand are stored as usual in GRanges objects. genDB must have a column named "type" taking the value "transcript" for rows corresponding to transcript and "exon" for rows corresponding to exons. It must also store transcript and gene ids. For instance, Cufflinks RABT module creates a gtf file with information formatted in this manner for known and de novo predicted isoforms.

Author(s)

Camille Stephan-Otto Attolini

See Also

See annotatedGenome-class for a description of the class. To obtain the island id for an entrez gene id or a transcript id see help(getIsland). To extract the transcripts for a entrez or island id see help(transcripts). See splitGenomeByLength for splitting an annotatedGenome according to gene length.

Examples

```
##Known transcripts from UCSC
## genDB<-makeTranscriptDbFromUCSC(genome="hg19", tablename="refGene")
## hg19DB <- procGenome(genDB, "hg19")

##Transcripts inferred by Cufflinks-RABT module
##genDB.Cuff <- import(transcripts.gtf)
##hg19DB.Cuff <- procGenome(genDB.Cuff, genome=hg19)
```

---

rmShortInserts *Remove reads with short insert sizes from imported BAM files.*

Description

In paired-end experiments short inserts (i.e. the 2 ends being very close to each other), may indicate RNA degradation or that a short RNA (e.g. miRNA) is being sequenced. Typically the goal is not to study alternative splicing for such short/degraded RNA; in this case it is recommendable to remove such short inserts to avoid biasing the insert size distribution. Requiring a minimum insert size can also result in significantly faster computations when quantifying alternative splicing via calc or calcDenovo.

Usage

```
rmShortInserts(bam, isizeMin=100)
```

Arguments

bam	Object with aligned reads, as returned by scanBam
isizeMin	Reads with insert size smaller than isizeMin will be removed.

Value

Named list, in the same format as that returned by scanBam.

Note

The insert size is stored in objects imported with scanBam in the element named isize.

Author(s)

David Rossell

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.
```

`simMultSamples` *Simulate paired end reads for multiple future samples based on pilot data, and obtain their expression estimates via casper*

Description

Simulate true mean expression levels for each group and future individual samples within each group, and simulate future observed data (in the form of Casper expression estimates).

These simulations serve as the basis for sample size calculation: if one were to sequence `nsamples` new RNA-seq samples, what data would we expect to see? The simulation is posterior predictive, i.e. based on the current available data `x`.

Usage

```
simMultSamples(B, nsamples, nreads, readLength, x, groups=group, distrs, genomeDB, verbose=TRUE, mc.co
```

Arguments

`B`	Number of simulations to obtain
`nsamples`	Vector indicating number of future samples per group, e.g. `nsamples=c(5,5)` to simulate 5 new samples for 2 groups.
`nreads`	number of RNA-seq reads per sample. `simMultSamples` assumes that exactly these many reads will be aligned to the genome.
`readLength`	Read length, i.e. in an experiment with paired reads at 100bp each, `readLength=100`.
`x`	`ExpressionSet` containing pilot data. `x[[group]]` indicates groups to be compared
`groups`	Name of column in `pData(x)` indicating the groups
`distrs`	Fragment start and length distributions. It can be either an object of class readDistrs, or a list where each element is of class readDistrs. In the latter case, an element is chosen at random for each individual sample (so that uncertainty in these distributions is taken into account)
`genomeDB`	annotatedGenome object
`verbose`	Set to `TRUE` to print progress
`mc.cores`	Number of cores to use in function. `mc.cores>1` requires package `parallel`

Details

The posterior predictive simulations is based on four steps: (1) simulate true expression for each group (mean and SD), (2) simulate true expression for future samples, (3) simulate paired reads for each future sample, (4) estimate expression from the reads via Casper. Below are some more details.

1. Simulate true mean expression in each group and residual variance for each gene. This is based on the NNGCV model in package EBarrays, adapted to take into account that the expression estimates in the pilot data `x` are noisy (which is why `simMultSamples` requires the SE / posterior SD associated to `exprs(x)`). The simulated values are returned in component `"simTruth"` of the `simMultSamples` output.

2. Simulate true isoform expression for each of the future samples. These are independent Normal draws with mean and variance generated in step 1. True gene expression is derived from the isoform expressions.

3. Determine the number of reads to be simulated for each gene based on its true expression (generated in step 2) and a Multinomial sampling model. Simulate aligned reads in the form of path counts. The number of simulated reads is reported in component `"simExpr"` of the `simMultSamples` output.

4. Obtain expression estimates from the path counts produced in step 3 via `calcExp`. These are reported in component `"simExpr"` of the `simMultSamples` output.

Value

Object of class `simulatedSamples`, which extends a list of length `B`. See the class documentation for some helpful methods (e.g. coef, exprs, mergeBatches). Each element is itself a list containing an individual simulation.

`simTruth`	`data.frame` indicating the mean and standard deviation of the Normal distribution used to generate data from each group
`simExpr`	`ExpressionSet` with Casper expression estimates, as returned by `calcExp`. `pData(simExpr)` indicates group information, and `fData(simExpr)` the number of simulated reads for each sample (in columns `explCnts`) and across all samples (in column `readCount`)

Author(s)

Victor Pena

`simReads` *Function to simulate paired end reads following given read start and fragment length distributions and gene and variant expressions.*

Description

This function generates path counts and bam files with simulated paired end reads according to given read start distribution, fragment length distribution and gene and variant expressions.

Usage

```
simReads(islandid, nSimReads, pis, rl, seed, writeBam, distrs, genomeDB,
repSims=FALSE, bamFile=NULL, stranded=FALSE, verbose=TRUE, chr=NULL, mc.cores=1)
```

Arguments

islandid	Island ID's from the genomeDB object to simulate reads
nSimReads	Named numeric vector with number of fragments to simulate in each island.
pis	Named numeric vector with relative expression of transcripts. Expressions add up to one for each island to simulate.
rl	Read length
seed	Seed of the random numbers generator
writeBam	Set to 1 to generate bam files with the simulated reads
distrs	Object of class 'readDistrs' with read start and fragment length distributions
genomeDB	Object of class 'annotatedGenome' with the genome to genererate reads from
repSims	Set to TRUE to return relative read starts and fragment lengths from the simulation
bamFile	Name of the bam file to write reads to. Must end with '.bam'
stranded	Set to TRUE to preserve gene strand when generating reads. The 'XS' tag will be added to reads in the bam file and the returned 'pc' object will be stranded
verbose	Set to TRUE to print progress
chr	Characters vector with chromosomes to simulate. Defaults to whole genome simulations.
mc.cores	Number of cores to use in function

Value

Nsim	Numerical vector with the number of reads simulated for each island.
pc	Object of class 'pathCounts' with simulated path counts
sims	Only if 'repSims' is set to TRUE. List with vectors of length 'n' with the following elements: -'varl': Length of variant for corresponding read -'st' Start of fragment relative to variant start (not in genomic coordinates) -len:Fragment length -'strand':Strand of gene for simulated read

Author(s)

Camille Stephan-Otto Attolini

Examples

```
data(hg19DB)
data(K562.r1l1)
distrs <- getDistrs(hg19DB,bam=K562.r1l1,readLength=75)

islandid <- c(10319,463)
```

```
txs <- unlist(lapply(hg19DB@transcripts[islandid], names))
pis <- vector(mode=numeric, length=length(txs))
npis <- sapply(hg19DB@transcripts[islandid],length)
pis[1:npis[1]] <- rep(1/npis[1],npis[1])
pis[-1:-npis[1]] <- rep(1/npis[2],npis[2])
names(pis) <- txs
nSimReads <- c(100, 100)
names(nSimReads) <- islandid

simpc <- simReads(islandid=islandid, nSimReads=nSimReads, pis=pis,
rl=75, repSims=TRUE, seed=1, writeBam=FALSE, distrs=distrs,genomeDB=hg19DB)
```

`simulatedSamples-class` *Class "simulatedSamples"*

Description

`simulatedSamples` stores multiple simulated isoform expression datasets. Each dataset contains the (simulation) true mean expression in each group and residual variance, as well as the estimated expression in each individual sample.

Objects from the Class

Objects are returned by `simMultSamples`.

Slots

The class extends a list directly.

.Data A list, each element containing a different simulated dataset

Methods

show `signature(object = "simulatedSamples")`: Displays general information about the object.

coef `signature(object = "simulatedSamples")`: Returns a matrix with difference between group means (simulation truth) in all simulated datasets

exprs `signature(object = "simulatedSamples")`: Returns a list of ExpressionSets containing the estimated expressions in each simulation.

mergeBatches `signature(x="ExpressionSet",y="simulatedSamples")`: Combines `x` with each element in `exprs` in `y`, and returns a list. See `help(mergeBatches)` for more details.

Author(s)

David Rossell

See Also

mergeBatches

Examples

```
showClass("simulatedSamples")
```

---

splitGenomeByLength *Split an annotatedGenome object into subsets according to gene length*

Description

splitGenomeByLength splits an annotatedGenome according to gene length (bp), which allows estimating the fragment start and length distribution for each subset separately.

Usage

```
splitGenomeByLength(DB, breaks=c(0,3000,5000,Inf))
```

Arguments

DB	Object containing annotated genome. Must be of class annotatedGenome, as returned by procGenome or createDenovoGenome.
breaks	Breakpoints to define gene subgroups.

Details

By default groups are <3000bp, 3000-5000bp, >5000bp, which work well for the human genome. Further sub-dividisions may result in unstable estimates of fragment start and length distributions.

Value

List where each component is of class annotatedGenome.

Author(s)

David Rossell

See Also

procGenome and createDenovoGenome for creating annotatedGenome objects. getDistrs for estimating fragment start and length distribution.

Examples

```
##Not run
## genDB<-makeTranscriptDbFromUCSC(genome="hg19", tablename="refGene")
## hg19DB <- procGenome(genDB, "hg19")
## hg19split <- splitGenomeByLength(hg19DB)
```

`subsetGenome` *subsetGenome subsets an object of class annotatedGenome for a set of island IDs or chromosome names.*

Description

~~ Methods for function `subsetGenome` in package **casper** ~~ Subset an annotatedGenome object by islands or chromosomes.

Usage

```
subsetGenome(islands, chr, genomeDB)
```

Arguments

`islands`	Vector of characters with the island IDs to retrieve from genome.
`chr`	Vector of characters with the names of chromosomes to retrieve from genome.
`genomeDB`	annotatedGenome object with genome to subset.

Methods

`signature(islands = "character", chr = "missing", genomeDB = "annotatedGenome")`
: Subset annotatedGenome object by a set of island IDs.

`signature(islands = "missing", chr = "character", genomeDB = "annotatedGenome")`
: Subset annotatedGenome object by chromosomes.

`transcripts` *Returns all transcripts associated to a given island or entrez id stored in an annotatedGenome object.*

Description

annotatedGenome objects store information regarding genes and transcripts. When there's an overlap in exons between several genes, these genes are grouped into gene islands.

The function transcripts retrieves all stored transcripts for a given gene or island.

Usage

```
transcripts(entrezid, islandid, genomeDB)
```

Arguments

entrezid	Character indicating single Entrez identifier. Can be left missing and specify islandid instead.
islandid	Character indicating island identifier. Can be left missing and specify entrezid instead
genomeDB	Object of class annotatedGenome

Value

IRangesList where each element in the list corresponds to a different transcript.

Methods

signature(x = "readDistrs") x is an object of type readDistrs, as returned by getDistrs. The plot function allows to visualize the fragment length and read start distributions.

See Also

genePlot to plot the resulting transcripts

Examples

```
data(hg19DB)
tx <- transcripts(entrezid="27",genomeDB=hg19DB)
tx
```

---

txLength ~~ *Methods for Function* txLength *in Package* **casper** ~~

Description

~~ Methods for function txLength in package **casper** ~~ Function to retrieve transcript lengths from annotated genome (class genomeDB).

Usage

```
txLength(islandid, txid, genomeDB)
```

Arguments

islandid	Retrieve length for transcripts in island islandid.
txid	Retrieve length for txid transcripts.
genomeDB	Annotated genome of class genomeDB.

Details

When called for the first time lengths are calculated and stored in the object genomeDB. Subsequent calls refer to these computed values.

Value

Named numeric vector with transcript lengths.

Methods

`signature(islandid = "character", txid = "missing", genomeDB = "annotatedGenome")`
Retrieve lengths from genomeDB for transcripts in `islandid` islands.

`signature(islandid = "missing", txid = "character", genomeDB = "annotatedGenome")`
Retrieve lengths from genomeDB for `txid` transcripts.

`signature(islandid = "missing", txid = "missing", genomeDB = "annotatedGenome")`
Retrieve or calculate lengths for all transcripts in the annotated genome genomeDB.

---

wrapKnown *Run all necessary steps to analyze a bam file with the casper pipeline.*

Description

Function to analyze a bam file to generate expression estimates, read start and fragment length distributions, path counts and processed reads.

Usage

```
wrapKnown(bamFile, verbose=FALSE, seed=1, mc.cores.int=1,
mc.cores=1, genomeDB, readLength, rpkm=TRUE, priorq=2, priorqGeneExpr=2,
citype=none, niter=10^3, burnin=100, keep.pbam=FALSE,
keep.multihits=TRUE, chroms=NULL)
```

Arguments

bamFile	Bam file with the sample to analyze. Index of bam file must be in the same directory.
verbose	Set to TRUE to display progress information.
seed	Set seed of random number generator.
mc.cores.int	Number of cores to use when loading bam files. This is a memory intensive step, therefore number of cores must be chosen according to available RAM memory.
mc.cores	Number of cores to use in expression estimation.
genomeDB	knownGenome object containing annotated genome, as returned by the procGenome function.
readLength	Read length in bp, e.g. in a paired-end experiment where 75bp are sequenced on each end one would set readLength=75.

rpkm	Set to FALSE to return relative expression levels, i.e. the proportion of reads generated from each variant per gene. These proportions add up to 1 for each gene. Set to TRUE to return reads per kilobase per million (RPKM).
priorq	Parameter of the prior distribution on the proportion of reads coming from each variant. The prior is Dirichlet with prior sample size for each variant equal to priorq. We recommend priorq=2 for estimation, as it pools the estimated expression away from 0 and 1 and returned lower estimation errors than priorq=1 in our simulated experiments.
priorqGeneExpr	Parameter for prior distribution on overall gene expression. Defaults to 2, which ensures non-zero estimates for all genes
citype	Set to "none" to return no credibility intervals. Set to "asymp" to return approximate 95% CIs (obtained via the delta method). Set to "exact" to obtain exact CIs via Monte Carlo simulation. Options "asymp" and especially "exact" can increase the computation time substantially.
niter	Number of Monte Carlo iterations. Only used when citype=="exact".
burnin	Number of burnin Monte Carlo iterations. Only used when citype=="exact".
keep.pbam	Set to TRUE to save processed bam object, as returned by procBam. This object can require substantial memory during execution and disk storage upon saving and is not needed for a default analysis.
keep.multihits	Set to FALSE to discard reads aligned to multiple positions.
chroms	Manually set chromosomes to be processed. By default only main chromosomes are considered (except 'chrM')

Details

Function executes the functions procBam, getDistrs and pathCounts for each chromosome. Once finished, distributions are merged and expressions computed. Distributions, processed reads, path counts and expression estimates are returned.

Value

distr	Object of class readDistrs
pbam	List of objects of class procBam with one element per chromosome
pc	Object of class pathCounts
exp	Object of class expressionSet

Author(s)

Camille Stephan-Otto Attolini

Examples

```
## genDB<-makeTranscriptDbFromUCSC(genome="hg19", tablename="refGene")
## hg19DB <- procGenome(genDB, "hg19")
##  bamFile="/path_to_bam/sorted.bam"
## ans <- wrapKnown(bamFile=bamFile, mc.cores.int=4, mc.cores=3, genomeDB=hg19DB, readLength=101)
##  names(ans)
##  head(exprs(ans$exp))
```

Index