Package 'SoupX'

May 26, 2022

Title Single Cell mRNA Soup eXterminator

Version 1.6.1

Date 2022-05-25

Author Matthew Daniel Young

Maintainer Matthew Daniel Young <my4@sanger.ac.uk>

Description Quantify, profile and remove ambient mRNA contamination (the ``soup'') from droplet based single cell RNA-seq experiments. Implements the method described in Young et al. (2018) <doi:10.1101/303727>.

URL https://github.com/constantAmateur/SoupX

Suggests knitr, rstan, DropletUtils, rmarkdown, formatR

VignetteBuilder knitr

Imports ggplot2, Matrix, methods, Seurat (>= 3.2.2)

Depends R (>= 3.5.0)

LazyData true

LazyDataCompression xz

License GPL-2

Encoding UTF-8

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2022-05-26 09:50:02 UTC

R topics documented:

`adjustCounts` *Remove background contamination from count matrix*

Description

After the level of background contamination has been estimated or specified for a channel, calculate the resulting corrected count matrix with background contamination removed.

Usage

```
adjustCounts(
  sc,
  clusters = NULL,
  method = c("subtraction", "soupOnly", "multinomial"),
  roundToInt = FALSE,
  verbose = 1,
  tol = 0.001,
  pCut = 0.01,
  ...
)
```

Arguments

sc	A SoupChannel object.
clusters	A vector of cluster IDs, named by cellIDs. If NULL clusters auto-loaded from sc. If FALSE, no clusters are used. See details.
method	Method to use for correction. See details. One of 'multinomial', 'soupOnly', or 'subtraction'
roundToInt	Should the resulting matrix be rounded to integers?
verbose	Integer giving level of verbosity. 0 = silence, 1 = Basic information, 2 = Very chatty, 3 = Debug.
tol	Allowed deviation from expected number of soup counts. Don't change this.
pCut	The p-value cut-off used when method='soupOnly'.
...	Passed to expandClusters.

Details

This essentially subtracts off the mean expected background counts for each gene, then redistributes any "unused" counts. A count is unused if its subtraction has no effect. For example, subtracting a count from a gene that has zero counts to begin with.

As expression data is highly sparse at the single cell level, it is highly recommended that clustering information be provided to allow the subtraction method to share information between cells. Without grouping cells into clusters, it is difficult (and usually impossible) to tell the difference between a count of 1 due to background contamination and a count of 1 due to endogenous expression. This ambiguity is removed at the cluster level where counts can be aggregated across cells. This information can then be propagated back to the individual cell level to provide a more accurate removal of contaminating counts.

To provide clustering information, either set clustering on the SoupChannel object with setClusters or explicitly passing the clusters parameter.

If roundToInt=TRUE, this function will round the result to integers. That is, it will take the floor of the connected value and then round back up with probability equal to the fractional part of the number.

The method parameter controls how the removal of counts in performed. This should almost always be left at the default ('subtraction'), which iteratively subtracts counts from all genes as described above. The 'soupOnly' method will use a p-value based estimation procedure to identify those genes that can be confidently identified as having endogenous expression and removes everything else (described in greater detail below). Because this method either removes all or none of the expression for a gene in a cell, the correction procedure is much faster. Finally, the 'multinomial' method explicitly maximises the multinomial likelihood for each cell. This method gives essentially identical results as 'subtraction' and is considerably slower.

In greater detail, the 'soupOnly' method is done by sorting genes within each cell by their p-value under the null of the expected soup fraction using a Poisson model. So that genes that definitely do have a endogenous contribution are at the end of the list with p=0. Those genes for which there is poor evidence of endogenous cell expression are removed, until we have removed approximately nUMIs*rho molecules. The cut-off to prevent removal of genes above nUMIs*rho in each cell is achieved by calculating a separate p-value for the total number of counts removed to exceed nUMIs*rho, again using a Poisson model. The two p-values are combined using Fisher's method and

the cut-off is applied to the resulting combined p-value calculated using a chi-squared distribution with 4 degrees of freedom.

Value

A modified version of the table of counts, with background contamination removed.

Examples

```
out = adjustCounts(scToy)
#Return integer counts only
out = adjustCounts(scToy,roundToInt=TRUE)
```

---

alloc *Allocate values to "buckets" subject to weights and constraints*

Description

Allocates `tgt` of something to `length(bucketLims)` different "buckets" subject to the constraint that each bucket has a maximum value of `bucketLims` that cannot be exceeded. By default counts are distributed equally between buckets, but weights can be provided using `ws` to have the redistribution prefer certain buckets over others.

Usage

```
alloc(tgt, bucketLims, ws = rep(1/length(bucketLims), length(bucketLims)))
```

Arguments

`tgt`	Value to distribute between buckets.
`bucketLims`	The maximum value that each bucket can take. Must be a vector of positive values.
`ws`	Weights to be used for each bucket. Default value makes all buckets equally likely.

Value

A vector of the same length as `bucketLims` containing values distributed into buckets.

autoEstCont *Automatically calculate the contamination fraction*

Description

The idea of this method is that genes that are highly expressed in the soup and are marker genes for some population can be used to estimate the background contamination. Marker genes are identified using the tfidf method (see quickMarkers). The contamination fraction is then calculated at the cluster level for each of these genes and clusters are then aggressively pruned to remove those that give implausible estimates.

Usage

```
autoEstCont(
  sc,
  topMarkers = NULL,
  tfidfMin = 1,
  soupQuantile = 0.9,
  maxMarkers = 100,
  contaminationRange = c(0.01, 0.8),
  rhoMaxFDR = 0.2,
  priorRho = 0.05,
  priorRhoStdDev = 0.1,
  doPlot = TRUE,
  forceAccept = FALSE,
  verbose = TRUE
)
```

Arguments

sc	The SoupChannel object.
topMarkers	A data.frame giving marker genes. Must be sorted by decreasing specificity of marker and include a column 'gene' that contains the gene name. If set to NULL, markers are estimated using quickMarkers.
tfidfMin	Minimum value of tfidf to accept for a marker gene.
soupQuantile	Only use genes that are at or above this expression quantile in the soup. This prevents inaccurate estimates due to using genes with poorly constrained contribution to the background.
maxMarkers	If we have heaps of good markers, keep only the best maxMarkers of them.
contaminationRange	Vector of length 2 that constrains the contamination fraction to lie within this range. Must be between 0 and 1. The high end of this range is passed to estimateNonExpressingCells as maximumContamination.
rhoMaxFDR	False discovery rate passed to estimateNonExpressingCells, to test if rho is less than maximumContamination.

priorRho	Mode of gamma distribution prior on contamination fraction.
priorRhoStdDev	Standard deviation of gamma distribution prior on contamination fraction.
doPlot	Create a plot showing the density of estimates?
forceAccept	Passed to setContaminationFraction. Should we allow very high contamination fractions to be used.
verbose	Be verbose?

Details

This set of marker genes is filtered to include only those with tf-idf value greater than tfidfMin. A higher tf-idf value implies a more specific marker. Specifically a cut-off t implies that a marker gene has the property that geneFreqGlobal < exp(-t/geneFreqInClust). See quickMarkers. It may be necessary to decrease this value for data sets with few good markers.

This set of marker genes is filtered down to include only the genes that are highly expressed in the soup, controlled by the soupQuantile parameter. Genes highly expressed in the soup provide a more precise estimate of the contamination fraction.

The pruning of implausible clusters is based on a call to estimateNonExpressingCells. The parameters maximumContamination=max(contaminationRange) and rhoMaxFDR are passed to this function. The defaults set here are calibrated to aggressively prune anything that has even the weakest of evidence that it is genuinely expressed.

For each cluster/gene pair the posterior distribution of the contamination fraction is calculated (based on gamma prior, controlled by priorRho and priorRhoStdDev). These posterior distributions are aggregated to produce a final estimate of the contamination fraction. The logic behind this is that estimates from clusters that truly estimate the contamination fraction will cluster around the true value, while erroneous estimates will be spread out across the range (0,1) without a 'preferred value'. The most probable value of the contamination fraction is then taken as the final global contamination fraction.

Value

A modified SoupChannel object where the global contamination rate has been set. Information about the estimation is also stored in the slot fit

Note

This function assumes that the channel contains multiple distinct cell types with different marker genes. If you try and run it on a channel with very homogenous cells (e.g. a cell line, flow-sorted cells), you will likely get a warning, an error, and/or an extremely high contamination estimate. In such circumstances your best option is usually to manually set the contamination to something reasonable.

See Also

quickMarkers

Examples

```
#Use less specific markers
scToy = autoEstCont(scToy,tfidfMin=0.8)
#Allow large contamination fractions to be allocated
scToy = autoEstCont(scToy,forceAccept=TRUE)
#Be quiet
scToy = autoEstCont(scToy,verbose=FALSE,doPlot=FALSE)
```

`calculateContaminationFraction`

Calculate the contamination fraction

Description

This function computes the contamination fraction using two user-provided bits of information. Firstly, a list of sets of genes that can be biologically assumed to be absent in at least some cells in your data set. For example, these might be haemoglobin genes or immunoglobulin genes, which should not be expressed outside of erythroyctes and antibody producing cells respectively.

Usage

```
calculateContaminationFraction(
  sc,
  nonExpressedGeneList,
  useToEst,
  verbose = TRUE,
  forceAccept = FALSE
)
```

Arguments

`sc`	A SoupChannel object.
`nonExpressedGeneList`	A list containing sets of genes which can be assumed to be non-expressed in a subset of cells (see details).
`useToEst`	A boolean matrix of dimensions ncol(toc) x length(nonExpressedGeneList) indicating which gene-sets should not be assumed to be non-expressed in each cell. Row names must correspond to the names of `nonExpressedGeneList`. Usually produced by `estimateNonExpressingCells`.
`verbose`	Print best estimate.
`forceAccept`	Passed to `setContaminationFraction`.

Details

Secondly, this function needs to know which cells definitely do not express the gene sets described above. Continuing with the haemoglobin example, which are the erythrocytes that are producing haemoglobin mRNAs and which are non-erythrocytes that we can safely assume produce no such genes. The assumption made is any expression from a gene set in cell marked as a "non-expressor" for that gene set, must be derived from the soup. Therefore, the level of contamination present can be estimated from the amount of expression of these genes seen in these cells.

Most often, the genesets are user supplied based on your knowledge of the experiment and the cells in which they are genuinely expressed is estimated using `estimateNonExpressingCells`. However, they can also be supplied directly if other information is available.

Usually, there is very little variation in the contamination fraction within a channel and very little power to detect the contamination accurately at a single cell level. As such, the default mode of operation simply estimates one value of the contamination fraction that is applied to all cells in a channel.

The global model fits a simple Poisson glm to the aggregated count data across all cells.

Finally, note that if you are not able to find a reliable set of genes to use for contamination estimation, or you do not trust the values produced, the contamination fraction can be manually set by the user using `setContaminationFraction`.

Value

A modified version of `sc` with estimates of the contamination (rho) added to the metaData table.

Examples

```
#Common gene list in real world data
geneList = list(HB=c('HBB','HBA2'))
#Gene list appropriate to toy data
geneList = list(CD7 = 'CD7')
ute = estimateNonExpressingCells(scToy,geneList)
sc = calculateContaminationFraction(scToy,geneList,ute)
```

`estimateNonExpressingCells` *Calculate which cells genuinely do not express a particular gene or set of genes*

Description

Given a list of correlated genes (e.g. Haemoglobin genes, Immunoglobulin genes, etc.), make an attempt to estimate which cells genuinely do not express each of these gene sets in turn. The central idea is that in cells that are not genuinely producing a class of mRNAs (such as haemoglobin genes), any observed expression of these genes must be due to ambient RNA contamination. As such, if we can identify these cells, we can use the observed level of expression of these genes to estimate the level of contamination.

Usage

```
estimateNonExpressingCells(
  sc,
  nonExpressedGeneList,
  clusters = NULL,
  maximumContamination = 1,
  FDR = 0.05
)
```

Arguments

sc	A SoupChannel object.
nonExpressedGeneList	A list containing sets of genes which will be used to estimate the contamination fraction.
clusters	A named vector indicating how to cluster cells. Names should be cell IDs, values cluster IDs. If NULL, we will attempt to load it from sc$metaData$clusters. If set to FALSE, each cell will be considered individually.
maximumContamination	The maximum contamination fraction that you would reasonably expect. The lower this value is set, the more aggressively cells are excluded from use in estimation.
FDR	A Poisson test is used to identify cells to exclude, this is the false discovery rate it uses. Higher FDR = more aggressive exclusion.

Details

The ideal way to do this would be to have a prior annotation of your data indicating which cells are (for instance) red blood cells and genuinely expression haemoglobin genes, and which do not and so only express haemoglobin genes due to contamination. If this is your circumstance, there is no need to run this function, you can instead pass a matrix encoding which cells are haemoglobin expressing and which are not to `calculateContaminationFraction` via the `useToEst` parameter.

This function will use a conservative approach to excluding cells that it thinks may express one of your gene sets. This is because falsely including a cell in the set of non-expressing cells may erroneously inflate your estimated contamination, whereas failing to include a genuine non-expressing cell in this set has no significant effect.

To this end, this function will exclude any cluster of cells in which any cell is deemed to have genuine expression of a gene set. Clustering of data is beyond the scope of this package, but can be performed by the user. In the case of 10X data mapped using cellranger and loaded using `load10X`, the cellranger graph based clustering is automatically loaded and used.

To decide if a cell is genuinely expressing a set of genes, a Poisson test is used. This tests whether the observed expression is greater than `maximumContamination` times the expected number of counts for a set of genes, if the cell were assumed to be derived wholly from the background. This process can be made less conservative (i.e., excluding fewer cells/clusters) by either decreasing the value of the maximum contamination the user believes is plausible (`maximumContamination`) or making the significance threshold for the test more strict (by reducing `FDR`).

Value

A matrix indicating which cells to be used to estimate contamination for each set of genes. Typically passed to the `useToEst` parameter of `calculateContaminationFraction` or `plotMarkerMap`.

See Also

calculateContaminationFraction plotMarkerMap

Examples

```
#Common gene list in real world data
geneList = list(HB=c('HBB','HBA2'))
#Gene list appropriate to toy data
geneList = list(CD7 = 'CD7')
ute = estimateNonExpressingCells(scToy,geneList)
```

`estimateSoup` *Get expression profile of soup*

Description

This is usually called by `SoupChannel`, rather than directly by the user. Uses the empty droplets in the range provided to calculate the expression profile of the soup under the assumption that these droplets only contain background.

Usage

```
estimateSoup(sc, soupRange = c(0, 100), keepDroplets = FALSE)
```

Arguments

`sc`	A `SoupChannel` object.
`soupRange`	Droplets with total UMI count in this range (excluding endpoints) are used to estimate soup.
`keepDroplets`	Storing the full table of counts for all droplets uses a lot of space and is really only used to estimate the soup profile. Therefore, it is dropped after the soup profile has been estimated unless this is set to `TRUE`.

Value

A modified version of `sc` with an extra `soupProfile` entry containing a data.frame with the soup profile and confidence limits for all genes.

Examples

```
#Load droplet and count tables
tod = Seurat::Read10X(system.file('extdata','toyData','raw_gene_bc_matrices','GRCh38',
                                  package='SoupX'))
toc = Seurat::Read10X(system.file('extdata','toyData','filtered_gene_bc_matrices','GRCh38',
                                  package='SoupX'))
#Suppress calculation of soup profile automatically on load
sc = SoupChannel(tod,toc,calcSoupProfile=FALSE)
#Retain table of droplets
sc = estimateSoup(sc,keepDroplets=TRUE)
#Or use non-default values
sc = estimateSoup(sc,soupRange=c(60,100))
```

---

expandClusters *Expands soup counts calculated at the cluster level to the cell level*

Description

Given a clustering of cells and soup counts calculated for each of those clusters, determines a most likely allocation of soup counts at the cell level.

Usage

```
expandClusters(clustSoupCnts, cellObsCnts, clusters, cellWeights, verbose = 1)
```

Arguments

clustSoupCnts	Matrix of genes (rows) by clusters (columns) where counts are number of soup counts for that gene/cluster combination.
cellObsCnts	Matrix of genes (rows) by cells (columns) giving the observed counts
clusters	Mapping from cells to clusters.
cellWeights	Weighting to give to each cell when distributing counts. This would usually be set to the number of expected soup counts for each cell.
verbose	Integer giving level of verbosity. 0 = silence, 1 = Basic information, 2 = Very chatty, 3 = Debug.

Value

A matrix of genes (rows) by cells (columns) giving the number of soup counts estimated for each cell. Non-integer values possible.

initProgBar *Create Seurat style progress bar*

Description

Creates progress bar that won't ruin log files and shows progress towards 100

Usage

```
initProgBar(min, max, ...)
```

Arguments

min	Minimum value of parameter.
max	Maximum value of parameter.
...	Passed to txtProgressBar

Value

A txtProgressBar object to use updating progress.

load10X *Load a collection of 10X data-sets*

Description

Loads unfiltered 10X data from each data-set and identifies which droplets are cells using the cellranger defaults.

Usage

```
load10X(
  dataDir,
  cellIDs = NULL,
  channelName = NULL,
  readArgs = list(),
  includeFeatures = c("Gene Expression"),
  verbose = TRUE,
  ...
)
```

Arguments

dataDir	Top level cellranger output directory (the directory that contains the raw_gene_bc_matrices folder).
cellIDs	Barcodes of droplets that contain cells. If NULL, use the default cellranger set.
channelName	The name of the channel to store. If NULL set to either names(dataDir) or dataDir is no name is set.
readArgs	A list of extra parameters passed to Seurat::Read10X.
includeFeatures	If multiple feature types are present, keep only the types mentioned here and collapse to a single matrix.
verbose	Be verbose?
...	Extra parameters passed to SoupChannel construction function.

Value

A SoupChannel object containing the count tables for the 10X dataset.

See Also

SoupChannel estimateSoup

Examples

```
sc = load10X(system.file('extdata','toyData',package='SoupX'))
```

PBMC_metaData *PBMC 4K meta data*

Description

Collection of bits of meta data relating to the 10X PBMC 4K data.

Usage

```
data(PBMC_metaData)
```

Format

PBMC_metaData is a data.frame with 4 columns: RD1, RD2, Cluster, and Annotation.

Details

This data set pertains to the 10X demonstration PBMC 4K data and includes metadata about it in the `data.frame` named `PBMC_metaData`.

`PBMC_metaData` was created using Seurat (v2) to calculate a tSNE representation of the data and cluster cells with these commands.

- `set.seed(1)`
- `srat = CreateSeuratObject(sc$toc)`
- `srat = NormalizeData(srat)`
- `srat = ScaleData(srat)`
- `srat = FindVariableGenes(srat)`
- `srat = RunPCA(srat,pcs.compute=30)`
- `srat = RunTSNE(srat,dims.use=seq(30))`
- `srat = FindClusters(srat,dims.use=seq(30),resolution=1)`
- `PBMC_metaData = as.data.frame(srat@dr$tsne@cell.embeddings)`
- `colnames(PBMC_metaData) = c('RD1','RD2')`
- `PBMC_metaData$Cluster = factor(srat@meta.data[rownames(PBMC_metaData),'res.1'])`
- `PBMC_metaData$Annotation = factor(c('7'='B','4'='B','1'='T_CD4','2'='T_CD4','3'='T_CD8','5'='T_C`

Source

https://support.10xgenomics.com/single-cell-gene-expression/datasets/2.1.0/pbmc4k

---

PBMC_sc *SoupChannel from PBMC data*

Description

`SoupChannel` created from 10X demonstration PBMC 4k data. The cells have been sub-sampled by a factor of 2 to reduce file size of package.

Usage

```
data(PBMC_sc)
```

Format

`PBMC_sc` is a `SoupChannel` object with 33,694 genes and 2,170 cells.

Details

PBMC_sc was created by running the following commands.

- set.seed(1137)
- tmpDir = tempdir(check=TRUE)
- download.file('http://cf.10xgenomics.com/samples/cell-exp/2.1.0/pbmc4k/pbmc4k_raw_gene_bc_matri
- download.file('http://cf.10xgenomics.com/samples/cell-exp/2.1.0/pbmc4k/pbmc4k_filtered_gene_bc_
- untar(file.path(tmpDir,'tod.tar.gz'),exdir=tmpDir)
- untar(file.path(tmpDir,'toc.tar.gz'),exdir=tmpDir)
- library(SoupX)
- PBMC_sc = load10X(tmpDir,calcSoupProfile=FALSE)
- PBMC_sc = SoupChannel(PBMC_sc$tod,PBMC_sc$toc[,sample(ncol(PBMC_sc$toc),round(ncol(PBMC_sc$toc)*

Source

https://support.10xgenomics.com/single-cell-gene-expression/datasets/2.1.0/pbmc4k

---

plotChangeMap *Plot maps comparing corrected/raw expression*

Description

Given some reduced dimensional representation of the data (such as UMAP or tSNE) that has been calculated however you would like, this provides a way to visualise how the expression of a geneSet changes after soup correction.

Usage

```
plotChangeMap(
  sc,
  cleanedMatrix,
  geneSet,
  DR,
  dataType = c("soupFrac", "binary", "counts"),
  logData = FALSE,
  pointSize = 0.5
)
```

Arguments

sc	SoupChannel object.
cleanedMatrix	A cleaned matrix to compare against the raw one. Usually the output of adjustCounts.
geneSet	A vector with the names of the genes to aggregate and plot evidence for.

DR	A data.frame, with rows named by unique cell IDs (i.e., <ChannelName>_<Barcode>) the first two columns of which give the coordinates of each cell in some reduced dimension representation of the data.
dataType	How should data be represented. Binary sets each cell to expressed or not, counts converts everything to counts, soupFrac plots the fraction of the observed counts that are identified as contamination (i.e., (old-new)/old) for each cell and is the default.
logData	Should we log the thing we plot?
pointSize	Size of points

Value

A ggplot2 containing the plot.

Examples

```
out = adjustCounts(scToy)
gg = plotChangeMap(scToy,out,'S100A9')
```

---

plotMarkerDistribution

Plots the distribution of the observed to expected expression for marker genes

---

Description

If each cell were made up purely of background reads, the expression fraction would equal that of the soup. This plot compares this expectation of pure background to the observed expression fraction in each cell, for each of the groups of genes in `nonExpressedGeneList`. For each group of genes, the distribution of this ratio is plotted across all cells. A value significantly greater than 1 (0 on log scale) can only be obtained if some of the genes in each group are genuinely expressed by the cell. That is, the assumption that the cell is pure background does not hold for that gene.

Usage

```
plotMarkerDistribution(
  sc,
  nonExpressedGeneList,
  maxCells = 150,
  tfidfMin = 1,
  ...
)
```

Arguments

sc	A SoupChannel object.
nonExpressedGeneList	Which sets of genes to use to estimate soup (see calculateContaminationFraction).
maxCells	Randomly plot only this many cells to prevent over-crowding.
tfidfMin	Minimum specificity cut-off used if finding marker genes (see quickMarkers).
...	Passed to estimateNonExpressingCells

Details

This plot is a useful diagnostic for the assumption that a list of genes is non-expressed in most cell types. For non-expressed cells, the ratio should cluster around the contamination fraction, while for expressed cells it should be elevated. The most useful non-expressed gene sets are those for which the genes are either strongly expressed, or not expressed at all. Such groups of genes will show up in this plot as a bimodal distribution, with one mode containing the cells that do not express these genes around the contamination fraction for this channel and another around a value at some value equal to or greater than 0 (1 on non-log scale) for the expressed cells.

The red line shows the global estimate of the contamination for each group of markers. This is usually lower than the low mode of the distribution as there will typically be a non-negligible number of cells with 0 observed counts (and hence -infinity log ratio).

If nonExpressedGeneList is missing, this function will try and find genes that are very specific to different clusters, as these are often the most useful in estimating the contamination fraction. This is meant only as a heuristic, which can hopefully provide some inspiration as to a class of genes to use to estimation the contamination for your experiment. Please do **NOT** blindly use the top N genes found in this way to estimate the contamination. That is, do not feed this list of genes into calculateContaminationFraction without any manual consideration or filtering as this *will over-estimate your contamination* (often by a large amount). For this reason, these gene names are not returned by the function.

Value

A ggplot2 object containing the plot.

Examples

```
gg = plotMarkerDistribution(scToy,list(CD7='CD7',LTB='LTB'))
```

---

plotMarkerMap *Plot ratio of observed to expected counts on reduced dimension map*

Description

Given some reduced dimensional representation of the data (such as UMAP or tSNE) that has been calculated however you would like, this provides a way to visualise how likely a set of genes are to be soup derived on that map. That is, given a set of genes, this function calculates how many counts would be expected if that droplet were nothing but soup and compares that to the observed count. This is done via a log2 ratio of the two values. A Poisson test is performed and points that have a statistically significant enrichment over the background (at 5

Usage

```
plotMarkerMap(
  sc,
  geneSet,
  DR,
  ratLims = c(-2, 2),
  FDR = 0.05,
  useToEst = NULL,
  pointSize = 2,
  pointShape = 21,
  pointStroke = 0.5,
  naPointSize = 0.25
)
```

Arguments

sc	SoupChannel object.
geneSet	A vector with the names of the genes to aggregate and plot evidence for.
DR	A data.frame, with rows named by unique cell IDs (i.e., <ChannelName>_<Barcode>) the first two columns of which give the coordinates of each cell in some reduced dimension representation of the data. Try and fetch automatically if missing.
ratLims	Truncate log ratios at these values.
FDR	False Discovery Rate for statistical test of enrichment over background.
useToEst	A vector (usually obtained from estimateNonExpressingCells), that will be used to mark cells instead of the usual Poisson test.
pointSize	Size of points
pointShape	Shape of points
pointStroke	Stroke size for points
naPointSize	Point size for NAs.

Value

A ggplot2 containing the plot.

Examples

```
gg = plotMarkerMap(scToy,'CD7')
```

plotSoupCorrelation *Plot correlation of expression profiles of soup and aggregated cells*

Description

Calculates an expression profile by aggregating counts across all cells and plots this (on a log10 scale) against the expression profile of the soup.

Usage

```
plotSoupCorrelation(sc)
```

Arguments

sc	A SoupChannel object.

Value

A ggplot2 object containing the plot.

print.SoupChannel *Print method for SoupChannel*

Description

Prints a summary of a SoupChannel object.

Usage

```
## S3 method for class 'SoupChannel'
print(x, ...)
```

Arguments

x	A SoupChannel object.
...	Currently unused.

Value

Nothing. Prints message to console.

`quickMarkers` *Gets top N markers for each cluster*

Description

Uses tf-idf ordering to get the top N markers of each cluster. For each cluster, either the top N or all genes passing the hypergeometric test with the FDR specified, whichever list is smallest.

Usage

```
quickMarkers(toc, clusters, N = 10, FDR = 0.01, expressCut = 0.9)
```

Arguments

`toc`	Table of counts. Must be a sparse matrix.
`clusters`	Vector of length `ncol(toc)` giving cluster membership.
`N`	Number of marker genes to return per cluster.
`FDR`	False discover rate to use.
`expressCut`	Value above which a gene is considered expressed.

Details

Term Frequency - Inverse Document Frequency is used in natural language processing to identify terms specific to documents. This function uses the same idea to order genes within a group by how predictive of that group they are. The main advantage of this is that it is extremely fast and gives reasonable results.

To do this, gene expression is binarised in each cell so each cell is either considered to express or not each gene. That is, we replace the counts with `toc > zeroCut`. The frequency with which a gene is expressed within the target group is compared to the global frequency to calculate the tf-idf score. We also calculate a multiple hypothesis corrected p-value based on a hypergeometric test, but this is extremely permissive.

Value

data.frame with top N markers (or all that pass the hypergeometric test) and their statistics for each cluster.

Examples

```
#Calculate markers of clusters in toy data
mrks = quickMarkers(scToy$toc,scToy$metaData$clusters)
## Not run:
#Calculate markers from Seurat (v3) object
mrks = quickMarkers(srat@assays$RNA@count,srat@active.ident)

## End(Not run)
```

scToy *Toy SoupChanel object*

Description

A SoupChannel object created from the toy data used in examples.

Usage

```
data(scToy)
```

Format

scToy is a SoupChannel object.

Details

The toy data is created from a modified version of the extremely reduced Seurat pbmc_small dataset. It includes clusters, tSNE coordinates and a flat estimate of 0.1 contamination. It includes data for only 226 genes and 62 cells and should not be used for anything other than testing functions as it is not representative of real data in any way.

setClusters *Sets clustering for SoupChannel*

Description

Adds or updates clustering information to meta-data table in SoupChannel object.

Usage

```
setClusters(sc, clusters)
```

Arguments

sc	A SoupChannel object.
clusters	A named vector, where entries are the cluster IDs and names are cellIDs. If no names are provided, the order is assumed to match the order in sc$metaData.

Value

An updated SoupChannel object with clustering information stored.

Examples

```
sc = load10X(system.file('extdata','toyData',package='SoupX'))
mDat = read.table(system.file('extdata','toyData','metaData.tsv',package='SoupX'),sep='\t')
sc = setClusters(sc,mDat$res.1)
```

setContaminationFraction *Manually set contamination fraction*

Description

Manually specify the contamination fraction.

Usage

```
setContaminationFraction(sc, contFrac, forceAccept = FALSE)
```

Arguments

sc	A SoupChannel object.
contFrac	The contamination fraction. Either a constant, in which case the same value is used for all cells, or a named vector, in which case the value is set for each cell.
forceAccept	A warning or error is usually returned for extremely high contamination fractions. Setting this to TRUE will turn these into messages and proceed.

Value

A modified SoupChannel object for which the contamination (rho) has been set.

Examples

```
sc = load10X(system.file('extdata','toyData',package='SoupX'))
sc = setContaminationFraction(sc,0.1)
```

setDR *Manually set dimension reduction for a channel*

Description

Manually specify the dimension reduction

Usage

```
setDR(sc, DR, reductName = NULL)
```

Arguments

sc	A SoupChannel object.
DR	The dimension reduction coordinates (e.g., tSNE). This must be a data.frame, with two columns giving the two dimension reduction coordinates. The data.frame must either have row names matching the row names of sc$metaData, or be ordered in the same order as sc$metaData.
reductName	What to name the reduction (defaults to column names provided).

Value

A modified SoupChannel object for which the dimension reduction has been set.

Examples

```
sc = load10X(system.file('extdata','toyData',package='SoupX'))
mDat = read.table(system.file('extdata','toyData','metaData.tsv',package='SoupX'),sep='\t')
sc = setDR(sc,mDat[,c('tSNE_1','tSNE_2')])
```

---

setSoupProfile *Set soup profile*

Description

Manually sets or updates the soup profile for a SoupChannel object.

Usage

```
setSoupProfile(sc, soupProfile)
```

Arguments

sc	A SoupChannel object.
soupProfile	A data.frame with columns `est` containing the fraction of soup for each gene, `counts` containing the total counts for each gene and with row names corresponding to the row names of `sc$toc`.

Value

An updated SoupChannel object with the soup profile set.

Examples

```
#Suppose only table of counts is available
toc = Seurat::Read10X(system.file('extdata','toyData','filtered_gene_bc_matrices','GRCh38',
                                  package='SoupX'))
#Suppress calculating soup profile automatically
sc = SoupChannel(toc,toc,calcSoupProfile=FALSE)
#And add manually
rowSums = Matrix::rowSums
soupProf = data.frame(row.names = rownames(toc),est=rowSums(toc)/sum(toc),counts=rowSums(toc))
sc = setSoupProfile(sc,soupProf)
```

SoupChannel *Construct a SoupChannel object*

Description

Creates a SoupChannel object that contains everything related to the soup estimation of a single channel.

Usage

```
SoupChannel(tod, toc, metaData = NULL, calcSoupProfile = TRUE, ...)
```

Arguments

tod	Table of droplets. A matrix with columns being each droplet and rows each gene.
toc	Table of counts. Just those columns of tod that contain cells.
metaData	Meta data pertaining to the cells. Optional. Must be a data-frame with rownames equal to column names of toc.
calcSoupProfile	By default, the soup profile is calculated using estimateSoup with default values. If you want to do something other than the defaults, set this to FALSE and call estimateSoup manually.
...	Any other named parameters to store.

Value

A SoupChannel object.

See Also

SoupChannelList estimateSoup setSoupProfile setClusters

Examples

```
#Load droplet and count tables
tod = Seurat::Read10X(system.file('extdata','toyData','raw_gene_bc_matrices','GRCh38',
                                  package='SoupX'))
toc = Seurat::Read10X(system.file('extdata','toyData','filtered_gene_bc_matrices','GRCh38',
                                  package='SoupX'))
#Default calculates soup profile
sc = SoupChannel(tod,toc)
names(sc)
#This can be suppressed
sc = SoupChannel(tod,toc,calcSoupProfile=FALSE)
names(sc)
```

SoupX *SoupX: Profile, quantify and remove ambient RNA expression from droplet based RNA-seq*

Description

This package implements the method described in REF. First a few notes about nomenclature: soup - Used a shorthand to refer to the ambient RNA which is contained in the input solution to droplet based RNA-seq experiments and ends up being sequenced along with the cell endogenous RNAs that the experiment is aiming to quantify. channel - This refers to a single run input into a droplet based sequencing platform. For Chromium 10X 3' sequencing there are currently 8 "channels" per run of the instrument. Because the profile of the soup depends on the input solution, this is the minimal unit on which the soup should be estimated and subtracted.

Details

The essential step in performing background correction is deciding which genes are not expressed in a reasonable fraction of cells. This is because SoupX estimates the contamination fraction by comparing the expression of these non-expressed genes in droplets containing cells to the soup defined from empty droplets. For solid tissue, the set of Haemoglobin genes usually works well. The key properties a gene should have are: - it should be easy to identify when it is truly expressed (i.e., when it's expressed, it should be highly expressed) - it should be highly specific to a certain cell type or group of cell types so that when the expression level is low, you can be confident that the expression is coming from the soup and not a very low level of expression from the cell

Spike-in RNAs are the best case scenario. In the case where you do not have spike-ins and haemoglobin genes are not viable estimators, the user should begin by using the plotMarkerDistribution function to plot those genes with bi-modal distributions that have a pattern of expression across cells that is consistent with high cell-type specificity. The user should then select a set of genes that can be used for estimation from this list. One or two high quality genes is usually sufficient to obtain a good estimate for the average contamination level of a channel.

Index