Package 'sccore'

July 13, 2023

Title Core Utilities for Single-Cell RNA-Seq

Version 1.0.4

Description Core utilities for single-cell RNA-seq data analysis. Contained within are utility functions for working with differential expression (DE) matrices and count matrices, a collection of functions for manipulating and plotting data via 'ggplot2', and functions to work with cell graphs and cell embeddings. Graph-based methods include embedding kNN cell graphs into a UMAP <doi:10.21105/joss.00861>, collapsing vertices of each cluster in the graph, and propagating graph labels.

License GPL-3

Encoding UTF-8

LazyData true

Imports dplyr, ggplot2, ggrepel, graphics, grDevices, igraph, irlba, magrittr, Matrix, methods, parallel, pbmcapply, pROC, Rcpp, rlang, scales, stats, tibble, utils, uwot, withr

Depends R (>= 3.5.0)

Suggests ggrastr (>= 0.1.7), jsonlite, rmumps, testthat

RoxygenNote 7.2.3

LinkingTo Rcpp, RcppArmadillo, RcppProgress, RcppEigen

NeedsCompilation yes

URL https://github.com/kharchenkolab/sccore

BugReports https://github.com/kharchenkolab/sccore/issues

Author Viktor Petukhov [aut],
Rasmus Rydbirk [aut],
Peter Kharchenko [aut],
Evan Biederstedt [aut, cre]

Maintainer Evan Biederstedt <evan.biederstedt@gmail.com>

Repository CRAN

Date/Publication 2023-07-12 23:50:02 UTC

R topics documented:

adjacentVertices *List of adjacent vertices from igraph object*

Description

List of adjacent vertices from igraph object

Usage

```
adjacentVertices(edge_verts)
```

Arguments

edge_verts	edge vertices of igraph graph object

Value

list of adjacent vertices

Examples

```
## Not run:
edges <- igraph::as_edgelist(conosGraph)
adjacentVertices(edges)

## End(Not run)
```

adjacent_vertex_weights *List of adjacent vertex weights from igraph object*

Description

List of adjacent vertex weights from igraph object

Usage

```
adjacent_vertex_weights(edge_verts, edge_weights)
```

Arguments

edge_verts	edge vertices of igraph graph object
edge_weights	edge weights of igraph graph object

Value

list of adjacent vertices

Examples

```
## Not run:
edges <- igraph::as_edgelist(conosGraph)
edge.weights <- igraph::edge.attributes(conosGraph)$weight
adjacent_vertex_weights(edges, edge.weights)

## End(Not run)
```

`appendSpecificityMetricsToDE` *Append specificity metrics to DE*

Description

Append specificity metrics to DE

Usage

```
appendSpecificityMetricsToDE(
  de.df,
  clusters,
  cluster.id,
  p2.counts,
  low.expression.threshold = 0,
  append.auc = FALSE
)
```

Arguments

`de.df`	data.frame of differential expression values
`clusters`	factor of clusters
`cluster.id`	names of 'clusters' factor. If a cluster.id doesn't exist in cluster names, an error is thrown.
`p2.counts`	counts from Pagoda2, refer to <https://github.com/kharchenkolab/pagoda2>
`low.expression.threshold`	numeric Threshold to remove expression values (default=0). Values under this threshold are discarded.
`append.auc`	boolean If TRUE, append AUC values (default=FALSE)

Value

data.frame of differential expression values with metrics attached

as_factor *convert character vector into a factor with names "values" and "levels"*

Description

convert character vector into a factor with names "values" and "levels"

Usage

```
as_factor(vals)
```

Arguments

vals	vector of values to evaluate

Value

factor with names "values" and "levels"

cellAnnotations *Conos cell annotations*

Description

Conos cell annotations

Usage

```
cellAnnotations
```

Format

An object of class `character` of length 3000.

checkPackageInstalled *Check whether a package is installed and suggest how to install from CRAN, Bioconductor, or other external source*

Description

Check whether a package is installed and suggest how to install from CRAN, Bioconductor, or other external source

Usage

```
checkPackageInstalled(
  pkgs,
  details = "to run this function",
  install.help = NULL,
  bioc = FALSE,
  cran = FALSE
)
```

Arguments

pkgs	character Package name(s)
details	character Helper text (default = "to run this function")
install.help	character Additional information on how to install package (default = NULL)
bioc	logical Package(s) is/are available from Bioconductor (default = FALSE)
cran	logical Package(s) is/are available from CRAN (default = FALSE)

Examples

```
## Not run:
checkPackageInstalled("sccore", cran = TRUE)

## End(Not run)
```

collapseCellsByType *Collapse count matrices by cell type, given min/max number of cells*

Description

Collapse count matrices by cell type, given min/max number of cells

Usage

```
collapseCellsByType(cm, groups, min.cell.count = 10, max.cell.count = Inf)
```

Arguments

cm	count matrix
groups	factor specifying cell types
min.cell.count	numeric Minimum number of cells to include (default=10)
max.cell.count	numeric Maximum number of cells to include (default=Inf). If Inf, there is no maximum.

Value

Subsetted factor of collapsed cells by type, with NA cells omitted

---

collapseGraphPaga *Collapse graph using PAGA 1.2 algorithm, Wolf et al 2019, Genome Biology (2019) <https://genomebiology.biomedcentral.com/articles/10.1186/s13059-019-1663-x>*

Description

Collapse graph using PAGA 1.2 algorithm, Wolf et al 2019, Genome Biology (2019) <https://genomebiology.biomedcentral.c
019-1663-x>

Usage

```
collapseGraphPaga(graph, groups, linearize = TRUE, winsorize = FALSE)
```

Arguments

graph	igraph graph object Graph to be collapsed
groups	factor on vertices describing cluster assignment (can specify integer vertex ids, or character vertex names which will be matched)
linearize	should normally be always TRUE (default=TRUE)
winsorize	winsorize final connectivity statistics value (default=FALSE) Note: Original PAGA has it as always TRUE, but in this case there is no way to distinguish level of connectivity for highly connected groups.

Value

collapsed graph

collapseGraphSum *Collapse Graph By Sum*

Description

Collapse Graph By Sum

Usage

```
collapseGraphSum(graph, groups, normalize = TRUE)
```

Arguments

graph	igraph graph object Graph to be collapsed
groups	factor on vertices describing cluster assignment (can specify integer vertex ids, or character vertex names which will be matched)
normalize	boolean Whether to recalculate edge weight as observed/expected (default=TRUE)

Value

collapsed graph

Examples

```
collapsed = collapseGraphPaga(conosGraph, igraph::V(conosGraph), linearize=TRUE, winsorize=FALSE)
```

colSumByFactor *Calculates factor-stratified sums for each column*

Description

Calculates factor-stratified sums for each column

Usage

```
colSumByFactor(sY, rowSel)
```

Arguments

sY	sparse matrix (dgCmatrix)
rowSel	integer factor. Note that the 0-th column will return sums for any NA values; 0 or negative values will be omitted

Value

Matrix

`conosClusterList` *Conos clusters list*

Description

Conos clusters list

Usage

```
conosClusterList
```

Format

An object of class `list` of length 2.

`conosGraph` *Conos graph*

Description

Conos graph

Usage

```
conosGraph
```

Format

An object of class `igraph` of length 100.

dotPlot *Dot plot adapted from Seurat:::DotPlot, see ?Seurat:::DotPlot for details*

Description

Dot plot adapted from Seurat:::DotPlot, see ?Seurat:::DotPlot for details

Usage

```
dotPlot(
  markers,
  count.matrix,
  cell.groups,
  marker.colour = "black",
  cluster.colour = "black",
  xlab = "Marker",
  ylab = "Cluster",
  n.cores = 1,
  text.angle = 45,
  gene.order = NULL,
  cols = c("blue", "red"),
  col.min = -2.5,
  col.max = 2.5,
  dot.min = 0,
  dot.scale = 6,
  scale.by = "radius",
  scale.center = FALSE,
  scale.min = NA,
  scale.max = NA,
  verbose = FALSE,
  ...
)
```

Arguments

markers	Vector of gene markers to plot
count.matrix	Merged count matrix, cells in rows and genes in columns
cell.groups	Named factor containing cell groups (clusters) and cell names as names
marker.colour	Character or numeric vector (default="black")
cluster.colour	Character or numeric vector (default="black")
xlab	string X-axis title (default="Marker")
ylab	string Y-axis title (default="Cluster")
n.cores	integer Number of cores (default=1)
text.angle	numeric Angle of text displayed (default=45)

gene.order	Either factor of genes passed to dplyr::mutate(levels=gene.order), or a boolean. (default=NULL) If TRUE, gene.order is set to the unique markers. If FALSE, gene.order is set to NULL. If NULL, the argument is ignored.
cols	Colors to plot (default=c("blue", "red")). The name of a palette from 'RColorBrewer::brewer.pal.info', a pair of colors defining a gradient, or 3+ colors defining multiple gradients (if 'split.by' is set).
col.min	numeric Minimum scaled average expression threshold (default=-2.5). Everything smaller will be set to this.
col.max	numeric Maximum scaled average expression threshold (default=2.5). Everything larger will be set to this.
dot.min	numeric The fraction of cells at which to draw the smallest dot (default=0). All cell groups with less than this expressing the given gene will have no dot drawn.
dot.scale	numeric Scale the size of the points, similar to cex (default=6)
scale.by	string Scale the size of the points by 'size' or by 'radius' (default="radius")
scale.center	boolean Center scaling, see 'scale()' argument 'center' (default=FALSE)
scale.min	numeric Set lower limit for scaling, use NA for default (default=NA)
scale.max	numeric Set upper limit for scaling, use NA for default (default=NA)
verbose	boolean Verbose output (default=TRUE)
...	Additional inputs passed to sccore::plapply(), see man for description.

Value

ggplot2 object

Examples

```
library(dplyr)
## Create merged count matrix
## In this example, cms is a list of count matrices from, e.g., Cellranger count,
## where cells are in columns and genes in rows
## cm <- sccore:::mergeCountMatrices(cms, transposed = FALSE) %>% Matrix::t()

## If coming from Conos, this can be extracted like so
## cm <- conos.obj$getJointCountMatrix(raw = FALSE) # Either normalized or raw values can be used

## Here, we create a random sparse matrix
cm <- Matrix::rsparsematrix(30,3,0.5) %>% abs(.) %>%
            `dimnames<-`(list(1:30,c("gene1","gene2","gene3")))

## Create marker vector
markers <- c("gene1","gene2","gene3")

## Additionally, color vectors can be included.
## These should have the same length as the input (markers, cell groups)
## Otherwise, they are recycled
col.markers <- c("black","black","red") # or c(1,1,2)
col.clusters <- c("black","red","black") # or c(1,2,1)
```

```
## Create annotation vector
annotation <- c(rep("cluster1",10),rep("cluster2",10),rep("cluster3",10)) %>%
    factor() %>% setNames(1:30)

## Plot. Here, the expression colours range from gray (low expression) to purple (high expression)
sccore:::dotPlot(markers = markers, count.matrix = cm, cell.groups = annotation,
    marker.colour = col.markers, cluster.colour = col.clusters, cols=c("gray","purple"))
```

`embeddingColorsPlot` *Set colors for embedding plot. Used primarily in embeddingPlot().*

Description

Set colors for embedding plot. Used primarily in embeddingPlot().

Usage

```
embeddingColorsPlot(
  plot.df,
  colors,
  groups = NULL,
  geom_point_w = ggplot2::geom_point,
  gradient.range.quantile = 1,
  color.range = "symmetric",
  legend.title = NULL,
  palette = NULL,
  plot.na = TRUE
)
```

Arguments

`plot.df`	data.frame for plotting. In embeddingPlot(), this is a tibble from tibble::rownames_to_column().
`colors`	vector of numbers, which must be shown with point colors, names contain cell names (default=NULL). This argument is ignored if groups are provided.
`groups`	vector of cluster labels, names contain cell names (default=NULL)
`geom_point_w`	function to work with geom_point layer from ggplot2 (default=ggplot2::geom_point)
`gradient.range.quantile`	Winsorization quantile for the numeric colors and gene gradient (default=1)
`color.range`	controls range, in which colors are estimated (default="symmetric"). Pass "all" to estimate range based on all values of "colors", pass "data" to estimate it only based on colors, presented in the embedding. Alternatively you can pass vector of length 2 with (min, max) values.
`legend.title`	legend title (default=NULL)

palette	vector or list or function (default=NULL). Accepts number of colors and return list of colors (i.e. see 'colorRampPalette') (default=NULL)
plot.na	boolean/numeric Whether to plot points, for which groups / colors are missed (default=is.null(subgroups), i.e. FALSE). If plot.na passed a numeric value below 0, the NA symbols are plotted below the cells. Otherwise if values >=0, they're plotted above the cells. Note that this argument is FALSE if 'subgroups' is NULL

Value

ggplot2 object

`embeddingGroupPlot` *Plotting function for cluster labels, names contain cell names. Used primarily in embeddingPlot().*

Description

Plotting function for cluster labels, names contain cell names. Used primarily in embeddingPlot().

Usage

```
embeddingGroupPlot(
  plot.df,
  groups,
  geom_point_w,
  min.cluster.size,
  mark.groups,
  font.size,
  legend.title,
  shuffle.colors,
  palette,
  plot.na,
  ...
)
```

Arguments

plot.df	data.frame for plotting. In embeddingPlot(), this is a tibble from tibble::rownames_to_column().
groups	vector of cluster labels, names contain cell names (default=NULL)
geom_point_w	function to work with geom_point layer from ggplot2 (default=ggplot2::geom_point)
min.cluster.size	labels for all groups with number of cells fewer than this parameter are considered as missed (default=0). This argument is ignored if groups aren't provided
mark.groups	plot cluster labels above points (default=TRUE)

font.size	font size for cluster labels (default=c(3, 7)). It can either be single number for constant font size or pair (min, max) for font size depending on cluster size
legend.title	legend title (default=NULL)
shuffle.colors	shuffle colors (default=FALSE)
palette	vector or list or function (default=NULL). Accepts number of colors and return list of colors (i.e. see 'colorRampPalette') (default=NULL)
plot.na	boolean/numeric Whether to plot points, for which groups / colors are missed (default=is.null(subgroups), i.e. FALSE). If plot.na passed a numeric value below 0, the NA symbols are plotted below the cells. Otherwise if values >=0, they're plotted above the cells. Note that this argument is FALSE if 'subgroups' is NULL
...	Additional arguments passed to ggplot2::geom_label_repel()

Value

ggplot2 object

`embeddingPlot` *Plot embedding with provided labels / colors using ggplot2*

Description

Plot embedding with provided labels / colors using ggplot2

Usage

```
embeddingPlot(
  embedding,
  groups = NULL,
  colors = NULL,
  subgroups = NULL,
  plot.na = is.null(subgroups),
  min.cluster.size = 0,
  mark.groups = TRUE,
  show.legend = FALSE,
  alpha = 0.4,
  size = 0.8,
  title = NULL,
  plot.theme = NULL,
  palette = NULL,
  color.range = "symmetric",
  font.size = c(3, 7),
  show.ticks = FALSE,
  show.labels = FALSE,
  legend.position = NULL,
  legend.title = NULL,
```

```
  gradient.range.quantile = 1,
  raster = FALSE,
  raster.dpi = 300,
  shuffle.colors = FALSE,
  keep.limits = !is.null(subgroups),
  ...
)
```

Arguments

embedding	two-column matrix with x and y coordinates of the embedding, rownames contain cell names and are used to match coordinates with groups or colors
groups	vector of cluster labels, names contain cell names (default=NULL)
colors	vector of numbers, which must be shown with point colors, names contain cell names (default=NULL). This argument is ignored if groups are provided.
subgroups	subset of 'groups', selecting the cells for plot (default=NULL). Ignored if 'groups' is NULL
plot.na	boolean/numeric Whether to plot points, for which groups / colors are missed (default=is.null(subgroups), i.e. FALSE). If plot.na passed a numeric value below 0, the NA symbols are plotted below the cells. Otherwise if values >=0, they're plotted above the cells. Note that this argument is FALSE if 'subgroups' is NULL
min.cluster.size	labels for all groups with number of cells fewer than this parameter are considered as missed (default=0). This argument is ignored if groups aren't provided
mark.groups	plot cluster labels above points (default=TRUE)
show.legend	show legend (default=FALSE)
alpha	opacity level [0, 1] (default=0.4)
size	point size (default=0.8)
title	plot title (default=NULL)
plot.theme	theme for the plot (default=NULL)
palette	vector or list or function (default=NULL). Accepts number of colors and return list of colors (i.e. see 'colorRampPalette') (default=NULL)
color.range	controls range, in which colors are estimated (default="symmetric"). Pass "all" to estimate range based on all values of "colors", pass "data" to estimate it only based on colors, presented in the embedding. Alternatively you can pass vector of length 2 with (min, max) values.
font.size	font size for cluster labels (default=c(3, 7)). It can either be single number for constant font size or pair (min, max) for font size depending on cluster size
show.ticks	show ticks and tick labels (default=FALSE)
show.labels	show labels (default=FALSE)
legend.position	vector with (x, y) positions of the legend (default=NULL)
legend.title	legend title (default=NULL)

`gradient.range.quantile`
: Winsorization quantile for the numeric colors and gene gradient (default=1)

`raster`
: boolean whether layer with the points be rasterized (default=FALSE). Setting of this argument to TRUE is useful when you need to export a plot with large number of points

`raster.dpi`
: dpi of the rasterized plot. (default=300). Ignored if raster == FALSE.

`shuffle.colors`
: shuffle colors (default=FALSE)

`keep.limits`
: Keep axis limits from original plot (default=!is.null(subgroups)). Useful when plotting subgroups, only meaningful it plot.na=FALSE

`...`
: Arguments passed on to `ggrepel::geom_label_repel`

  `mapping` Set of aesthetic mappings created by `aes` or `aes_`. If specified and `inherit.aes = TRUE` (the default), is combined with the default mapping at the top level of the plot. You only need to supply `mapping` if there isn't a mapping defined for the plot.

  `data` A data frame. If specified, overrides the default data frame defined at the top level of the plot.

  `stat` The statistical transformation to use on the data for this layer, as a string.

  `position` Position adjustment, either as a string, or the result of a call to a position adjustment function.

  `parse` If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath

  `box.padding` Amount of padding around bounding box, as unit or number. Defaults to 0.25. (Default unit is lines, but other units can be specified by passing `unit(x, "units")`).

  `label.padding` Amount of padding around label, as unit or number. Defaults to 0.25. (Default unit is lines, but other units can be specified by passing `unit(x, "units")`).

  `point.padding` Amount of padding around labeled point, as unit or number. Defaults to 0. (Default unit is lines, but other units can be specified by passing `unit(x, "units")`).

  `label.r` Radius of rounded corners, as unit or number. Defaults to 0.15. (Default unit is lines, but other units can be specified by passing `unit(x, "units")`).

  `label.size` Size of label border, in mm.

  `min.segment.length` Skip drawing segments shorter than this, as unit or number. Defaults to 0.5. (Default unit is lines, but other units can be specified by passing `unit(x, "units")`).

  `arrow` specification for arrow heads, as created by `arrow`

  `force` Force of repulsion between overlapping text labels. Defaults to 1.

  `force_pull` Force of attraction between a text label and its corresponding data point. Defaults to 1.

  `max.time` Maximum number of seconds to try to resolve overlaps. Defaults to 0.5.

  `max.iter` Maximum number of iterations to try to resolve overlaps. Defaults to 10000.

`max.overlaps` Exclude text labels that overlap too many things. Defaults to 10.

`nudge_x, nudge_y` Horizontal and vertical adjustments to nudge the starting position of each text label. The units for `nudge_x` and `nudge_y` are the same as for the data units on the x-axis and y-axis.

`xlim, ylim` Limits for the x and y axes. Text labels will be constrained to these limits. By default, text labels are constrained to the entire plot area.

`na.rm` If `FALSE` (the default), removes missing values with a warning. If `TRUE` silently removes missing values.

`direction` "both", "x", or "y" – direction in which to adjust position of labels

`seed` Random seed passed to `set.seed`. Defaults to `NA`, which means that `set.seed` will not be called.

`verbose` If `TRUE`, some diagnostics of the repel algorithm are printed

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders`.

Value

ggplot2 object

Examples

```
library(sccore)
embeddingPlot(umapEmbedding, show.ticks=TRUE, show.labels=TRUE, title="UMAP embedding")
```

`embedGraphUmap` *Embed a graph into a UMAP, Uniform Manifold Approximation and Projection for Dimension Reduction, <https://github.com/lmcinnes/umap>, <doi:10.21105/joss.00861>*

Description

Embed a graph into a UMAP, Uniform Manifold Approximation and Projection for Dimension Reduction, <https://github.com/lmcinnes/umap>, <doi:10.21105/joss.00861>

Usage

```
embedGraphUmap(
  graph,
  min.prob = 0.001,
  min.visited.verts = 1000,
  n.cores = 1,
  max.hitting.nn.num = 0,
```

```
  max.commute.nn.num = 0,
  min.prob.lower = 1e-07,
  n.neighbors = 40,
  n.epochs = 1000,
  spread = 15,
  min.dist = 0.001,
  return.all = FALSE,
  n.sgd.cores = n.cores,
  verbose = TRUE,
  ...
)
```

Arguments

graph	input igraph object
min.prob	numeric Minimum probability for proximity when calculating hitting time per neighbors (default=1e-3)
min.visited.verts	numeric Minimum number of vertices visted when calculating hitting time per neighbors (default=1000)
n.cores	numeric Number of cores to use (default=1)
max.hitting.nn.num	numeric Maximum adjacencies for calculating hitting time per neighbor, hitting_time_per_neighbors() (default=0)
max.commute.nn.num	numeric Maximum adjacencies for calculating commute time per neighbor, commute_time_per_node() (default=0)
min.prob.lower	numeric Probability threshold to continue iteration in depth first search hitting time, dfs_hitting_time() (default=1e-7)
n.neighbors	numeric Number of neighbors (default=40)
n.epochs	numeric Number of epochs to use during the optimization of the embedded coordinates (default=1000). See 'n_epochs' in uwot::umap()
spread	numeric The effective scale of embedded points (numeric default=15). See 'spread' in uwot::umap()
min.dist	numeric The effective minimum distance between embedded points (default=0.001). See 'min.dist' in uwot::umap()
return.all	boolean If TRUE, return list(adj.info=adj.info, commute.times=commute.times, umap=umap). Otherwise, just return UMAP(default=FALSE)
n.sgd.cores	numeric Number of cores to use during stochastic gradient descent. If set to > 1, then results will not be reproducible, even if 'set.seed' is called with a fixed seed before running (default=n_threads) See 'n_sgd_threads' in uwot::umap()
verbose	boolean Verbose output (default=TRUE)
...	Additional arguments passed to embedKnnGraph()

Value

resulting UMAP embedding

`embedKnnGraph` *Embed a k-nearest neighbor (kNN) graph within a UMAP. Used within embedGraphUmap(). Please see McInnes et al <doi:10.21105/joss.00861> for the UMAP description and implementation.*

Description

Embed a k-nearest neighbor (kNN) graph within a UMAP. Used within embedGraphUmap(). Please see McInnes et al <doi:10.21105/joss.00861> for the UMAP description and implementation.

Usage

```
embedKnnGraph(
  commute.times,
  n.neighbors,
  names = NULL,
  n.cores = 1,
  n.epochs = 1000,
  spread = 15,
  min.dist = 0.001,
  n.sgd.cores = n.cores,
  target.dims = 2,
  verbose = TRUE,
  ...
)
```

Arguments

`commute.times`	graph commute times from get_nearest_neighbors(). The definition of commute_time(u, v) is the expected time starting at u = to reach v and then return to u .
`n.neighbors`	numeric Number of neighbors
`names`	vector of names for UMAP rownames (default=NULL)
`n.cores`	numeric Number of cores to use (except during stochastic gradient descent) (default=1). See 'n_threads' in uwot::umap()
`n.epochs`	numeric Number of epochs to use during the optimization of the embedded coordinates (default=1000). See 'n_epochs' in uwot::umap()
`spread`	numeric The effective scale of embedded points (numeric default=15). See 'spread' in uwot::umap()
`min.dist`	numeric The effective minimum distance between embedded points (default=0.001). See 'min.dist' in uwot::umap()

n.sgd.cores	numeric Number of cores to use during stochastic gradient descent. If set to > 1, then results will not be reproducible, even if 'set.seed' is called with a fixed seed before running (default=n.cores) See 'n_sgd_threads' in uwot::umap()
target.dims	numeric Dimensions for 'n_components' in uwot::umap(n_components=target.dims) (default=2)
verbose	boolean Verbose output (default=TRUE)
...	arguments passed to uwot::umap()

Value

resulting kNN graph embedding within a UMAP

`extendMatrix` *Extend matrix to include new columns in matrix*

Description

Extend matrix to include new columns in matrix

Usage

```
extendMatrix(mtx, col.names)
```

Arguments

mtx	Matrix
col.names	Columns that should be included in matrix

Value

Matrix with new columns but rows retained

Examples

```
library(dplyr)
gene.union <- lapply(conosClusterList, colnames) %>% Reduce(union, .)
extendMatrix(conosClusterList[[1]], col.names=gene.union)
```

fac2col *Utility function to translate a factor into colors*

Description

Utility function to translate a factor into colors

Usage

```
fac2col(
  x,
  s = 1,
  v = 1,
  shuffle = FALSE,
  min.group.size = 1,
  return.details = FALSE,
  unclassified.cell.color = "gray50",
  level.colors = NULL
)
```

Arguments

x	input factor
s	numeric The "saturation" to be used to complete the HSV color descriptions (default=1) See ?rainbow in Palettes, grDevices
v	numeric The "value" to be used to complete the HSV color descriptions (default=1) See ?rainbow in Palettes, grDevices
shuffle	boolean If TRUE, shuffles columns with shuffle(columns) (default=FALSE)
min.group.size	integer Exclude groups of size less than the min.group.size (default=1)
return.details	boolean If TRUE, returns a list list(colors=y, palette=col). Otherwise, just returns the factor (default=FALSE)
unclassified.cell.color	Color for unclassified cells (default='gray50')
level.colors	(default=NULL)

Value

vector or list of colors

Examples

```
genes = factor(c("BRAF", "NPC1", "PAX3", "BRCA2", "FMR1"))
fac2col(genes)
```

fac2palette *Encodes logic of how to handle named-vector and functional palettes. Used primarily within embeddingGroupPlot()*

Description

Encodes logic of how to handle named-vector and functional palettes. Used primarily within embeddingGroupPlot()

Usage

```
fac2palette(groups, palette, unclassified.cell.color = "gray50")
```

Arguments

groups	vector of cluster labels, names contain cell names
palette	vector or list or function (default=NULL). Accepts number of colors and return list of colors (i.e. see 'colorRampPalette')
unclassified.cell.color	Color for unclassified cells (default='gray50')

Value

vector or palette

getClusterGraph *Collapse vertices belonging to each cluster in a graph*

Description

Collapse vertices belonging to each cluster in a graph

Usage

```
getClusterGraph(
  graph,
  groups,
  method = "sum",
  plot = FALSE,
  node.scale = 50,
  edge.scale = 50,
  edge.alpha = 0.3,
  seed = 1,
  ...
)
```

Arguments

graph	igraph graph object Graph to be collapsed
groups	factor on vertices describing cluster assignment (can specify integer vertex ids, or character vertex names which will be matched)
method	string Method to be used, either "sum" or "paga" (default="sum")
plot	boolean Whether to show collapsed graph plot (default=FALSE)
node.scale	numeric Scaling to control value of 'vertex.size' in plot.igraph() (default=50)
edge.scale	numeric Scaling to control value of 'edge.width' in plot.igraph() (default=50)
edge.alpha	numeric Scaling to control value of 'alpha.f' in adjustcolor() within plot.igraph() (default=0.3)
seed	numeric Set seed via set.seed() for plotting (default=1)
...	arguments passed to collapseGraphSum()

Value

collapsed graph

Examples

```
cluster.graph = getClusterGraph(conosGraph, igraph::V(conosGraph))
```

---

get_nearest_neighbors *Get nearest neighbors method on graph*

Description

Get nearest neighbors method on graph

Usage

```
get_nearest_neighbors(
  adjacency_list,
  transition_probabilities,
  n_verts = 0L,
  n_cores = 1L,
  min_prob = 0.001,
  min_visited_verts = 1000L,
  min_prob_lower = 1e-05,
  max_hitting_nn_num = 0L,
  max_commute_nn_num = 0L,
  verbose = TRUE
)
```

Arguments

adjacency_list	igraph adjacency list
transition_probabilities	vector of transition probabilites
n_verts	numeric Number of vertices (default=0)
n_cores	numeric Number of cores to use (default=1)
min_prob	numeric Minimum probability for proximity when calculating hitting time per neighbors (default=1e-3)
min_visited_verts	numeric Minimum number of vertices visted when calculating hitting time per neighbors (default=1000)
min_prob_lower	numeric Probability threshold to continue iteration in depth first search hitting time, dfs_hitting_time() (default=1e-5)
max_hitting_nn_num	numeric Maximum adjacencies for calculating hitting time per neighbor, hitting_time_per_neighbors() (default=0)
max_commute_nn_num	numeric Maximum adjacencies for calculating commute time per neighbor, commute_time_per_node() (default=0)
verbose	boolean Whether to have verbose output (default=TRUE)

Value

list of commute times based on adjacencies

---

graphToAdjList *Convert igraph graph into an adjacency list*

Description

Convert igraph graph into an adjacency list

Usage

```
graphToAdjList(graph)
```

Arguments

graph	input igraph object

Value

adjacency list, defined by list(idx=adj.list, probabilities=probs, names=edge.list.fact$levels

Examples

```
library(dplyr)
edge.list.fact <- igraph::as_edgelist(conosGraph) %>% as_factor()
edge.list <- matrix(edge.list.fact$values, ncol=2)
n.nodes <- length(igraph::V(conosGraph))
splitVectorByNodes(edge.list[,1], edge.list[,2], n.nodes)
```

`heatFilter` *Graph filter with the heat kernel:* $f(x) = exp(-\beta|x/\lambda\_m - a|\hat{}b)$

Description

Graph filter with the heat kernel: $f(x) = exp(-\beta|x/\lambda_m - a|^b)$

Usage

```
heatFilter(x, l.max, order = 1, offset = 0, beta = 30)
```

Arguments

`x`	numeric Values to be filtered. Normally, these are graph laplacian engenvalues.
`l.max`	numeric Maximum eigenvalue on the graph (λ_m in the equation)
`order`	numeric Parameter b in the equation. Larger values correspond to the sharper kernel form (default=1). The values should be positive.
`offset`	numeric Mean kernel value (a in the equation), must be in [0:1] (default=0)
`beta`	numeric Parameter β in the equation. Larger values provide stronger smoothing. $\beta = 0$ corresponds to no smoothing (default=30).

Value

smoothed values for 'x'

See Also

Other graph smoothing: `computeChebyshevCoeffs()`, `smoothChebyshev()`, `smoothSignalOnGraph()`

`jsDist` *Jensen–Shannon distance metric (i.e. the square root of the Jensen–Shannon divergence) between the columns of a dense matrix m*

Description

Jensen–Shannon distance metric (i.e. the square root of the Jensen–Shannon divergence) between the columns of a dense matrix m

Usage

```
jsDist(m)
```

Arguments

`m`	Input matrix

Value

Vectorized version of the lower triangle as an R distance object, stats::dist()

Examples

```
ex = matrix(1:9, nrow = 3, ncol = 3)
jsDist(ex)
```

`mergeCountMatrices` *Merge list of count matrices into a common matrix, entering 0s for the missing entries*

Description

Merge list of count matrices into a common matrix, entering 0s for the missing entries

Usage

```
mergeCountMatrices(cms, transposed = FALSE, ...)
```

Arguments

`cms`	List of count matrices
`transposed`	boolean Indicate whether 'cms' is transposed, e.g. cells in rows and genes in columns (default=FALSE)
`...`	Parameters for 'plapply' function

Value

A merged extended matrix, with 0s for missing entries

Examples

```
mergeCountMatrices(conosClusterList, n.cores=1)
## 12 x 67388 sparse Matrix of class "dgCMatrix"
```

`multi2dend` *Translate multilevel segmentation into a dendrogram, with the lowest level of the dendrogram listing the cells*

Description

Translate multilevel segmentation into a dendrogram, with the lowest level of the dendrogram listing the cells

Usage

```
multi2dend(cl, counts, deep = FALSE, dist = "cor")
```

Arguments

`cl`	igraph communities object, returned from igraph community detection functions
`counts`	dgCmatrix of counts
`deep`	boolean If TRUE, take (cl$memberships[1,]). Otherwise, uses as.integer(membership(cl)) (default=FALSE)
`dist`	Distance metric used (default='cor'). Eiether 'cor' for the correlation distance in log10 space, or 'JS' for the Jensen–Shannon distance metric (i.e. the square root of the Jensen–Shannon divergence)

Value

resulting dendrogram

plapply *Parallel, optionally verbose lapply. See ?parallel::mclapply for more info.*

Description

Parallel, optionally verbose lapply. See ?parallel::mclapply for more info.

Usage

```
plapply(
  ...,
  progress = FALSE,
  n.cores = parallel::detectCores(),
  mc.preschedule = FALSE,
  mc.allow.recursive = TRUE,
  fail.on.error = FALSE
)
```

Arguments

`...`	Additional arguments passed to mclapply(), lapply(), or pbmcapply::pbmclapply()
`progress`	Show progress bar via pbmcapply::pbmclapply() (default=FALSE).
`n.cores`	Number of cores to use (default=parallel::detectCores()). When n.cores=1, regular lapply() is used. Note: doesn't work when progress=TRUE
`mc.preschedule`	if set to `TRUE` then the computation is first divided to (at most) as many jobs are there are cores and then the jobs are started, each job possibly covering more than one value. If set to `FALSE` then one job is forked for each value of `X`. The former is better for short computations or large number of values in `X`, the latter is better for jobs that have high variance of completion time and not too many values of `X` compared to `mc.cores`.
`mc.allow.recursive`	boolean Unless true, calling mclapply in a child process will use the child and not fork again (default=TRUE)
`fail.on.error`	boolean Whether to fail and report and error (using stop()) as long as any of the individual tasks has failed (default =FALSE)

Value

list, as returned by lapply

Examples

```
square = function(x){ x**2 }
plapply(1:10, square, n.cores=1, progress=TRUE)
```

`propagateLabels` *Estimate labeling distribution for each vertex, based on provided labels.*

Description

Estimate labeling distribution for each vertex, based on provided labels.

Usage

```
propagateLabels(graph, labels, method = "diffusion", ...)
```

Arguments

`graph`	igraph graph object
`labels`	vector of factor or character labels, named by cell names, used in propagateLabelsSolver() and propagateLabelsDiffusion()
`method`	string Type of propagation. Either 'diffusion' or 'solver'. (default='diffusion') 'solver' gives better result but has bad asymptotics, so it is inappropriate for datasets > 20k cells.
`...`	additional arguments passed to either propagateLabelsSolver() or propagateLabelsDiffusion()

Value

matrix with distribution of label probabilities for each vertex by rows.

Examples

```
propagateLabels(conosGraph, labels=cellAnnotations)
```

`propagateLabelsDiffusion` *Estimate labeling distribution for each vertex, based on provided labels using a Random Walk on graph*

Description

Estimate labeling distribution for each vertex, based on provided labels using a Random Walk on graph

Usage

```
propagateLabelsDiffusion(
  graph,
  labels,
  max.iters = 100,
  diffusion.fading = 10,
  diffusion.fading.const = 0.1,
  tol = 0.025,
  fixed.initial.labels = TRUE,
  verbose = TRUE
)
```

Arguments

graph	igraph graph object Graph input
labels	vector of factor or character labels, named by cell names
max.iters	integer Maximal number of iterations (default=100)
diffusion.fading	numeric Constant used for diffusion on the graph, exp(-diffusion.fading * (edge_length + diffusion.fading.const)) (default=10.0)
diffusion.fading.const	numeric Another constant used for diffusion on the graph, exp(-diffusion.fading * (edge_length + diffusion.fading.const)) (default=0.1)
tol	numeric Absolute tolerance as a stopping criteria (default=0.025)
fixed.initial.labels	boolean Prohibit changes of initial labels during diffusion (default=TRUE)
verbose	boolean Verbose mode (default=TRUE)

Value

matrix from input graph, with labels propagated

Examples

```
propagateLabelsDiffusion(conosGraph, labels=cellAnnotations)
```

---

propagateLabelsSolver *Propagate labels using Zhu, Ghahramani, Lafferty (2003) algorithm, "Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions" <http://mlg.eng.cam.ac.uk/zoubin/papers/zgl.pdf>*

Description

Propagate labels using Zhu, Ghahramani, Lafferty (2003) algorithm, "Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions" <http://mlg.eng.cam.ac.uk/zoubin/papers/zgl.pdf>

Usage

```
propagateLabelsSolver(graph, labels, solver = "mumps")
```

Arguments

graph	igraph graph object Graph input
labels	vector of factor or character labels, named by cell names
solver	Method of solver to use (default="mumps"), either "Matrix" or "mumps" (i.e. "rmumps::Rmumps")

Value

result from Matrix::solve() or rmumps::Rmumps

Examples

```
propagateLabelsSolver(conosGraph, labels=cellAnnotations)
```

---

propagate_labels *Label propagation*

Description

Label propagation

Usage

```
propagate_labels(
  edge_verts,
  edge_weights,
  vert_labels,
  max_n_iters = 10L,
  verbose = TRUE,
  diffusion_fading = 10,
  diffusion_fading_const = 0.5,
  tol = 0.005,
  fixed_initial_labels = FALSE
)
```

Arguments

edge_verts	edge vertices of igraph graph object
edge_weights	edge weights of igraph graph object
vert_labels	vector of factor or character labels, named by cell names
max_n_iters	integer Maximal number of iterations (default=10)
verbose	boolean Verbose mode (default=TRUE)
diffusion_fading	numeric Constant used for diffusion on the graph, exp(-diffusion.fading * (edge_length + diffusion.fading.const)) (default=10.0)
diffusion_fading_const	numeric Another constant used for diffusion on the graph, exp(-diffusion.fading * (edge_length + diffusion.fading.const)) (default=0.5)
tol	numeric Absolute tolerance as a stopping criteria (default=5e-3)
fixed_initial_labels	boolean Prohibit changes of initial labels during diffusion (default=FALSE)

Value

matrix from input graph, with labels propagated

---

saveDeAsJson *Save DE results as JSON tables for viewing in browser*

Description

Save DE results as JSON tables for viewing in browser

Usage

```
saveDeAsJson(
  de.raw,
  sample.groups = NULL,
  saveprefix = NULL,
  dir.name = "JSON",
  gene.metadata = NULL,
  verbose = TRUE
)
```

Arguments

de.raw	List of DE results from e.g. cacoa, conos
sample.groups	Sample groups as named list, each element containing a vector of samples. Can be retrieved from e.g. package cacoa (default=NULL)
saveprefix	Prefix for created files (default=NULL)

dir.name	Name for directory with results. If it doesn't exist, it will be created. To disable, set as NULL (default="JSON")
gene.metadata	(default=NULL) # Needs explanation
verbose	Show progress (default=TRUE)

Value

JSON files, table of content, and viewer files for viewing DE results in browser

Examples

```
## Not run:
saveDeAsJson(de.raw, sample.groups)

## End(Not run)
## The results can be viewed in a webbrowser by opening toc.html
```

---

setMinMax *Set range for values in object. Changes values outside of range to min or max. Adapted from Seurat::MinMax*

Description

Set range for values in object. Changes values outside of range to min or max. Adapted from Seurat::MinMax

Usage

```
setMinMax(obj, min, max)
```

Arguments

obj	Object to manipulate
min	Minimum value
max	Maximum value

Value

An object with the same dimensions as input but with altered range in values

Examples

```
example_matrix =  matrix(rep(c(1:5), 3), 5)
setMinMax(example_matrix, 2, 4)
```

`smoothSignalOnGraph` *Smooth Signal on Graph*

Description

Smooth Signal on Graph

Usage

```
smoothSignalOnGraph(
  signal,
  filter,
  graph = NULL,
  lap = NULL,
  l.max = NULL,
  m = 50,
  ...
)
```

Arguments

`signal`	signal to be smoothed
`filter`	function that accepts signal 'x' and the maximal Laplacian eigenvalue 'l.max'. See `heatFilter` as an example.
`graph`	igraph object with the graph (default=NULL)
`lap`	graph laplacian (default=NULL). If NULL, 'lap' estimated from graph.
`l.max`	maximal eigenvalue of 'lap' (default=NULL). If NULL, estimated from 'lap'.
`m`	numeric Maximum order of Chebyshev coeff to compute (default=50)
`...`	Arguments passed on to `smoothChebyshev` `n.cores` numeric Number of cores for parallel run (default=1) `progress.chunks` numeric Number of chunks per core for estimating progress (default=5). Large values are not suggested, as it may bring overhead. `progress` boolean Flag on whether progress must be shown (default=TRUE, i.e. 'progress.chunks > 1')

See Also

Other graph smoothing: `computeChebyshevCoeffs()`, `heatFilter()`, `smoothChebyshev()`

`smooth_count_matrix` *Smooth gene expression, used primarily within conos::correctGenes. Used to smooth gene expression values in order to better represent the graph structure. Use diffusion of expression on graph with the equation dv = exp(-a * (v + b))*

Description

Smooth gene expression, used primarily within conos::correctGenes. Used to smooth gene expression values in order to better represent the graph structure. Use diffusion of expression on graph with the equation dv = exp(-a * (v + b))

Usage

```
smooth_count_matrix(
  edge_verts,
  edge_weights,
  count_matrix,
  is_label_fixed,
  max_n_iters = 10L,
  diffusion_fading = 1,
  diffusion_fading_const = 0.1,
  tol = 0.001,
  verbose = TRUE,
  normalize = FALSE
)
```

Arguments

`edge_verts`	edge vertices of igraph graph object
`edge_weights`	edge weights of igraph graph object
`count_matrix`	gene count matrix
`is_label_fixed`	boolean Whether label is fixed
`max_n_iters`	integer Maximal number of iterations (default=10)
`diffusion_fading`	numeric Constant used for diffusion on the graph, exp(-diffusion.fading * (edge_length + diffusion.fading.const)) (default=1.0)
`diffusion_fading_const`	numeric Another constant used for diffusion on the graph, exp(-diffusion.fading * (edge_length + diffusion.fading.const)) (default=0.1)
`tol`	numeric Absolute tolerance as a stopping criteria (default=1e-3)
`verbose`	boolean Verbose mode (default=TRUE)
`normalize`	boolean Whether to normalize values (default=FALSE)

Value

matrix from input graph, with labels propagated

---

sn *Set names equal to values, a stats::setNames wrapper function*

Description

Set names equal to values, a stats::setNames wrapper function

Usage

```
sn(x)
```

Arguments

x	an object for which names attribute will be meaningful

Value

An object with names assigned equal to values

Examples

```
vec = c(1, 2, 3, 4)
sn(vec)
```

---

splitVectorByNodes *splitVectorByNodes*

Description

splitVectorByNodes

Usage

```
splitVectorByNodes(vec, nodes, n.nodes)
```

Arguments

vec	input vector to be divided
nodes	nodes used to divide the vector 'vec' via split()
n.nodes	numeric The number of nodes for splitting

Value

list from vec with names given by the nodes

Examples

```
adjList = graphToAdjList(conosGraph)
print(names(adjList))
## [1] "idx" "probabilities" "names"
length(adjList$names)
## [1] 12000
```

`styleEmbeddingPlot` *Set plot.theme, legend, ticks for embedding plot. Used primarily in embeddingPlot().*

Description

Set plot.theme, legend, ticks for embedding plot. Used primarily in embeddingPlot().

Usage

```
styleEmbeddingPlot(
  gg,
  plot.theme = NULL,
  title = NULL,
  legend.position = NULL,
  show.legend = TRUE,
  show.ticks = TRUE,
  show.labels = TRUE,
  relabel.axis = TRUE
)
```

Arguments

`gg`	ggplot2 object to plot
`plot.theme`	theme for the plot (default=NULL)
`title`	plot title (default=NULL)
`legend.position`	vector with (x, y) positions of the legend (default=NULL)
`show.legend`	show legend (default=TRUE)
`show.ticks`	show ticks and tick labels (default=TRUE)
`show.labels`	show labels (default=TRUE)
`relabel.axis`	boolean If TRUE, relabel axes with ggplot2::labs(x='Component 1', y='Component 2') (default=TRUE)

Value

ggplot2 object

`umapEmbedding` *UMAP embedding*

Description

UMAP embedding

Usage

```
umapEmbedding
```

Format

An object of class `matrix` (inherits from `array`) with 12000 rows and 2 columns.

`val2col` *Utility function to translate values into colors.*

Description

Utility function to translate values into colors.

Usage

```
val2col(x, gradientPalette = NULL, zlim = NULL, gradient.range.quantile = 0.95)
```

Arguments

`x`	input values
`gradientPalette`	gradient palette (default=NULL). If NULL, use colorRampPalette(c('gray90','red'), space = "Lab")(1024) if the values are non-negative; otherwise colorRampPalette(c("blue", "grey90", "red"), space = "Lab")(1024) is used
`zlim`	a two-value vector specifying limits of the values that should correspond to the extremes of the color gradient
`gradient.range.quantile`	extreme quantiles of values that should be trimmed prior to color mapping (default=0.95)

Examples

```
colors <- val2col( rnorm(10) )
```

`val2ggcol` *Helper function to return a ggplot color gradient for a numeric vector ggplot(aes(color=x, ...), ...) + val2ggcol(x)*

Description

Helper function to return a ggplot color gradient for a numeric vector ggplot(aes(color=x, ...), ...) + val2ggcol(x)

Usage

```
val2ggcol(
  values,
  gradient.range.quantile = 1,
  color.range = "symmetric",
  palette = NULL,
  midpoint = NULL,
  oob = scales::squish,
  return.fill = FALSE,
  ...
)
```

Arguments

`values`	values by which the color gradient is determined
`gradient.range.quantile`	numeric Trimming quantile (default=1). Either a single number or two numbers - for lower and upper quantile.
`color.range`	either a vector of two values explicitly specifying the values corresponding to the start/end of the gradient, or string "symmetric" or "all" (default="symmetric"). "symmetric": range will fit data, but will be symmetrized around zeros, "all": gradient will match the span of the range of the data (after gradient.range.quantile)
`palette`	an optional palette (default=NULL). The default becomes blue-gray90-red; if the values do not straddle 0, then truncated gradients (blue-gray90 or gray90-red) will be used
`midpoint`	optional midpoint (default=NULL). Set for the center of the resulting range by default
`oob`	function to determine what to do with the values outside of the range (default =scales::squish). Refer to 'oob' parameter in ggplot
`return.fill`	boolean Whether to return fill gradients instead of color (default=FALSE)
`...`	additional arguments are passed to ggplot2::scale_color_gradient* functions, i.e. scale_color_gradient(), scale_color_gradient2(), scale_color_gradientn()

Value

ggplot2::scale_colour_gradient object

Index