Package 'fishpond'

April 12, 2022

Title Fishpond: differential transcript and gene expression with inferential replicates

Version 2.0.1

Maintainer Michael Love <michaelisaiahlove@gmail.com>

Description Fishpond contains methods for differential transcript and gene expression analysis of RNA-seq data using inferential replicates for uncertainty of abundance quantification, as generated by Gibbs sampling or bootstrap sampling. Also the package contains utilities for working with Salmon and Alevin quantification files.

Imports graphics, stats, utils, methods, abind, gtools, qvalue, S4Vectors, SummarizedExperiment, matrixStats, svMisc, Rcpp, Matrix, SingleCellExperiment, jsonlite

Suggests testthat, knitr, rmarkdown, macrophage, tximeta, org.Hs.eg.db, samr, DESeq2, apeglm, tximportData, limma

LinkingTo Rcpp

SystemRequirements C++11

License GPL-2

Encoding UTF-8

URL https://github.com/mikelove/fishpond

biocViews Sequencing, RNASeq, GeneExpression, Transcription, Normalization, Regression, MultipleComparison, BatchEffect, Visualization, DifferentialExpression, DifferentialSplicing, AlternativeSplicing, SingleCell

VignetteBuilder knitr

LazyData true

RoxygenNote 7.1.2

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

git_branch RELEASE_3_14

git_last_commit 64edeb7

git_last_commit_date 2021-11-30

Date/Publication 2022-04-12

Author Anzi Zhu [aut, ctb],
Michael Love [aut, cre],
Avi Srivastava [aut, ctb],
Rob Patro [aut, ctb],
Joseph Ibrahim [aut, ctb],
Hirak Sarkar [ctb],
Euphy Wu [ctb],
Scott Van Buren [ctb],
Dongze He [ctb],
Steve Lianoglou [ctb],
Wes Wilson [ctb]

R topics documented:

`fishpond-package` *Downstream methods for Salmon and Alevin expression data*

Description

This package provides statistical methods and other tools for working with Salmon and Alevin quantification of RNA-seq data. In particular, it contains the Swish non-parametric method for detecting differential transcript expression (DTE). Swish can also be used to detect differential gene expresion (DGE).

Details

The main functions are:

- scaleInfReps - scaling transcript or gene expression data
- labelKeep - labelling which features have sufficient counts
- swish - perform non-parametric differential analysis
- Plots, e.g., plotMASwish, plotInfReps
- isoformProportions - convert counts to isoform proportions
- makeInfReps - create pseudo-inferential replicates
- splitSwish - split Swish analysis across jobs with Snakemake

All software-related questions should be posted to the Bioconductor Support Site:

https://support.bioconductor.org

The code can be viewed at the GitHub repository, which also lists the contributor code of conduct:

https://github.com/mikelove/fishpond

Author(s)

Anqi Zhu, Avi Srivastava, Joseph G. Ibrahim, Rob Patro, Michael I. Love

References

Swish method:

Zhu, A., Srivastava, A., Ibrahim, J.G., Patro, R., Love, M.I. (2019) Nonparametric expression analysis using inferential replicate counts. Nucleic Acids Research. https://doi.org/10.1093/nar/gkz622

Compression, makeInfReps and splitSwish:

Van Buren, S., Sarkar, H., Srivastava, A., Rashid, N.U., Patro, R., Love, M.I. (2020) Compression of quantification uncertainty for scRNA-seq counts. bioRxiv. https://doi.org/10.1101/2020.07.06.189639

---

addStatsFromCSV *Read statistics and nulls from CSV file*

Description

After running splitSwish and the associated Snakefile, this function can be used to gather and add the results to the original object. See the alevin section of the vignette for an example.

Usage

```
addStatsFromCSV(y = NULL, infile, estPi0 = FALSE)
```

Arguments

y	a SummarizedExperiment (if NULL, function will output a data.frame)
infile	character, path to the summary.csv file
estPi0	logical, see swish

Value

the SummarizedExperiment with metadata columns added, or if y is NULL, a data.frame of compiled results

---

computeInfRV *Compute inferential relative variance (InfRV)*

Description

InfRV is used the Swish publication for visualization. This function provides computation of the mean InfRV, a simple statistic that measures inferential uncertainty. It also computes and adds the mean and variance of inferential replicates, which can be useful ahead of plotInfReps. Note that InfRV is not used in the swish statistical method at all, it is just for visualization. See function code for details.

Usage

```
computeInfRV(y, pc = 5, shift = 0.01, meanVariance, useCounts = FALSE)
```

Arguments

y	a SummarizedExperiment
pc	a pseudocount parameter for the denominator
shift	a final shift parameter
meanVariance	logical, use pre-computed inferential mean and variance assays instead of counts and computed variance from infReps. If missing, will use pre-computed mean and variance when present
useCounts	logical, whether to use the MLE count matrix for the mean instead of mean of inferential replicates. this argument is for backwards compatability, as previous versions used counts. Default is FALSE

Value

a SummarizedExperiment with meanInfRV in the metadata columns

deswish *deswish: DESeq2-apeglm With Inferential Samples Helps*

Description

The DESeq2-apeglm With Inferential Samples implementation supposes a hierarchical distribution of log2 fold changes. The final posterior standard deviation is calculated by adding the posterior variance from modeling biological replicates computed by `apeglm`, and the observed variance on the posterior mode over inferential replicates. This function requires the DESeq2 and apeglm packages to be installed and will print an error if they are not found.

Usage

```
deswish(y, x, coef)
```

Arguments

`y`	a SummarizedExperiment containing the inferential replicate matrices, as output by `tximeta`, and then with `labelKeep` applied. One does not need to run `scaleInfReps` as scaling is done internally via DESeq2.
`x`	the design matrix
`coef`	the coefficient to test (see `lfcShrink`)

Value

a SummarizedExperiment with metadata columns added: the log2 fold change and posterior SD using inferential replicates, and the original log2 fold change (apeglm) and its posterior SD

References

The `DESeq` and `lfcShrink` function in the `DESeq2` package:

Zhu, Ibrahim, Love "Heavy-tailed prior distributions for sequence count data: removing the noise and preserving large differences" Bioinformatics (2018).

Love, Huber, Anders "Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2" Genome Biology (2014).

Examples

```
# a small example... 500 genes, 10 inf reps
y <- makeSimSwishData(m=500, numReps=10)
y <- labelKeep(y)
#y <- deswish(y, ~condition, "condition_2_vs_1")
```

getTrace *Obtain a trace of inferential replicates for a sample*

Description

Simple helper function to obtain a trace (e.g. MCMC trace) of the ordered inferential replicates for one samples. Supports either multiple features, `idx`, or multiple samples, `samp_idx` (not both). Returns a tidy data.frame for easy plotting.

Usage

```
getTrace(y, idx, samp_idx)
```

Arguments

`y`	a SummarizedExperiment with inferential replicates as assays `infRep1` etc.
`idx`	the names or row numbers of the gene or transcript to plot
`samp_idx`	the names or column numbers of the samples to plot

Value

a data.frame with the counts along the interential replicates, possible with additional columns specifying feature or sample

Examples

```
y <- makeSimSwishData()
getTrace(y, "gene-1", "s1")
```

importAllelicCounts *Import allelic counts as a SummarizedExperiment*

Description

Read in Salmon quantification of allelic counts from a diploid transcriptome. Assumes that diploid transcripts are marked with the following suffix: an underscore and a consistent symbol for each of the two alleles, e.g. `ENST123_M` and `ENST123_P`, or `ENST123_alt` and `ENST123_ref`. There must be exactly two alleles for each transcript, and the `--keep-duplicates` option should be used in Salmon indexing to avoid removing transcripts with identical sequence. The output object has half the number of transcripts, with the two alleles either stored in a `"wide"` object, or as re-named `"assays"`. Note carefully that the symbol provided to `a1` is used as the effect allele, and `a2` is used as the non-effect allele (see the `format` argument description and Value description below).

Usage

```
importAllelicCounts(
  coldata,
  a1,
  a2,
  format = c("wide", "assays"),
  tx2gene = NULL,
  ...
)
```

Arguments

coldata	a data.frame as used in tximeta
a1	the symbol for the effect allele
a2	the symbol for the non-effect allele
format	either "wide" or "assays" for whether to combine the allelic counts as columns (wide) or put the allelic count information in different assay slots (assays). For wide output, the data for the non-effect allele (a2) comes first, then the effect allele (a1), e.g. [a2 | a1]. The ref level of the factor variable se$allele will be "a2" (so by default comparisons will be: a1 vs a2). For assays output, all of the original matrices are renamed with a prefix, either a1- or a2-.
tx2gene	optional, a data.frame with first column indicating transcripts, second column indicating genes (or any other transcript grouping). Either this should include the a1 and a2 suffix for the transcripts and genes, or those will be added internally, if it is detected that the first transcript does not have these suffices. For example if _alt or _ref, or _M or _P (as indicated by the a1 and a2 arguments) are not present in the table, the table rows will be duplicated with those suffices added on behalf of the user. If not provided, the output object will be transcript-level. Note: do not attempt to set the txOut argument, it will conflict with internal calls to downstream functions. Note: if the a1/a2 suffices are not at the end of the transcript name in the quantification files, e.g. ENST123_M|<metadata>, then ignoreAfterBar=TRUE can be used to match regardless of the string following | in the quantification files.
...	any arguments to pass to tximeta

Details

Requires the tximeta package. skipMeta=TRUE is used, as it is assumed the diploid transcriptome does not match any reference transcript collection. This may change in future iterations of the function, depending on developments in upstream software.

Value

a SummarizedExperiment, with allele counts (and other data) combined into a wide matrix [a2 | a1], or as assays (a1, then a2). The original strings associated with a1 and a2 are stored in the metadata of the object, in the alleles list element. Note the ref level of se$allele will be "a2", such that comparisons by default will be a1 vs a2 (effect vs non-effect).

isoformProportions *Create isoform proportions from scaled data*

Description

Takes output of scaled (and optionally filtered) counts and returns isoform proportions by dividing out the total scaled count for the gene for each sample. The operation is performed on the `counts` assay, then creating a new assay called `isoProp`, and on all of the inferential replicates, turning them from counts into isoform proportions. Any transcripts (rows) from single isoform genes are removed, and the transcripts will be re-ordered by gene ID.

Usage

```
isoformProportions(y, geneCol = "gene_id", quiet = FALSE)
```

Arguments

`y`	a SummarizedExperiment
`geneCol`	the name of the gene ID column in the metadata columns for the rows of `y`
`quiet`	display no messages

Value

a SummarizedExperiment, with single-isoform transcripts removed, and transcripts now ordered by gene

labelKeep *Label rows to keep based on minimal count*

Description

Adds a column `keep` to `mcols(y)` that specifies which rows of the SummarizedExperiment will be included in statistical testing. Rows are not removed, just marked with the logical `keep`.

Usage

```
labelKeep(y, minCount = 10, minN = 3, x)
```

Arguments

`y`	a SummarizedExperiment
`minCount`	the minimum count
`minN`	the minimum sample size at `minCount`
`x`	the name of the condition variable, will use the smaller of the two groups to set `minN`. Similar to edgeR's `filterByExpr`, as the smaller group grows past 10, `minN` grows only by 0.7 increments of sample size

Value

a SummarizedExperiment with a new column keep in mcols(y)

Examples

```
y <- makeSimSwishData()
y <- scaleInfReps(y)
y <- labelKeep(y)
```

---

loadFry *Load in data from alevin-fry USA mode*

Description

Enables easy loading of sparse data matrices provided by alevin-fry USA mode. Alevin-fry - https://www.biorxiv.org/content/10.1101/2021.06.29.450377v1

Usage

```
load_fry_raw(fryDir, quiet = FALSE)

loadFry(fryDir, outputFormat = "scRNA", nonzero = FALSE, quiet = FALSE)
```

Arguments

fryDir	path to the output directory returned by alevin-fry quant command. This directory should contain a metainfo.json, and an alevin folder which contains quants_mat.mtx, quants_mat_cols.txt and quants_mat_rows.txt
quiet	logical whether to display no messages
outputFormat	can be *either* be a list that defines the desired format of the output SingleCellExperiment object *or* a string that represents one of the pre-defined output formats, which are "scRNA", "snRNA", "scVelo" and "velocity". See details for the explainations of the pre-defined formats and how to define custom format.
nonzero	whether to filter cells with non-zero expression value across all genes (default FALSE). If TRUE, this will filter based on all assays. If a string vector of assay names, it will filter based on the matching assays in the vector.

Value

A SingleCellExperiment object that contains one or more assays. Each assay consists of a gene by cell count matrix. The row names are feature names, and the column names are cell barcodes

Details about `loadFry`

This function consumes the result folder returned by running alevin-fry quant in unspliced, spliced, ambiguous (USA) quantification mode, and returns a `SingleCellExperiement` object that contains a final count for each gene within each cell. In USA mode, alevin-fry quant returns a count matrix contains three types of count for each feature (gene) within each sample (cell or nucleus), which represent the spliced mRNA count of the gene (S), the unspliced mRNA count of the gene (U), and the count of UMIs whose splicing status is ambiguous for the gene (A). For each assay defined by `outputFormat`, these three counts of a gene within a cell will be summed to get the final count of the gene according to the rule defined in the `outputFormat`. The returned object will contains the desired assays defined by `outputFormat`, with rownames as the barcode of samples and colnames as the feature names.

Details about the output format

The `outputFormat` argument takes *either* be a list that defines the desired format of the output `SingleCellExperiment` object *or* a string that represents one of the pre-defined output format.

Currently the pre-defined formats of the output `SingleCellExperiment` object are:

"scRNA": This format is recommended for single cell experiments. It returns a `counts` assay that contains the S+A count of each gene in each cell.

"snRNA": This format is recommended for single nucleus experiments. It returns a `counts` assay that contains the U+S+A count of each gene in each cell.

"raw": This format put the three kinds of counts into three separate assays, which are `unspliced`, `spliced` and `ambiguous`.

"velocity": This format contains two assays. The `spliced` assay contains the S+A count of each gene in each cell. The `unspliced` assay contains the U counts of each gene in each cell.

"scVelo": This format is for direct entry into velociraptor R package or other scVelo downstream analysis pipeline for velocity analysis in R with Bioconductor. It adds the expected "S"-pliced assay and removes errors for size factors being non-positive.

A custom output format can be defined using a list. Each element in the list defines an assay in the output `SingleCellExperiment` object. The name of an element in the list will be the name of the corresponding assay in the output object. Each element in the list should be defined as a vector that takes at least one of the three kinds of count, which are U, S and A. See the provided toy example for defining a custom output format.

Details about `load_fry_raw`

This function processes alevin-fry's quantification result contained within the input folder.This function returns a list that consists of the gene count matrix, the gene names list, the barcode list, and some metadata, such as the number of genes in the experiment and whether alevin-fry was executed in USA mode. In the returned list, the all-in-one count matrix, `count_mat`, returned from the USA mode of alevin-fry consists of the spliced count of genes defined in `gene.names` for all barcodes defined in `barcodes`, followed by the unspliced count of genes in the same order for all cells, then followed by the ambiguous count of genes in the same order for all cells.

Examples

```
# Get path for minimal example avelin-fry output dir
testdat <- fishpond:::readExampleFryData("fry-usa-basic")

# This is exactly how the velocity format defined internally.
custom_velocity_format <- list("spliced"=c("S","A"), "unspliced"=c("U"))

# Load alevin-fry gene quantification in velocity format
sce <- loadFry(fryDir=testdat$parent_dir, outputFormat=custom_velocity_format)
SummarizedExperiment::assayNames(sce)

# Load the same data but use pre-defined, velociraptor R pckage desired format
scvelo_format <- "scVelo"

scev <- loadFry(fryDir=testdat$parent_dir, outputFormat=scvelo_format, nonzero=TRUE)
SummarizedExperiment::assayNames(scev)
```

`makeInfReps` *Make pseudo-inferential replicates from mean and variance*

Description

Makes pseudo-inferential replicate counts from `mean` and `variance` assays. The simulated counts are drawn from a negative binomial distribution, with `mu=mean` and `size` set using a method of moments estimator for dispersion.

Usage

```
makeInfReps(y, numReps, minDisp = 0.001)
```

Arguments

`y`	a SummarizedExperiment
`numReps`	how many inferential replicates
`minDisp`	the minimal dispersion value, set after method of moments estimation from inferential mean and variance

Details

Note that these simulated counts only reflect marginal variance (one transcript or gene at a time), and do not capture the covariance of counts across transcripts or genes, unlike imported inferential replicate data. Therefore, `makeInfReps` should not be used with `summarizeToGene` to create gene-level inferential replicates if inferential replicates were originally created on the transcript level. Instead, import the original inferential replicates.

Value

a SummarizedExperiment

References

Van Buren, S., Sarkar, H., Srivastava, A., Rashid, N.U., Patro, R., Love, M.I. (2020) Compression of quantification uncertainty for scRNA-seq counts. bioRxiv. https://doi.org/10.1101/2020.07.06.189639

Examples

```
library(SummarizedExperiment)
mean <- matrix(1:4,ncol=2)
variance <- mean
se <- SummarizedExperiment(list(mean=mean, variance=variance))
se <- makeInfReps(se, numReps=50)
```

`makeSimSwishData` *Make simulated data for swish for examples/testing*

Description

Makes a small swish dataset for examples and testing. The first six genes have some differential expression evidence in the counts, with varying degree of inferential variance across inferential replicates (1-2: minor, 3-4: some, 5-6: substantial). The 7th and 8th genes have all zeros to demonstrate `labelKeep`.

Usage

```
makeSimSwishData(
  m = 1000,
  n = 10,
  numReps = 20,
  null = FALSE,
  meanVariance = FALSE
)
```

Arguments

`m`	number of genes
`n`	number of samples
`numReps`	how many inferential replicates to generate
`null`	logical, whether to make an all null dataset
`meanVariance`	logical, whether to output only mean and variance of inferential replicates

Value

a SummarizedExperiment

Examples

```
library(SummarizedExperiment)
y <- makeSimSwishData()
assayNames(y)
```

`miniSwish` *Helper function for distributing Swish on a subset of data*

Description

This function is called by the `Snakefile` that is generated by `splitSwish`. See alevin example in the vignette. As such, it doesn't need to be run by users in an interactive R session.

Usage

```
miniSwish(
  infile,
  outfile,
  numReps = 20,
  lengthCorrect = FALSE,
  overwrite = FALSE,
  ...
)
```

Arguments

`infile`	path to an RDS file of a SummarizedExperiment
`outfile`	a CSV file to write out
`numReps`	how many inferential replicates to generate
`lengthCorrect`	logical, see `scaleInfReps`, and Swish vignette. As this function is primarily for alevin, the default is `FALSE`
`overwrite`	logical, whether `outfile` should overwrite an existing file
`...`	arguments passed to `swish`

Details

Note that the default for length correction is FALSE, as opposed to the default in `scaleInfReps` which is TRUE. The default for `numReps` here is 20.

Value

nothing, files are written out

plotInfReps *Plot inferential replicates for a gene or transcript*

Description

For datasets with inferential replicates, boxplots are drawn for the two groups and potentially grouped by covariates. For datasets with only mean and variance, points and intervals (95 approximation) are drawn. Additionally, for numeric x values, points and intervals will be drawn and computeInfRV should be run first in order to add the mean and variance statistics.

Usage

```
plotInfReps(
  y,
  idx,
  x,
  cov = NULL,
 colsDrk = c("dodgerblue", "goldenrod4", "royalblue4", "red3", "purple4", "darkgreen"),
  colsLgt = c("lightblue1", "goldenrod1", "royalblue1", "salmon1", "orchid1",
    "limegreen"),
  xaxis,
  xlab,
  ylim,
  main,
  mainCol,
  legend = FALSE,
  legendPos = "topleft",
  legendTitle = FALSE,
  legendCex = 1,
  useMean = TRUE,
  q = qnorm(0.975),
  applySF = FALSE,
  reorder,
  thin
)
```

Arguments

y	a SummarizedExperiment (see swish)
idx	the name or row number of the gene or transcript
x	the name of the condition variable for splitting and coloring the samples or cells. Also can be a numeric, e.g. pseudotime, in which case, cov can be used to designate groups for coloring
cov	the name of the covariate for adjustment
colsDrk	dark colors for the lines of the boxes

colsLgt	light colors for the inside of the boxes
xaxis	logical, whether to label the sample numbers. default is TRUE if there are less than 30 samples
xlab	the x-axis label
ylim	y limits
main	title
mainCol	name of metadata column to use for title (instead of rowname)
legend	logical, show simple legend (default FALSE)
legendPos	character, position of the legend (default "topleft")
legendTitle	logical, whether to add the name of the grouping variable as a title on the legend (default FALSE)
legendCex	numeric, size of the legend (default 1)
useMean	logical, when inferential replicates are not present or when x is continuous, whether to use the mean assay or the counts assay for plotting
q	numeric, the quantile to use when plotting the intervals when inferential replicates are not present or when x is continuous. Default is qnorm(.975) ~= 1.96 corresponding to 95 intervals
applySF	logical, when inferential replicates are not present, should y$sizeFactor be divided out from the mean and interval plots (default FALSE)
reorder	logical, should points within a group defined by condition and covariate be reordered by their count value (default is FALSE, except for alevin data)
thin	integer, should the mean and interval lines be drawn thin (the default switches from 0 [not thin] to 1 [thinner] at n=150 cells, and from 1 [thinner] to 2 [thinnest] at n=400 cells)

Value

nothing, a plot is displayed

Examples

```
y <- makeSimSwishData()
plotInfReps(y, 3, "condition")

y <- makeSimSwishData(n=40)
y$batch <- factor(rep(c(1,2,3,1,2,3),c(5,10,5,5,10,5)))
plotInfReps(y, 3, "condition", "batch")
```

plotMASwish *MA plot*

Description

MA plot

Usage

```
plotMASwish(y, alpha = 0.05, sigcolor = "blue", ...)
```

Arguments

y	a SummarizedExperiment (see swish)
alpha	the FDR threshold for coloring points
sigcolor	the color for the significant points
...	passed to plot

Value

nothing, a plot is displayed

Examples

```
y <- makeSimSwishData()
y <- scaleInfReps(y)
y <- labelKeep(y)
y <- swish(y, x="condition")
plotMASwish(y)
```

readEDS *readEDS - a utility function for quickly reading in Alevin's EDS format*

Description

readEDS - a utility function for quickly reading in Alevin's EDS format

Usage

```
readEDS(numOfGenes, numOfOriginalCells, countMatFilename, tierImport = FALSE)
```

Arguments

numOfGenes	number of genes
numOfOriginalCells	number of cells
countMatFilename	pointer to the EDS file, quants_mat.gz
tierImport	whether the countMatFilename refers to a quants tier file

Value

a genes x cells sparse matrix, of the class dgCMatrix

---

scaleInfReps *Scale inferential replicate counts*

Description

A helper function to scale the inferential replicates to the mean sequencing depth. The scaling takes into account a robust estimator of size factor (median ratio method is used). First, counts are corrected per row using the effective lengths (for gene counts, the average transcript lengths), then scaled per column to the geometric mean sequence depth, and finally are adjusted per-column up or down by the median ratio size factor to minimize systematic differences across samples.

Usage

```
scaleInfReps(
  y,
  lengthCorrect = TRUE,
  meanDepth = NULL,
  sfFun = NULL,
  minCount = 10,
  minN = 3,
  saveMeanScaled = FALSE,
  quiet = FALSE
)
```

Arguments

y	a SummarizedExperiment with: infReps a list of inferential replicate count matrices, counts the estimated counts matrix, and length the effective lengths matrix
lengthCorrect	whether to use effective length correction (default is TRUE)
meanDepth	(optional) user can specify a different mean sequencing depth. By default the geometric mean sequencing depth is computed

sfFun	(optional) size factors function. An alternative to the median ratio can be provided here to adjust the scaledTPM so as to remove remaining library size differences. Alternatively, one can provide a numeric vector of size factors
minCount	for internal filtering, the minimum count
minN	for internal filtering, the minimum sample size at minCount
saveMeanScaled	store the mean of scaled inferential replicates as an assay 'meanScaled'
quiet	display no messages

Value

a SummarizedExperiment with the inferential replicates as scaledTPM with library size already corrected (no need for further normalization). A column log10mean is also added which is the log10 of the mean of scaled counts across all samples and all inferential replicates.

Examples

```
y <- makeSimSwishData()
y <- scaleInfReps(y)
```

---

splitSwish *Function for splitting SummarizedExperiment into separate RDS files*

Description

The splitSwish function splits up the y object along genes and writes a Snakefile that can be used with Snakemake to distribute running swish across genes. This workflow is primarily designed for large single cell datasets, and so the default is to not perform length correction within the distributed jobs. See the alevin section of the vignette for an example. See the Snakemake documention for details on how to run and customize a Snakefile: https://snakemake.readthedocs.io

Usage

```
splitSwish(y, nsplits, prefix = "swish", snakefile = NULL, overwrite = FALSE)
```

Arguments

y	a SummarizedExperiment
nsplits	integer, how many pieces to break y into
prefix	character, the path of the RDS files to write out, e.g. prefix="/path/to/swish" will generate swish.rds files at this path
snakefile	character, the path of a Snakemake file, e.g. Snakefile, that should be written out. If NULL, then no Snakefile is written out
overwrite	logical, whether the snakefile and RDS files (swish1.rds, ...) should overwrite existing files

Value

nothing, files are written out

References

Compression and splitting across jobs:

Van Buren, S., Sarkar, H., Srivastava, A., Rashid, N.U., Patro, R., Love, M.I. (2020) Compression of quantification uncertainty for scRNA-seq counts. bioRxiv. https://doi.org/10.1101/2020.07.06.189639

Snakemake:

Koster, J., Rahmann, S. (2012) Snakemake - a scalable bioinformatics workflow engine. Bioinformatics. https://doi.org/10.1093/bioinformatics/bts480

`swish` *swish: SAMseq With Inferential Samples Helps*

Description

Performs non-parametric inference on rows of `y` for various experimental designs. See References for details.

Usage

```
swish(
  y,
  x,
  cov = NULL,
  pair = NULL,
  interaction = FALSE,
  cor = c("none", "spearman", "pearson"),
  nperms = 100,
  estPi0 = FALSE,
  qvaluePkg = "qvalue",
  pc = 5,
  nRandomPairs = 30,
  fast = 1,
  returnNulls = FALSE,
  quiet = FALSE
)
```

Arguments

`y`	a SummarizedExperiment containing the inferential replicate matrices of median-ratio-scaled TPM as assays 'infRep1', 'infRep2', etc.

x	the name of the condition variable. A factor with two levels for a two group analysis (possible to adjust for covariate or matched samples, see next two arguments). The log fold change is computed as non-reference level over reference level (see vignette: 'Note on factor levels')
cov	the name of the covariate for adjustment. If provided a stratified Wilcoxon in performed. Cannot be used with pair (unless using cor)
pair	the name of the pair variable, which should be the number of the pair. Can be an integer or factor. If specified, a signed rank test is used to build the statistic. All samples across x must be pairs if this is specified. Cannot be used with cov (unless using cor)
interaction	logical, whether to perform a test of an interaction between x and cov. See Details.
cor	character, whether to compute correlation of x with the log counts, and signifance testing on the correlation as a test statistic. Either "spearman" or "pearson" correlations can be computed. For Spearman the correlation is computed over ranks of x and ranks of inferential replicates. For Pearson, the correlation is computed for x and log2 of the inferential replicates plus pc. Default is "none", e.g. two-group comparison using the rank sum test or other alternatives listed above. Additionally, correlation can be computed between a continuous variable cov and log fold changes across x matched by pair
nperms	the number of permutations. if set above the possible number of permutations, the function will print a message that the value is set to the maximum number of permutations possible
estPi0	logical, whether to estimate pi0
qvaluePkg	character, which package to use for q-value estimation, samr or qvalue
pc	pseudocount for finite estimation of log2FC, not used in calculation of test statistics, locfdr or qvalue
nRandomPairs	the number of random pseudo-pairs (only used with interaction=TRUE and un-matched samples) to use to calculate the test statistic
fast	an integer, toggles different methods based on speed (fast=1 is default, 0 is slower). See Details.
returnNulls	logical, only return the stat vector, the log2FC vector, and the nulls matrix (default FALSE)
quiet	display no messages

Details

interaction: The interaction tests are different than the other tests produced by swish, in that they focus on a difference in the log2 fold change across levels of x when comparing the two levels in cov. If pair is specified, this will perform a Wilcoxon rank sum test on the two groups of matched sample LFCs. If pair is not included, multiple random pairs of samples within the two groups are chosen, and again a Wilcoxon rank sum test compared the LFCs across groups.

fast: '0' involves recomputing ranks of the inferential replicates for each permutation, '1' (default) is roughly 10x faster by avoiding re-computing ranks for each permutation. The fast argument is only relevant for the following three experimental designs: (1) two group Wilcoxon, (2) stratified

Wilcoxon, e.g. `cov` is specified, and (3) the paired interaction test, e.g. `pair` and `cov` are specified. For paired design and general interaction test, there are not fast/slow alternatives.

Value

a SummarizedExperiment with metadata columns added: the statistic (either a centered Wilcoxon Mann-Whitney or a signed rank statistic, aggregated over inferential replicates), a log2 fold change (the median over inferential replicates, and averaged over pairs or groups (if groups, weighted by sample size), the local FDR and q-value, as estimated by the `samr` package.

References

The citation for `swish` method is:

Anqi Zhu, Avi Srivastava, Joseph G Ibrahim, Rob Patro, Michael I Love "Nonparametric expression analysis using inferential replicate counts" Nucleic Acids Research (2019). https://doi.org/10.1093/nar/gkz622

The `swish` method builds upon the `SAMseq` method, and extends it by incorporating inferential uncertainty, as well as providing methods for additional experimental designs (see vignette).

For reference, the publication describing the `SAMseq` method is:

Jun Li and Robert Tibshirani "Finding consistent patterns: A nonparametric approach for identifying differential expression in RNA-Seq data" Stat Methods Med Res (2013). https://doi.org/10.1177/0962280211428386

Examples

```
library(SummarizedExperiment)
set.seed(1)
y <- makeSimSwishData()
y <- scaleInfReps(y)
y <- labelKeep(y)
y <- swish(y, x="condition")

# histogram of the swish statistics
hist(mcols(y)$stat, breaks=40, col="grey")
cols = rep(c("blue","purple","red"),each=2)
for (i in 1:6) {
  arrows(mcols(y)$stat[i], 20,
         mcols(y)$stat[i], 10,
         col=cols[i], length=.1, lwd=2)
}

# plot inferential replicates
plotInfReps(y, 1, "condition")
plotInfReps(y, 3, "condition")
plotInfReps(y, 5, "condition")
```

Index