Package 'scProkaR'

November 7, 2026

Type Package

Title Microbial Single-Cell RNA-Seq Analysis for Bacterial Systems

Version 0.99.3

Description scProkaR provides a SingleCellExperiment-centred workflow for microbial single-cell RNA sequencing data. It standardises bacterial input objects from count matrices, 10x Genomics directories or Seurat objects, and computes prokaryote-specific quality-control metrics such as ribosomal RNA fraction. It performs batch integration with several backends and scores the resulting embeddings with scIB-inspired benchmarking metrics. It also supports pseudobulk differential expression with edgeR and Time Aware Trajectory Analysis (TATA), a time-informed trajectory abstraction for perturbation time courses.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

Depends R (>= 4.6.0)

Imports bluster, edgeR, ggplot2, grDevices, grid, igraph, Matrix, methods, S4Vectors, SingleCellExperiment, splines, stats, SummarizedExperiment, utils

Suggests batchelor, BiocNeighbors, BiocSingular, BiocStyle, cluster, DropletUtils, FNN, harmony, irlba, knitr, RANN, rmarkdown, scrapper (>= 1.0.0), SeuratObject, testthat (>= 3.0.0), uwot

VignetteBuilder knitr

Config/testthat/edition 3

biocViews Software, SingleCell, RNASeq, Transcriptomics, GeneExpression, DifferentialExpression, Clustering, GraphAndNetwork, QualityControl, MultipleComparison, Normalization, BatchEffect, DimensionReduction, TimeCourse, Microbiome, Sequencing, Visualization

LazyData false

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL https://github.com/Monash-Li-Lab/scProkaR

BugReports https://github.com/Monash-Li-Lab/scProkaR/issues

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

git_branch devel

git_last_commit ecce4be

git_last_commit_date 2026-10-28

Repository Bioconductor 3.24

Date/Publication 2026-11-06

Author Jinxin Zhao [aut, cre] (ORCID: <https://orcid.org/0000-0002-0604-3541>),
Ning Liu [aut],
Alexander von Humboldt Foundation [fnd] (Fellowship support for Jinxin Zhao)

Maintainer Jinxin Zhao <jinxin.zhao@monash.edu>

Contents

scProkaR-package . . . 3
adjacent_timepoint_auc . . . 4
aggregate_pseudobulk . . . 5
anticausal_edge_mass . . . 6
BenchmarkIntegration . . . 8
branch_onset_metrics . . . 9
build_knn_graph . . . 10
build_tata_graph . . . 11
cluster_graph_states . . . 12
cluster_tata_gene_trends . . . 13
compute_tata_branch_probabilities . . . 14
compute_tata_gene_trends . . . 15
compute_tata_pseudotime . . . 16
compute_time_weights . . . 17
compute_topology_weights . . . 18
CreateBacObject . . . 19
directed_reachability_concordance . . . 21
FilterBacCells . . . 22
filter_pseudobulk_samples . . . 23
IntegrateBacData . . . 24
local_temporal_order_metrics . . . 26
MergeBacObjects . . . 27
normalize_pseudobulk . . . 28
orient_pseudotime_to_time . . . 29
PlotIntegrationOverview . . . 30
PlotReduction . . . 31
plot_pairwise_de_ma . . . 32
plot_pairwise_de_volcano . . . 34
plot_tata_branch_confusion . . . 35
plot_tata_branch_probabilities . . . 36
plot_tata_branch_selection . . . 37
plot_tata_cluster_graph . . . 38
plot_tata_edge_causality . . . 40
plot_tata_gene_trends . . . 41
plot_tata_gene_trend_heatmap . . . 41

plot_tata_results 42
plot_tata_terminal_probabilities 43
plot_tata_time_calibration 44
plot_tata_trajectory_embedding 45
plot_time_series_deg_curves 46
RegisterExistingIntegrationEmbeddings 48
RegisterIntegrationEmbedding 49
RunBacQC 50
RunIntegratedClustering 51
RunIntegratedUMAP 52
run_edger_pairwise_de 53
run_edger_spline_de 55
run_scproka_de 57
run_tata 58
sce1 60
select_tata_branch_cells 61
simulate_tata_multidrug_sce 62
simulate_tata_sce 63
simulate_tata_spiral_sce 64

Index **65**

`scProkaR-package` *scProkaR: Microbial Single-Cell RNA-Seq Analysis for Bacterial Systems*

Description

`scProkaR` provides a `SingleCellExperiment`-centered toolkit for microbial single-cell analysis in R. The package combines bacterial QC and integration workflows with pseudobulk differential expression utilities and Time Aware Trajectory Analysis (TATA) for time-informed trajectory abstraction.

Example dataset

The package includes `sce1`, a small `SingleCellExperiment` object derived from a three-treatment bacterial single-cell experiment. The object contains raw counts together with `sample`, `clusters`, `treatments`, and `timepoints` metadata, and is used throughout the package workflow vignette.

Author(s)

Maintainer: Jinxin Zhao <jinxin.zhao@monash.edu> (ORCID)

Authors:

- Ning Liu <ning.liu@adelaide.edu.au>

See Also

Useful links:

- https://github.com/Monash-Li-Lab/scProkaR
- Report bugs at https://github.com/Monash-Li-Lab/scProkaR/issues

`adjacent_timepoint_auc` *Compute adjacent-timepoint AUC for a pseudotime ordering*

Description

For each adjacent pair of experimental timepoints, compute the probability that a cell from the later timepoint has larger pseudotime than a cell from the earlier timepoint. This is equivalent to the Mann-Whitney AUC.

Usage

```
adjacent_timepoint_auc(
  pseudotime,
  timepoint,
  group = NULL,
  average = c("weighted", "macro"),
  min_cells = 10L
)
```

Arguments

`pseudotime`	Numeric vector of inferred pseudotime values.
`timepoint`	Experimental time labels for the same cells.
`group`	Optional grouping variable, for example treatment or condition. If provided, adjacent-timepoint AUC is computed within each group before averaging.
`average`	How to summarize the per-pair AUC values. Choices are: • `"weighted"`: weight each comparison by the number of earlier-later cell pairs. • `"macro"`: give every valid comparison equal weight.
`min_cells`	Minimum number of cells required in each timepoint group for a comparison to be included.

Value

A list with:

- `summary`: a one-row `data.frame` containing the overall adjacent-timepoint AUC and the number of valid comparisons.
- `pair_table`: a `data.frame` with AUC values for each adjacent timepoint comparison.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce, dimred = "PCA", time_col = "timepoint", k = 15)
```

```
oriented <- orient_pseudotime_to_time(
    pseudotime = tata$cell_pseudotime,
    timepoint = tata$sce$timepoint
)

# Pooled across all cells.
auc_all <- adjacent_timepoint_auc(
    pseudotime = oriented,
    timepoint = tata$sce$timepoint
)
auc_all$summary
head(auc_all$pair_table)

# Computed within each drug condition before averaging.
auc_by_drug <- adjacent_timepoint_auc(
    pseudotime = oriented,
    timepoint = tata$sce$timepoint,
    group = tata$sce$condition,
    min_cells = 5
)
auc_by_drug$summary
```

`aggregate_pseudobulk` *Aggregate single-cell data into pseudobulk samples*

Description

Summarize single-cell counts into pseudobulk samples defined by one or more metadata columns. This is a common first step before sample-level differential expression analysis with tools such as edgeR.

Usage

```
aggregate_pseudobulk(
  sce,
  sample_cols,
  assay_name = "counts",
  aggregation = c("sum", "mean"),
  min_cells = 1L,
  add_logcounts = TRUE,
  prior_count = 2,
  sample_prefix = "pb_"
)
```

Arguments

`sce`	A `SingleCellExperiment`.
`sample_cols`	Character vector of `colData(sce)` columns that together define each pseudobulk sample, for example `c("sample_id", "timepoint", "treatment", "cluster")`.
`assay_name`	Assay containing the input counts.
`aggregation`	Aggregation method. Choices are:

- "sum": sum counts across cells in each pseudobulk sample. This is the recommended default for count-based edgeR workflows.
- "mean": average expression across cells in each pseudobulk sample. This can be useful for exploratory summaries but is not ideal for count-based NB models.

min_cells	Minimum number of cells required for a pseudobulk sample to be retained.
add_logcounts	Logical indicating whether to add a logcounts assay using edgeR::cpm(..., log = TRUE).
prior_count	Prior count passed to edgeR::cpm() when add_logcounts = TRUE.
sample_prefix	Prefix for the generated pseudobulk column names.

Value

A pseudobulk SingleCellExperiment with one column per aggregated sample. The output colData includes the grouping metadata, ncells, and lib.size.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600,
    n_features = 60,
    n_pcs = 10
)

# Split cells into three pseudo-replicates per condition/timepoint.
strata <- interaction(sce$condition, sce$timepoint, drop = TRUE)
rep_id <- integer(ncol(sce))
for (level_name in levels(strata)) {
    idx <- which(strata == level_name)
    rep_id[idx] <- sample(rep(seq_len(3), length.out = length(idx)))
}
sce$replicate <- factor(rep_id)

pb <- aggregate_pseudobulk(
    sce,
    sample_cols = c("condition", "timepoint", "replicate"),
    aggregation = "sum",
    min_cells = 5
)
pb
head(as.data.frame(SummarizedExperiment::colData(pb)))
```

---

anticausal_edge_mass *Measure the weight of anti-causal edges in a directed graph*

Description

Compare each directed graph edge with the median experimental time of its source and target clusters, and quantify how much total edge weight points backward in time.

Usage

```
anticausal_edge_mass(
  cluster_graph,
  clusters,
  timepoint,
  tolerance = 0,
  weight_attr = "weight"
)
```

Arguments

cluster_graph	Directed cluster-level graph as an igraph.
clusters	Cluster label for each cell.
timepoint	Experimental time labels for the same cells.
tolerance	A non-negative tolerance for small median-time differences. Directed edges with time differences in [-tolerance, tolerance] are treated as time-neutral rather than anti-causal.
weight_attr	Edge attribute to use as the edge weight.

Value

A list with:

- summary: a one-row data.frame containing the total edge weight, the anti-causal edge mass, its proportion, and a higher-is-better causal edge score.
- edge_table: a data.frame describing each directed edge.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce, dimred = "PCA", time_col = "timepoint", k = 15)

causality <- anticausal_edge_mass(
    cluster_graph = tata$cluster_graph,
    clusters = tata$sce$cluster,
    timepoint = tata$sce$timepoint
)
causality$summary
table(causality$edge_table$temporal_class)

# Treat median-time gaps below one sampling interval as time-neutral.
anticausal_edge_mass(
    cluster_graph = tata$cluster_graph,
    clusters = tata$sce$cluster,
    timepoint = tata$sce$timepoint,
    tolerance = 24
)$summary
```

`BenchmarkIntegration` *Benchmark integration quality with scIB-inspired metrics*

Description

Computes a compact R-native metric panel across available reduced dimensions and summarizes the results with scores and optional plots.

Usage

```
BenchmarkIntegration(
  sce,
  batch_col,
  label_col = NULL,
  methods = NULL,
 metrics = c("pcr_batch", "ilisi", "clisi", "kbet_like", "graph_connectivity", "ari",
    "nmi", "silhouette"),
  return_plots = TRUE
)
```

Arguments

`sce`	A `SingleCellExperiment`.
`batch_col`	Column in `colData(sce)` containing batch labels.
`label_col`	Optional biological label column for conservation metrics.
`methods`	Optional vector of integration method names or reducedDim names to benchmark, matched without regard to case. When `NULL`, stored integration results and other reductions named `integrated_*` are used. If neither is present, all available reductions are used. Exact matches take precedence over case-insensitive matches.
`metrics`	Metric names to calculate.
`return_plots`	If `TRUE`, return summary ggplot objects.

Details

Any embedding already present in `reducedDims(sce)` can be benchmarked by passing its reduced-dimension name in `methods`, and external embeddings can be registered with `RegisterIntegrationEmbedding()` or `RegisterExistingIntegrationEmbeddings()`.

Value

A list with `scores`, `ranking`, and optional `plots`.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 300, n_features = 60, n_pcs = 10
)
pca <- SingleCellExperiment::reducedDim(sce, "PCA")
```

```
# Use the drug condition as batch and a quick clustering as the
# biological label to conserve.
sce$batch <- sce$condition
sce$cell_type <- factor(stats::kmeans(pca, centers = 3)$cluster)

# Register a second embedding so it is scored alongside the PCA.
sce <- RegisterIntegrationEmbedding(
    sce, pca[, seq_len(5)],
    method_name = "pca_top5"
)

bench <- BenchmarkIntegration(
    sce,
    batch_col = "batch",
    label_col = "cell_type",
    methods = c("PCA", "pca_top5"),
    return_plots = FALSE
)
head(bench$scores)
bench$ranking
```

`branch_onset_metrics` *Measure branch-emergence timing from pseudotime*

Description

Compare the known order or timing of branch emergence with the inferred pseudotime of sufficiently committed cells in each branch.

Usage

```
branch_onset_metrics(
  pseudotime,
  branch,
  branch_activation,
  true_onset,
  activation_threshold = 0.25
)
```

Arguments

`pseudotime`	Numeric vector of inferred pseudotime values.
`branch`	Branch or condition label for each cell.
`branch_activation`	Numeric activation score indicating how committed a cell is to its branch.
`true_onset`	Named numeric vector giving the known onset time of each branch. The names must match a subset of the values in `branch`.
`activation_threshold`	Minimum activation value required for a cell to be treated as branch-committed when estimating the branch onset.

Value

A list with:

- summary: a one-row data.frame containing Kendall tau, Spearman rho, normalized branch-onset mean absolute error, a higher-is-better branch onset order score, a higher-is-better branch onset accuracy score, and the number of branches used.
- onset_table: a data.frame with the true and inferred onset values for each branch.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce, dimred = "PCA", time_col = "timepoint", k = 15)

oriented <- orient_pseudotime_to_time(
    pseudotime = tata$cell_pseudotime,
    timepoint = tata$sce$timepoint
)

# The simulation activates drug_A first, then drug_C, then drug_B.
onset <- branch_onset_metrics(
    pseudotime = oriented,
    branch = tata$sce$simulated_branch,
    branch_activation = tata$sce$branch_activation,
    true_onset = c(drug_A = 24, drug_B = 96, drug_C = 48)
)
onset$summary
onset$onset_table
```

---

build_knn_graph *Build a cell-cell kNN graph*

Description

Build an undirected cell-cell k-nearest-neighbour graph from a low-dimensional representation.

Usage

```
build_knn_graph(x, dimred = "PCA", dims = NULL, k = 15)
```

Arguments

x	A SingleCellExperiment or a numeric matrix with cells in rows.
dimred	Name of the reduced dimension to use when x is a SingleCellExperiment.
dims	Optional subset of dimensions to use.
k	Number of neighbours.

Value

A list containing an igraph object, a sparse adjacency matrix, the embedding used, neighbour indices, neighbour distances, and kNN metadata.

Examples

```
sce <- simulate_tata_sce(n_cells = 1000, n_features = 60)
knn <- build_knn_graph(sce, dimred = "PCA", k = 10)
igraph::vcount(knn$graph)
```

---

build_tata_graph *Build the TATA cluster graph*

Description

Combine topology and temporal consistency into final TATA edge weights, assign edge direction, and prune weak edges to obtain the cluster-level TATA graph.

Usage

```
build_tata_graph(
  topology_table,
  time_table,
  clusters,
  embedding,
  timepoint,
  alpha = 1,
  beta = 1,
  direction_threshold = 0.1,
  prune_threshold = 0.01,
  time_weight_mode = c("directional_confidence", "ordered")
)
```

Arguments

topology_table	Output from compute_topology_weights().
time_table	Output from compute_time_weights().
clusters	Cluster labels for cells.
embedding	Low-dimensional embedding used to compute cluster centroids.
timepoint	Experimental time labels.
alpha	Exponent for topology weights.
beta	Exponent for time weights.
direction_threshold	Threshold for assigning edge direction.
prune_threshold	Minimum TATA weight to retain an edge.
time_weight_mode	How to incorporate time into the final edge weight. Choices are:

- "directional_confidence": reward strong directional evidence regardless of whether the supported direction is a -> b or b -> a.
- "ordered": use the original ordered-pair formulation (flow_score + 1) / 2, which is stricter but depends on the ordering of cluster_a and cluster_b.

Value

A list containing the directed cluster graph, the cluster edge table, and cluster vertex metadata.

Examples

```
set.seed(1)
sce <- simulate_tata_sce(n_cells = 1000, n_features = 60)
knn <- build_knn_graph(sce, dimred = "PCA", k = 12)
clusters <- cluster_graph_states(knn$graph, method = "louvain")
timepoint <- SummarizedExperiment::colData(sce)$timepoint
topology_table <- compute_topology_weights(knn$adjacency, clusters)
time_table <- compute_time_weights(knn$adjacency, clusters, timepoint)
tata_graph <- build_tata_graph(
    topology_table = topology_table,
    time_table = time_table,
    clusters = clusters,
    embedding = knn$embedding,
    timepoint = timepoint
)
igraph::ecount(tata_graph$graph)
head(tata_graph$vertex_table)
```

---

cluster_graph_states *Cluster graph-connected cell states*

Description

Cluster the cell graph into coarse states using a graph-based community detection method.

Usage

```
cluster_graph_states(cell_graph, method = c("louvain", "leiden", "walktrap"))
```

Arguments

cell_graph	An undirected cell-level igraph.
method	Clustering method. Choices are:

- "louvain": fast modularity-based community detection.
- "leiden": Leiden clustering if supported by the installed igraph.
- "walktrap": random-walk clustering that can produce broader partitions.

Value

A factor of inferred cluster labels.

Examples

```
sce <- simulate_tata_sce(n_cells = 1000, n_features = 60)
knn <- build_knn_graph(sce, dimred = "PCA", k = 10)
clusters <- cluster_graph_states(knn$graph, method = "louvain")
table(clusters)
```

---

cluster_tata_gene_trends

Cluster TATA gene trends by shape

Description

Cluster branch-specific fitted gene trends by shape using k-means.

Usage

```
cluster_tata_gene_trends(trend_result, n_clusters = 4)
```

Arguments

trend_result	Result list returned by compute_tata_gene_trends().
n_clusters	Number of trend clusters.

Value

A data frame mapping genes to trend clusters.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60, n_pcs = 10
)
tata <- run_tata(sce,
    dimred = "PCA", time_col = "timepoint", k = 15,
    expected_branches = 3
)
trends <- compute_tata_gene_trends(
    tata,
    genes = head(rownames(tata$sce), 12),
    grid_length = 30
)
set.seed(2)
trend_clusters <- cluster_tata_gene_trends(trends, n_clusters = 3)
head(trend_clusters)
table(trend_clusters$trend_cluster)
```

`compute_tata_branch_probabilities` *Compute TATA branch probabilities and entropy*

Description

Estimate terminal-branch absorption probabilities for clusters and cells on the TATA abstract graph, then derive an entropy-style plasticity score for each cell.

Usage

```
compute_tata_branch_probabilities(
  tata_result,
  terminal_clusters = NULL,
  assignment_threshold = 0.5,
  expected_branches = NULL,
  backward_penalty = 0.15
)
```

Arguments

`tata_result`	Result list returned by `run_tata()`.
`terminal_clusters`	Optional character vector of terminal clusters. If `NULL`, TATA identifies terminal clusters from the directed abstract graph and cluster pseudotime.
`assignment_threshold`	Minimum maximum branch probability required for a hard branch assignment. Cells below this threshold are labelled `"ambiguous"`.
`expected_branches`	Optional expected number of terminal branches. If supplied, TATA keeps up to that many high-pseudotime terminal candidates rather than relying only on automatically collapsed terminal states.
`backward_penalty`	Multiplicative penalty applied to transitions that move backward in cluster pseudotime when constructing the transition matrix.

Value

A list with terminal clusters, cluster-level branch probabilities, cell-level branch probabilities, and entropy/plasticity summaries.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60, n_pcs = 10
)
tata <- run_tata(sce,
    dimred = "PCA", time_col = "timepoint", k = 15,
    expected_branches = 3
)
```

```
branch_res <- compute_tata_branch_probabilities(
    tata,
    expected_branches = 3
)
branch_res$terminal_clusters
head(branch_res$cluster_branch_probabilities)
table(branch_res$branch_probabilities$tata_branch_assignment)
```

`compute_tata_gene_trends` *Compute branch-specific TATA gene trends*

Description

Fit smooth branch-weighted gene-expression trends along TATA pseudotime for one or more genes.

Usage

```
compute_tata_gene_trends(
  tata_result,
  genes,
  branches = NULL,
  assay_name = NULL,
  probability_threshold = 0.05,
  grid_length = 100,
  span = 0.5
)
```

Arguments

`tata_result`	Result list returned by `run_tata()`.
`genes`	Character vector of genes to model.
`branches`	Optional subset of terminal branches.
`assay_name`	Assay to use. If `NULL`, TATA prefers `logcounts` and falls back to `counts`.
`probability_threshold`	Minimum branch probability used to retain cells when fitting a branch-specific trend.
`grid_length`	Number of pseudotime grid points for fitted curves.
`span`	Smoothing span passed to `stats::loess()`.

Value

A list containing the fitted trend table, the selected branches, and the genes used.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60, n_pcs = 10
)
tata <- run_tata(sce,
    dimred = "PCA", time_col = "timepoint", k = 15,
    expected_branches = 3
)
trends <- compute_tata_gene_trends(
    tata,
    genes = head(rownames(tata$sce), 6),
    grid_length = 30
)
trends$branches
head(trends$trend_table)
with(trends$trend_table, tapply(fitted_expression, branch, range))
```

`compute_tata_pseudotime` *Compute TATA pseudotime*

Description

Compute a simple cluster-level pseudotime on the directed TATA graph using shortest-path distances from the root cluster with inverse edge weights, then propagate that pseudotime to cells using within-cluster time ranks. If some clusters are not reachable in the directed graph, TATA falls back to the undirected abstract topology for those clusters so late branches are not dropped simply because an intermediate direction is ambiguous.

Usage

```
compute_tata_pseudotime(
  cluster_graph,
  sce,
  cluster_col = "cluster",
  time_col = "timepoint",
  root_cluster = NULL,
  within_cluster_fraction = 0.5,
  eps = 1e-08
)
```

Arguments

`cluster_graph`	Directed cluster-level TATA graph.
`sce`	A `SingleCellExperiment`.
`cluster_col`	Column in `colData(sce)` containing inferred clusters.
`time_col`	Column in `colData(sce)` containing experimental time labels.
`root_cluster`	Optional root cluster. If `NULL`, the root is chosen as the cluster with the earliest median timepoint.

within_cluster_fraction
: Scale factor used to spread cells within each cluster after assigning cluster-level pseudotime.

eps
: Small constant to avoid division by zero.

Value

A list with root cluster, cluster pseudotime table, raw cell pseudotime, scaled cell pseudotime, and the within-cluster step size.

Examples

```
set.seed(1)
sce <- simulate_tata_sce(n_cells = 1000, n_features = 60)
tata <- run_tata(sce, dimred = "PCA", k = 15)

# Recompute pseudotime directly from the abstracted cluster graph.
pt <- compute_tata_pseudotime(
    cluster_graph = tata$cluster_graph,
    sce = tata$sce,
    cluster_col = "cluster",
    time_col = "timepoint"
)

pt$root_cluster
head(pt$cluster_pseudotime)
summary(pt$cell_pseudotime_scaled)
```

---

compute_time_weights *Compute time consistency for cluster-cluster edges*

Description

For each inter-cluster cell-cell graph edge, compute `sign(t_b - t_a)` after orienting the cluster pair as `(cluster_a, cluster_b)`. The mean sign is the temporal flow score.

Usage

```
compute_time_weights(adjacency, clusters, timepoint)
```

Arguments

adjacency
: Sparse cell-cell adjacency matrix.

clusters
: Cluster labels for cells.

timepoint
: Experimental time labels for cells.

Value

A data frame of temporal flow scores and time weights.

Examples

```
set.seed(1)
sce <- simulate_tata_sce(n_cells = 1000, n_features = 60)
knn <- build_knn_graph(sce, dimred = "PCA", k = 12)
clusters <- cluster_graph_states(knn$graph, method = "louvain")
time_table <- compute_time_weights(
    adjacency = knn$adjacency,
    clusters = clusters,
    timepoint = SummarizedExperiment::colData(sce)$timepoint
)
head(time_table[, c(
    "cluster_a", "cluster_b", "flow_score",
    "time_weight"
)])
```

`compute_topology_weights` *Compute topology-based cluster connectivity weights*

Description

For each cluster pair `(a, b)`, count observed inter-cluster edges and compare them to a simple degree-aware expectation.

Usage

```
compute_topology_weights(adjacency, clusters, eps = 1e-08)
```

Arguments

`adjacency`	Sparse cell-cell adjacency matrix.
`clusters`	Cluster labels for cells.
`eps`	Small constant to avoid division by zero.

Value

A data frame of observed edges, expected edges, and topology weights.

Examples

```
set.seed(1)
sce <- simulate_tata_sce(n_cells = 1000, n_features = 60)
knn <- build_knn_graph(sce, dimred = "PCA", k = 12)
clusters <- cluster_graph_states(knn$graph, method = "louvain")
topology_table <- compute_topology_weights(
    adjacency = knn$adjacency,
    clusters = clusters
)
head(topology_table[order(-topology_table$topology_weight), ])
```

CreateBacObject *Create a standardized bacterial single-cell object*

Description

CreateBacObject() standardizes raw microbial single-cell counts into a SingleCellExperiment, aligns cell and feature metadata, and records package provenance used by downstream scProkaR workflows.

Usage

```
CreateBacObject(
  x,
  counts_assay = "counts",
  cell_metadata = NULL,
  feature_metadata = NULL,
  feature_name_col = NULL,
  sample_col = NULL,
  batch_col = NULL,
  condition_col = NULL,
  time_col = NULL,
  sample_id_value = NULL,
  batch_value = NULL,
  condition_value = NULL,
  time_value = NULL,
  organism = "bacteria",
  seurat_assay = NULL,
  seurat_layer = "counts",
  transfer_reductions = TRUE,
  run_unintegrated = FALSE,
  unintegrated_feature_set = c("hvg", "all"),
  unintegrated_dims = seq_len(30),
  unintegrated_cluster_col = "unintegrated_clusters",
  unintegrated_umap_name = "umap.unintegrated",
  unintegrated_k = 20,
  unintegrated_resolution = 2,
  unintegrated_algorithm = c("louvain", "walktrap", "leiden")
)
```

Arguments

x	A gene-by-cell count matrix, a SingleCellExperiment, or a loaded Seurat object from an .rds file, or a Cell Ranger output directory.
counts_assay	Assay name to use when x is a SingleCellExperiment.
cell_metadata	Optional cell-level metadata with one row per cell.
feature_metadata	Optional feature-level metadata with one row per gene.
feature_name_col	Optional rowData column to use as the primary feature name when x is a SingleCellExperiment or 10x-derived object. If NULL, scProkaR will prefer common columns such as

Symbol, gene_name, or feature_name when available. For 10x-derived input, Symbol is preferred by default when present.

sample_col — Optional sample identifier column in cell_metadata.

batch_col — Optional batch identifier column in cell_metadata.

condition_col — Optional condition column in cell_metadata.

time_col — Optional time or ordering column in cell_metadata.

sample_id_value — Optional fixed sample identifier to assign to all cells during object creation. If sample_col is NULL, a sample_id column is created automatically.

batch_value — Optional fixed batch identifier to assign to all cells during object creation. If batch_col is NULL, a batch column is created automatically.

condition_value — Optional fixed condition value to assign to all cells during object creation. If condition_col is NULL, a condition column is created automatically.

time_value — Optional fixed time value to assign to all cells during object creation. If time_col is NULL, a time column is created automatically.

organism — Organism label stored in package metadata.

seurat_assay — Assay name to extract when x is a Seurat object. If NULL, the active/default Seurat assay is used.

seurat_layer — Layer to use as raw counts when x is a Seurat object. Defaults to "counts".

transfer_reductions — Logical indicating whether existing Seurat dimensional reductions (for example pca and umap) should be copied into reducedDims(sce).

run_unintegrated — Logical indicating whether to run a standard unintegrated preprocessing workflow after object creation. This computes normalized counts, identifies highly variable features, runs PCA, clusters cells, and computes an unintegrated UMAP without requiring Seurat.

unintegrated_feature_set — Feature selection strategy passed to the unintegrated PCA step. Defaults to "hvg".

unintegrated_dims — Integer vector of dimensions to retain for PCA, neighbor graph construction, clustering, and UMAP.

unintegrated_cluster_col — Column name used to store unintegrated cluster labels in colData(sce).

unintegrated_umap_name — Reduced-dimension name used to store the unintegrated UMAP embedding.

unintegrated_k — Number of nearest neighbours used for unintegrated graph construction and clustering.

unintegrated_resolution — Cluster granularity used for unintegrated clustering. Higher values typically produce more clusters.

unintegrated_algorithm — Graph clustering backend for the unintegrated workflow.

For Cell Ranger / 10x input, users can pass the directory path directly. scProkaR will import the matrix, standardize cell metadata, and by default use rowData(sce)$Symbol as the feature name column when that column is available. Set feature_name_col explicitly to override this behavior.

Value

A standardized SingleCellExperiment.

Examples

```
set.seed(1)
genes <- paste0("gene", seq_len(40))
cells <- paste0("cell", seq_len(60))
counts <- matrix(
    rpois(length(genes) * length(cells), lambda = 3),
    nrow = length(genes),
    dimnames = list(genes, cells)
)
cell_meta <- data.frame(
    sample_id = rep(c("s1", "s2"), each = 30),
    time = rep(c(0, 30, 60), times = 20),
    row.names = cells
)
feature_meta <- data.frame(
    gene_class = rep(c("core", "accessory"), length.out = 40),
    row.names = genes
)

sce <- CreateBacObject(
    counts,
    cell_metadata = cell_meta,
    feature_metadata = feature_meta,
    sample_col = "sample_id",
    time_col = "time",
    organism = "Escherichia coli"
)
sce
table(sce$sample_id, sce$time)
S4Vectors::metadata(sce)$scProkaR$organism
```

---

directed_reachability_concordance

Quantify how well a directed cluster graph follows real time

Description

For ordered cluster pairs with increasing median experimental time, measure whether the directed cluster graph supports reachability in the forward direction.

Usage

```
directed_reachability_concordance(
  cluster_graph,
  clusters,
  timepoint,
  min_time_gap = 0,
  weight_by = c("cells", "pairs")
)
```

Arguments

cluster_graph	Directed cluster-level graph as an igraph.
clusters	Cluster label for each cell.
timepoint	Experimental time labels for the same cells.
min_time_gap	Minimum difference in median cluster time required before a pair is considered a meaningful forward-time comparison.
weight_by	How to weight cluster-pair comparisons. Choices are: • "cells": weight each ordered pair by the product of the two cluster sizes. • "pairs": weight every ordered pair equally.

Value

A list with:

- summary: a one-row data.frame containing the weighted forward reachability concordance and the number of ordered cluster pairs.
- pair_table: a data.frame describing each ordered cluster pair.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce, dimred = "PCA", time_col = "timepoint", k = 15)

reachability <- directed_reachability_concordance(
    cluster_graph = tata$cluster_graph,
    clusters = tata$sce$cluster,
    timepoint = tata$sce$timepoint
)
reachability$summary

# Cluster pairs separated by at least one full sampling interval.
head(reachability$pair_table[reachability$pair_table$time_gap >= 24, ])
```

FilterBacCells *Filter cells using bacterial QC thresholds*

Description

Applies simple threshold-based filtering to QC metrics produced by RunBacQC().

Usage

```
FilterBacCells(
  sce,
  min_counts = NULL,
  min_features = NULL,
  max_rrna_fraction = NULL,
  custom_filter = NULL
)
```

Arguments

sce	A SingleCellExperiment with QC metrics already present.
min_counts	Minimum allowed total counts.
min_features	Minimum allowed number of detected features.
max_rrna_fraction	Maximum allowed rRNA fraction.
custom_filter	Optional custom filter. May be a logical vector, a column name in colData(sce), or a function returning a logical vector.

Value

A filtered SingleCellExperiment.

Examples

```
set.seed(1)
genes <- c("rrsA", "rrlB", "rplA", "rpsB", paste0("gene", 1:26))
counts <- matrix(
    rpois(length(genes) * 50, lambda = 3),
    nrow = length(genes),
    dimnames = list(genes, paste0("cell", seq_len(50)))
)
sce <- SingleCellExperiment::SingleCellExperiment(
    assays = list(counts = counts)
)
sce <- RunBacQC(sce)

filtered <- FilterBacCells(
    sce,
    min_counts = 80,
    min_features = 20,
    max_rrna_fraction = 0.2
)
c(before = ncol(sce), after = ncol(filtered))

# A custom filter may be given as a function of colData()
custom <- FilterBacCells(
    sce,
    custom_filter = function(cd) {
        cd$detected_features >= stats::median(cd$detected_features)
    }
)
ncol(custom)
```

---

filter_pseudobulk_samples

Filter pseudobulk samples by size

Description

Filter a pseudobulk SingleCellExperiment using minimum numbers of cells and library size.

Usage

```
filter_pseudobulk_samples(pb, min_cells = 10L, min_lib_size = NULL)
```

Arguments

pb	A pseudobulk SingleCellExperiment, typically returned by aggregate_pseudobulk().
min_cells	Minimum number of cells required.
min_lib_size	Minimum library size required.

Value

A filtered pseudobulk SingleCellExperiment.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600,
    n_features = 60,
    n_pcs = 10
)
sce$replicate <- factor(sample(seq_len(3), ncol(sce), replace = TRUE))

pb <- aggregate_pseudobulk(
    sce,
    sample_cols = c("condition", "timepoint", "replicate"),
    aggregation = "sum",
    min_cells = 1
)

# Drop small or shallowly sequenced pseudobulk samples.
kept <- filter_pseudobulk_samples(
    pb,
    min_cells = 10,
    min_lib_size = 1000
)
c(before = ncol(pb), after = ncol(kept))
range(kept$ncells)
```

IntegrateBacData *Integrate microbial single-cell batches*

Description

Runs one of several integration backends and stores the corrected embedding in reducedDims(sce).

Usage

```
IntegrateBacData(
  sce,
  batch_col,
  method = c("mnn", "harmony"),
```

```
  feature_set = c("hvg", "all"),
  dims = seq_len(30),
  assay_name = "counts",
  integrated_name = NULL,
  ...
)
```

Arguments

sce	A SingleCellExperiment.
batch_col	Column in colData(sce) containing batch assignments.
method	Integration backend: "mnn" or "harmony".
feature_set	Either "hvg" or "all".
dims	Integer vector of dimensions to retain.
assay_name	Assay used for preprocessing.
integrated_name	Optional name for the output reduced dimension.
...	Additional backend-specific arguments.

Value

A SingleCellExperiment with an integrated reduced dimension.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 300, n_features = 60, n_pcs = 10
)
# Assign a technical batch label to integrate over.
sce$batch <- rep(c("batch1", "batch2"), length.out = ncol(sce))

if (requireNamespace("batchelor", quietly = TRUE)) {
    sce <- IntegrateBacData(
        sce,
        batch_col = "batch",
        method = "mnn",
        dims = 1:8
    )
    print(SingleCellExperiment::reducedDimNames(sce))
}

if (requireNamespace("harmony", quietly = TRUE)) {
    sce <- IntegrateBacData(
        sce,
        batch_col = "batch",
        method = "harmony",
        dims = 1:8
    )
    harmony_emb <- SingleCellExperiment::reducedDim(
        sce, "integrated_harmony"
    )
    print(dim(harmony_emb))
}
```

`local_temporal_order_metrics` *Compute local temporal order metrics on a cell graph*

Description

Summarize how often pseudotime agrees with the known direction of time on local cell-cell graph edges.

Usage

```
local_temporal_order_metrics(pseudotime, adjacency, timepoint)
```

Arguments

`pseudotime`	Numeric vector of inferred pseudotime values.
`adjacency`	Sparse adjacency matrix for the cell-cell graph.
`timepoint`	Experimental time labels for the same cells.

Value

A one-row `data.frame` with the number of compared edges, local time concordance, local temporal inversion rate, a higher-is-better temporal direction score, and the fraction of edges tied in pseudotime.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce, dimred = "PCA", time_col = "timepoint", k = 15)

oriented <- orient_pseudotime_to_time(
    pseudotime = tata$cell_pseudotime,
    timepoint = tata$sce$timepoint
)
local_temporal_order_metrics(
    pseudotime = oriented,
    adjacency = tata$adjacency_matrix,
    timepoint = tata$sce$timepoint
)
```

`MergeBacObjects` *Merge multiple scProkaR or SingleCellExperiment objects*

Description

`MergeBacObjects()` combines multiple per-sample `SingleCellExperiment` objects without relying on external `cbind()` method dispatch. This is useful when merging sample-level objects created by `CreateBacObject()` before batch integration.

Usage

```
MergeBacObjects(..., objects = NULL, gene_mode = c("intersect", "union"))
```

Arguments

`...`	Individual `SingleCellExperiment` objects to merge.
`objects`	Optional list of `SingleCellExperiment` objects.
`gene_mode`	Either `"intersect"` or `"union"`.

Details

By default the function keeps the intersection of genes across inputs, which is the safest choice before integration.

Value

A merged `SingleCellExperiment`.

Examples

```
set.seed(1)
genes_a <- paste0("gene", 1:30)
genes_b <- paste0("gene", 11:40)
cells <- paste0("cell", seq_len(20))
counts_a <- matrix(
    rpois(length(genes_a) * 20, lambda = 4),
    nrow = length(genes_a),
    dimnames = list(genes_a, cells)
)
counts_b <- matrix(
    rpois(length(genes_b) * 20, lambda = 4),
    nrow = length(genes_b),
    dimnames = list(genes_b, cells)
)

sce_a <- CreateBacObject(counts_a, sample_id_value = "sampleA")
sce_b <- CreateBacObject(counts_b, sample_id_value = "sampleB")

## Keep only the genes shared by both samples.
merged <- MergeBacObjects(sce_a, sce_b, gene_mode = "intersect")
dim(merged)
table(merged$sample_id)
head(colnames(merged), 3)
```

normalize_pseudobulk *Normalize pseudobulk samples for exploratory analysis*

Description

Compute edgeR normalization factors and add a `logcounts` assay for plotting and exploratory analysis.

Usage

```
normalize_pseudobulk(
  pb,
  assay_name = "counts",
  method = c("TMM", "upperquartile", "none"),
  output_assay = "logcounts",
  prior_count = 2
)
```

Arguments

`pb`	A pseudobulk `SingleCellExperiment`.
`assay_name`	Assay containing pseudobulk counts.
`method`	Normalization method. Choices are: • `"TMM"`: trimmed mean of M values normalization via `edgeR::calcNormFactors(..., method = "TMM")`. • `"upperquartile"`: upper-quartile normalization via edgeR. • `"none"`: do not apply normalization factors; still compute `logcounts` from the raw library sizes.
`output_assay`	Name of the output log-scale assay.
`prior_count`	Prior count passed to `edgeR::cpm(..., log = TRUE)`.

Value

The input pseudobulk object with updated `colData` normalization fields and a log-scale assay.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600,
    n_features = 60,
    n_pcs = 10
)
sce$replicate <- factor(sample(seq_len(3), ncol(sce), replace = TRUE))

pb <- aggregate_pseudobulk(
    sce,
    sample_cols = c("condition", "timepoint", "replicate"),
    aggregation = "sum",
    min_cells = 5
)
```

```
# TMM normalization factors plus a log-CPM assay for plotting.
pb <- normalize_pseudobulk(pb, method = "TMM")
summary(pb$norm.factors)
SummarizedExperiment::assay(pb, "logcounts")[1:3, 1:3]
```

`orient_pseudotime_to_time` *Orient pseudotime to match increasing real time*

Description

Flip an inferred pseudotime vector when needed so that it best agrees with increasing experimental time. This is useful when benchmarking methods whose pseudotime direction is arbitrary up to reversal.

Usage

```
orient_pseudotime_to_time(
  pseudotime,
  timepoint,
  method = c("spearman", "kendall", "pearson")
)
```

Arguments

`pseudotime`	Numeric vector of inferred pseudotime values.
`timepoint`	Experimental time labels for the same cells.
`method`	Correlation used to choose the orientation. Choices are: • `"spearman"`: rank-based and the default choice for monotone trajectories. • `"kendall"`: more conservative rank correlation. • `"pearson"`: linear correlation on the scaled values.

Value

A numeric vector scaled to `[0, 1]` and oriented so that larger values agree as well as possible with later experimental time.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce, dimred = "PCA", time_col = "timepoint", k = 15)

oriented <- orient_pseudotime_to_time(
    pseudotime = tata$cell_pseudotime,
    timepoint = tata$sce$timepoint
)
summary(oriented)
```

```
# Larger oriented values should now fall on later experimental times.
tapply(oriented, tata$sce$timepoint, median)
```

`PlotIntegrationOverview` *Plot Seurat-style integration result panels*

Description

Creates a standard set of integration diagnostic plots inspired by common Seurat workflows: unintegrated and integrated views coloured by batch and, optionally, a biological label, plus a split-panel view of the integrated embedding.

Usage

```
PlotIntegrationOverview(
  sce,
  unintegrated_reduction = "umap",
  integrated_reduction = NULL,
  batch_col,
  label_col = NULL,
  split_by = NULL,
  point_size = 0.4,
  point_alpha = 0.8,
  shuffle = FALSE
)
```

Arguments

`sce`	A `SingleCellExperiment`.
`unintegrated_reduction`	Reduction name for the pre-integration view.
`integrated_reduction`	Reduction name for the integrated view. If `NULL`, the function prefers a stored `umap_integrated_*` reduction and otherwise falls back to `integrated_*`.
`batch_col`	Batch column in `colData(sce)`.
`label_col`	Optional biological label column in `colData(sce)`.
`split_by`	Optional metadata column used to facet the integrated plot. This is the `scProkaR` equivalent of Seurat's `split.by`.
`point_size`	Point size passed to `PlotReduction()`.
`point_alpha`	Point alpha passed to `PlotReduction()`.
`shuffle`	Whether to shuffle plotting order.

Value

A named list of `ggplot2` objects.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 300, n_features = 60, n_pcs = 10
)
sce$batch <- rep(c("batch1", "batch2"), length.out = ncol(sce))

# Any corrected embedding can stand in for the integrated view; here a
# toy correction is registered so the example needs no extra backend.
corrected <- SingleCellExperiment::reducedDim(sce, "PCA")[, 1:2]
is_batch2 <- sce$batch == "batch2"
corrected[is_batch2, ] <- corrected[is_batch2, ] * 0.9
sce <- RegisterIntegrationEmbedding(
    sce,
    embedding = corrected,
    method_name = "toy"
)

plots <- PlotIntegrationOverview(
    sce,
    unintegrated_reduction = "PCA",
    integrated_reduction = "integrated_toy",
    batch_col = "batch",
    label_col = "condition",
    split_by = "batch"
)
print(names(plots))
print(class(plots$integrated_by_batch))
```

`PlotReduction` *Plot any stored reduced dimension*

Description

Creates a quick scatter plot for an existing reduced dimension such as an original Seurat UMAP, an integrated latent space, or a UMAP computed by `RunIntegratedUMAP()`.

Usage

```
PlotReduction(
  sce,
  reduction,
  colour_by = NULL,
  point_size = 0.4,
  point_alpha = 0.8,
  palette = NULL,
  facet_by = NULL,
  shuffle = FALSE
)
```

Arguments

sce	A SingleCellExperiment.
reduction	Reduced-dimension name to plot.
colour_by	Optional colData(sce) column used for colouring points.
point_size	Point size.
point_alpha	Point alpha.
palette	Optional colour palette. For discrete variables, this should be a character vector of colours; for continuous variables, it should be a gradient vector.
facet_by	Optional colData(sce) column used to facet the plot.
shuffle	Whether to shuffle plotting order before drawing points.

Value

A ggplot2 object.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 300, n_features = 60, n_pcs = 10
)
plot <- PlotReduction(
    sce,
    reduction = "PCA",
    colour_by = "condition",
    point_size = 0.6,
    shuffle = TRUE
)
print(class(plot))
print(plot$labels[c("x", "y")])

# Facet the same embedding by sampling time.
faceted <- PlotReduction(
    sce,
    reduction = "PCA",
    colour_by = "condition",
    facet_by = "timepoint"
)
print(nrow(faceted$data))
```

---

plot_pairwise_de_ma *Plot an MA plot for a pairwise edgeR contrast*

Description

Visualize one pairwise contrast from run_edger_pairwise_de() as an MA plot using average abundance and log-fold change.

Usage

```
plot_pairwise_de_ma(
  de_result,
  contrast = NULL,
  fdr_cutoff = 0.05,
  point_size = 1.5,
  point_alpha = 0.75
)
```

Arguments

de_result	Result list returned by run_edger_pairwise_de().
contrast	Name of the contrast to plot. If NULL, the first available contrast is used.
fdr_cutoff	FDR threshold used to define significant genes.
point_size	Point size.
point_alpha	Point alpha.

Value

A ggplot2 object.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60, n_pcs = 10
)
sce$replicate <- rep(paste0("rep", 1:3), length.out = ncol(sce))

pb <- aggregate_pseudobulk(
    sce,
    sample_cols = c("condition", "timepoint", "replicate"),
    aggregation = "sum",
    min_cells = 5
)
pb <- normalize_pseudobulk(pb, method = "TMM")

de <- run_edger_pairwise_de(
    pb,
    group_col = "timepoint",
    covariates = "condition",
    contrast_type = "reference"
)

p <- plot_pairwise_de_ma(de, contrast = "120_vs_0", fdr_cutoff = 0.05)
p
```

plot_pairwise_de_volcano *Plot a volcano plot for a pairwise edgeR contrast*

Description

Visualize one pairwise contrast from `run_edger_pairwise_de()` as a volcano plot using log-fold change and FDR.

Usage

```
plot_pairwise_de_volcano(
  de_result,
  contrast = NULL,
  fdr_cutoff = 0.05,
  lfc_cutoff = 0,
  top_n_labels = 10L,
  point_size = 1.5,
  point_alpha = 0.75
)
```

Arguments

de_result	Result list returned by `run_edger_pairwise_de()`.
contrast	Name of the contrast to plot. If `NULL`, the first available contrast is used.
fdr_cutoff	FDR threshold used to define significant genes.
lfc_cutoff	Absolute log-fold-change threshold used to highlight larger effects.
top_n_labels	Number of top genes to label by FDR. Set to `0` to omit labels.
point_size	Point size.
point_alpha	Point alpha.

Value

A `ggplot2` object.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60, n_pcs = 10
)
sce$replicate <- rep(paste0("rep", 1:3), length.out = ncol(sce))

pb <- aggregate_pseudobulk(
    sce,
    sample_cols = c("condition", "timepoint", "replicate"),
    aggregation = "sum",
    min_cells = 5
)
pb <- normalize_pseudobulk(pb, method = "TMM")
```

```
de <- run_edger_pairwise_de(
    pb,
    group_col = "timepoint",
    covariates = "condition",
    contrast_type = "reference"
)

p <- plot_pairwise_de_volcano(
    de,
    contrast = "120_vs_0",
    fdr_cutoff = 0.05,
    lfc_cutoff = 0.25,
    top_n_labels = 5
)
p
```

`plot_tata_branch_confusion` *Plot truth versus inferred TATA branch assignments*

Description

For simulated data, compare a ground-truth branch label with the inferred hard TATA branch assignment.

Usage

```
plot_tata_branch_confusion(
  tata_result,
  truth_col = "simulated_branch",
  predicted_col = "tata_branch_assignment",
  normalize = c("truth", "predicted", "none")
)
```

Arguments

`tata_result`	Result list returned by `run_tata()`.
`truth_col`	`colData` column containing the reference branch labels.
`predicted_col`	`colData` column containing the inferred TATA branch assignment.
`normalize`	Normalization scheme. Choices are: • `"truth"`: normalize each truth branch to sum to 1. • `"predicted"`: normalize each predicted branch to sum to 1. • `"none"`: raw counts.

Value

A `ggplot2` object.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce,
    dimred = "PCA", time_col = "timepoint", k = 15,
    expected_branches = 3
)

# `simulated_branch` records the true branch of each simulated cell.
p <- plot_tata_branch_confusion(tata,
    truth_col = "simulated_branch",
    normalize = "truth"
)
p
```

---

plot_tata_branch_probabilities

Plot TATA branch probabilities on an embedding

Description

Plot one panel per terminal branch, colored by the corresponding TATA branch probability.

Usage

```
plot_tata_branch_probabilities(
  tata_result,
  dimred = "UMAP",
  branches = NULL,
  point_size = 0.6,
  point_alpha = 0.8,
  ncol = 2
)
```

Arguments

tata_result	Result list returned by `run_tata()`.
dimred	Reduced dimension to plot.
branches	Optional subset of terminal branches to display.
point_size	Point size.
point_alpha	Point alpha.
ncol	Number of facet columns.

Value

A `ggplot2` object.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce,
    dimred = "PCA", time_col = "timepoint", k = 15,
    expected_branches = 3
)

# One panel per terminal branch, coloured by branch probability.
p <- plot_tata_branch_probabilities(tata, dimred = "UMAP", ncol = 2)
p
```

---

plot_tata_branch_selection

Plot selected cells from one TATA branch

Description

Highlight cells that belong to one terminal branch using either a hard or a soft branch-selection rule.

Usage

```
plot_tata_branch_selection(
  tata_result,
  branch,
  dimred = "UMAP",
  selection_mode = c("soft", "hard"),
  probability_threshold = 0.25,
  point_size = 0.6,
  point_alpha = 0.8
)
```

Arguments

`tata_result`	Result list returned by `run_tata()`.
`branch`	Terminal branch or terminal cluster to select.
`dimred`	Reduced dimension to plot.
`selection_mode`	How to define selected cells. Choices are: • `"soft"`: select cells whose probability for the branch is at least `probability_threshold`. • `"hard"`: select cells whose hard TATA branch assignment equals `branch`.
`probability_threshold`	Minimum branch probability required for `selection_mode = "soft"`.
`point_size`	Point size.
`point_alpha`	Point alpha.

Value

A ggplot2 object.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce,
    dimred = "PCA", time_col = "timepoint", k = 15,
    expected_branches = 3
)

# Highlight cells with an appreciable probability of one terminal fate.
branch <- tata$terminal_clusters[1]
p <- plot_tata_branch_selection(tata,
    branch = branch, dimred = "UMAP",
    selection_mode = "soft", probability_threshold = 0.25
)
p
```

---

plot_tata_cluster_graph

Plot the cluster-level TATA graph

Description

Plot the directed cluster-level TATA graph. The graph can be drawn in its own abstract layout or positioned on a chosen reduced dimension so that the graph coordinates match the embedding shown to the user.

Usage

```
plot_tata_cluster_graph(
  tata_result,
  dimred = NULL,
  layout_mode = NULL,
  cluster_col = NULL,
  show_cells = FALSE,
  colour_by = NULL,
  node_colour_by = c("median_time", "cluster"),
  main = "TATA cluster graph",
  vertex_label_size = 3.5,
  edge_width_scale = 5,
  point_size = 0.4,
  point_alpha = 0.45,
  arrow_size = 0.15
)
```

Arguments

tata_result	Result list returned by `run_tata()`.

dimred	Optional reduced dimension used to position cluster nodes. When supplied together with layout_mode = "embedding", cluster centroids are computed from this embedding.
layout_mode	Layout mode for the graph. Choices are: • "graph": use the stored abstract graph coordinates, or a force-directed layout if they are unavailable. • "embedding": place cluster nodes at their centroids in the selected reduced dimension.
cluster_col	Optional cluster column used when layout_mode = "embedding". By default this is taken from tata_result$parameters.
show_cells	Logical indicating whether to draw cells in the background when layout_mode = "embedding".
colour_by	Optional colData column used to color background cells. If NULL and show_cells = TRUE, the cluster column is used when available.
node_colour_by	Node coloring scheme. Choices are: • "median_time": color nodes by cluster median time. • "cluster": color nodes by cluster identity, which is useful for embedding-aligned graph plots.
main	Plot title.
vertex_label_size	Label size.
edge_width_scale	Relative edge width scale.
point_size	Background point size when cells are drawn.
point_alpha	Background point alpha when cells are drawn.
arrow_size	Arrow size for directed edges.

Value

A ggplot2 object.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce,
    dimred = "PCA", time_col = "timepoint", k = 15,
    expected_branches = 3
)

# Abstract graph layout, nodes coloured by cluster median time.
p <- plot_tata_cluster_graph(tata)

# The same graph positioned on the UMAP embedding.
p_embedded <- plot_tata_cluster_graph(tata,
    dimred = "UMAP",
    layout_mode = "embedding", show_cells = TRUE,
    node_colour_by = "cluster"
```

```
)
p_embedded
```

`plot_tata_edge_causality` *Plot TATA edge causality diagnostics*

Description

Visualize how topology support and temporal evidence combine for each cluster-cluster edge in the abstract TATA graph.

Usage

```
plot_tata_edge_causality(tata_result, label_edges = FALSE, kept_only = FALSE)
```

Arguments

`tata_result`	Result list returned by `run_tata()`.
`label_edges`	Logical indicating whether to label each point by the cluster-pair direction label.
`kept_only`	Logical indicating whether to plot only retained edges.

Value

A `ggplot2` object.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce,
    dimred = "PCA", time_col = "timepoint", k = 15,
    expected_branches = 3
)

# All candidate edges, showing which ones survived pruning.
p <- plot_tata_edge_causality(tata)

# Retained edges only, labelled by their inferred direction.
p_kept <- plot_tata_edge_causality(tata,
    kept_only = TRUE,
    label_edges = TRUE
)
p_kept
```

plot_tata_gene_trends *Plot fitted TATA gene trends*

Description

Plot fitted branch-specific gene-expression trends over TATA pseudotime.

Usage

```
plot_tata_gene_trends(trend_result, genes = NULL, branches = NULL, ncol = 2)
```

Arguments

trend_result	Result list returned by compute_tata_gene_trends().
genes	Optional subset of genes to plot.
branches	Optional subset of branches to plot.
ncol	Number of facet columns.

Value

A ggplot2 object.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce,
    dimred = "PCA", time_col = "timepoint", k = 15,
    expected_branches = 3
)

# Fit trends for a few early- and late-programme marker genes.
module <- SummarizedExperiment::rowData(tata$sce)$feature_module
genes <- head(rownames(tata$sce)[module %in% c("early", "late")], 4)
trends <- compute_tata_gene_trends(tata, genes = genes)
p <- plot_tata_gene_trends(trends, ncol = 2)
p
```

plot_tata_gene_trend_heatmap *Plot a heatmap of TATA gene trends*

Description

Plot a heatmap of branch-specific fitted gene-expression trends over TATA pseudotime.

Usage

```
plot_tata_gene_trend_heatmap(
  trend_result,
  branches = NULL,
  scale_rows = TRUE,
  n_clusters = NULL
)
```

Arguments

trend_result	Result list returned by compute_tata_gene_trends().
branches	Optional subset of branches.
scale_rows	Logical indicating whether to z-score each gene-branch trend before plotting.
n_clusters	Optional number of k-means clusters used to order genes by trend shape.

Value

A ggplot2 object.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce,
    dimred = "PCA", time_col = "timepoint", k = 15,
    expected_branches = 3
)

# Z-scored trends, with genes grouped by k-means on trend shape.
trends <- compute_tata_gene_trends(tata,
    genes = head(rownames(tata$sce), 12)
)
p <- plot_tata_gene_trend_heatmap(trends,
    scale_rows = TRUE,
    n_clusters = 3
)
p
```

---

plot_tata_results *Plot an overview of TATA results*

Description

Create a compact set of standard TATA result plots analogous to a trajectory overview dashboard.

Usage

```
plot_tata_results(tata_result, dimred = "UMAP", n_probability_cells = 6)
```

Arguments

tata_result	Result list returned by run_tata().
dimred	Reduced dimension to use for the embedding plots.
n_probability_cells	Number of representative cells to show in the terminal-probability barplot.

Value

A named list of ggplot2 objects.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce,
    dimred = "PCA", time_col = "timepoint", k = 15,
    expected_branches = 3
)

plots <- plot_tata_results(tata, dimred = "UMAP")
names(plots)
plots$pseudotime
```

---

plot_tata_terminal_probabilities

Plot branch probabilities for representative cells

Description

Show a barplot of terminal-branch probabilities for selected cells.

Usage

```
plot_tata_terminal_probabilities(tata_result, cell_ids = NULL, n_cells = 6)
```

Arguments

tata_result	Result list returned by run_tata().
cell_ids	Optional cell ids. If NULL, the cells with the highest TATA branch entropy are shown.
n_cells	Number of representative cells to display when cell_ids is NULL.

Value

A ggplot2 object.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce,
    dimred = "PCA", time_col = "timepoint", k = 15,
    expected_branches = 3
)

# The most branch-ambiguous cells have the flattest probability bars.
p <- plot_tata_terminal_probabilities(tata, n_cells = 4)
p
```

`plot_tata_time_calibration` *Plot TATA time calibration*

Description

Compare inferred TATA pseudotime against the known experimental time labels, optionally split by a branch or condition label.

Usage

```
plot_tata_time_calibration(
  tata_result,
  time_col = NULL,
  branch_col = NULL,
  summary = c("median_iqr", "boxplot"),
  pseudotime_col = "tata_pseudotime_scaled"
)
```

Arguments

`tata_result`	Result list returned by `run_tata()`.
`time_col`	`colData` column containing experimental time labels. By default this is taken from `tata_result$parameters`.
`branch_col`	Optional `colData` column used to group curves, for example a true simulated branch or treatment label.
`summary`	Plot style. Choices are: • `"median_iqr"`: median pseudotime with an IQR ribbon for each group. • `"boxplot"`: per-timepoint boxplots of pseudotime.
`pseudotime_col`	`colData` column containing the cell-level TATA pseudotime values to plot.

Value

A `ggplot2` object.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce,
    dimred = "PCA", time_col = "timepoint", k = 15,
    expected_branches = 3
)

# Median pseudotime with an IQR ribbon for each treatment condition.
p <- plot_tata_time_calibration(tata, branch_col = "condition")

# The same comparison shown as per-timepoint boxplots.
p_box <- plot_tata_time_calibration(tata,
    branch_col = "condition",
    summary = "boxplot"
)
p_box
```

---

plot_tata_trajectory_embedding

Plot TATA trajectories over an embedding

Description

Overlay inferred TATA trajectory paths on top of a reduced-dimensional embedding such as UMAP. The plotted paths follow root-to-terminal routes in the cluster graph and can be shown as smoothed curves for a more classical trajectory-style visualization.

Usage

```
plot_tata_trajectory_embedding(
  tata_result,
  dimred = "UMAP",
  colour_by = "timepoint",
  max_paths = NULL,
  point_size = 0.6,
  point_alpha = 0.75,
  smooth_paths = TRUE,
  n_curve_points = 100,
  show_centroids = TRUE,
  show_labels = TRUE,
  curve_colour = "black",
  label_size = 3
)
```

Arguments

`tata_result`	Result list returned by `run_tata()`.
`dimred`	Reduced dimension to plot.

colour_by	Optional colData column used to color cells. Set to NULL for a grey background.
max_paths	Optional maximum number of paths to plot after path selection. NULL keeps all available paths.
point_size	Background point size.
point_alpha	Background point alpha.
smooth_paths	Whether to smooth the inferred trajectory paths.
n_curve_points	Number of interpolation points per curve.
show_centroids	Whether to draw cluster centroids.
show_labels	Whether to label cluster centroids.
curve_colour	Colour of the inferred trajectory curves.
label_size	Cluster label size.

Value

A ggplot2 object.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60,
    n_pcs = 10
)
tata <- run_tata(sce,
    dimred = "PCA", time_col = "timepoint", k = 15,
    expected_branches = 3
)

# Overlay the three root-to-terminal routes on the UMAP embedding.
p <- plot_tata_trajectory_embedding(tata,
    dimred = "UMAP",
    colour_by = "timepoint", max_paths = 3
)
p
```

---

plot_time_series_deg_curves

Plot fitted spline expression curves from time-series DE

Description

Plot fitted spline curves for selected genes, optionally overlaid across conditions. When a requested gene was significant in one condition but not another, the non-significant condition is still drawn using a dotted line.

Usage

```
plot_time_series_deg_curves(
  spline_result,
  genes = NULL,
  top_n = 12L,
  scales = "free_y",
  ncol = NULL
)
```

Arguments

spline_result	Result list returned by run_edger_spline_de().
genes	Optional character vector of genes to plot. If NULL, the top genes ranked by FDR are used.
top_n	Number of genes to plot when genes = NULL.
scales	Facet scale setting passed to ggplot2::facet_wrap().
ncol	Optional number of columns used by ggplot2::facet_wrap().

Value

A ggplot2 object.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60, n_pcs = 10
)
sce$replicate <- rep(paste0("rep", 1:3), length.out = ncol(sce))

pb <- aggregate_pseudobulk(
    sce,
    sample_cols = c("condition", "timepoint", "replicate"),
    aggregation = "sum",
    min_cells = 5
)
pb <- normalize_pseudobulk(pb, method = "TMM")

spline_de <- run_edger_spline_de(
    pb,
    time_col = "timepoint",
    condition_col = "condition",
    df = 3,
    curve_grid_length = 50
)

# Top temporally variable genes, overlaid across the three drugs.
p <- plot_time_series_deg_curves(spline_de, top_n = 4, ncol = 2)
p
```

`RegisterExistingIntegrationEmbeddings` *Register existing reduced dimensions as integration embeddings*

Description

Registers embeddings already stored in a `SingleCellExperiment`, including Seurat CCA/RPCA, Scanorama, scVI, SCALEX, BBKNN, ComBat and LIGER latent spaces. It does not run these external tools or import their native objects. The selected reductions are copied or referenced and recorded in `metadata(sce)$scProkaR$integration`, so `BenchmarkIntegration()` can compare them with scProkaR MNN and Harmony outputs.

Usage

```
RegisterExistingIntegrationEmbeddings(
  sce,
  reductions = NULL,
  method_names = NULL,
  copy = TRUE,
  prefix = "integrated_",
  metadata = list(),
  overwrite = FALSE
)
```

Arguments

`sce`	A `SingleCellExperiment`.
`reductions`	Character vector of existing reduced-dimension names. If `NULL`, likely integration reductions are detected from their names. Pass names explicitly when automatic detection is unsuitable.
`method_names`	Optional names to use in benchmark tables. Must be unique and have the same length as `reductions`. By default, names are inferred from the reductions and converted to lower case.
`copy`	Logical. If `TRUE`, reductions not already starting with `prefix` are copied to `paste0(prefix, method_name)`. If `FALSE`, their original names are registered directly.
`prefix`	Prefix used when `copy = TRUE`.
`metadata`	Optional named list added to each method's metadata. A list entry named by `method_name`, if present, is used for that method instead. `source` and `original_reduction` are recorded by this function.
`overwrite`	Logical indicating whether copied reduced dimensions may replace existing reductions with different values.

Value

A `SingleCellExperiment` with registered integration embeddings.

Examples

```
set.seed(1)
sce <- simulate_tata_sce(n_cells = 1000, n_features = 60)
# Stand-in for a latent space from an external integration tool.
SingleCellExperiment::reducedDim(sce, "X_scVI") <-
    SingleCellExperiment::reducedDim(sce, "PCA")[, seq_len(3)]
sce <- RegisterExistingIntegrationEmbeddings(sce, reductions = "X_scVI")
SingleCellExperiment::reducedDimNames(sce)
S4Vectors::metadata(sce)$scProkaR$integration$results$scvi
```

RegisterIntegrationEmbedding

Register an external embedding for benchmarking and downstream analysis

---

Description

Adds a precomputed embedding to `reducedDims(sce)` and records it in `metadata(sce)$scProkaR$integration`, allowing custom integration outputs to be benchmarked alongside native scProkaR methods.

Usage

```
RegisterIntegrationEmbedding(
  sce,
  embedding,
  method_name,
  reduction_name = NULL,
  metadata = list(),
  overwrite = FALSE
)
```

Arguments

`sce`	A `SingleCellExperiment`.
`embedding`	A matrix-like cell embedding with one row per cell.
`method_name`	Name used when recording the embedding.
`reduction_name`	Optional reducedDim name. Defaults to `paste0("integrated_", method_name)`.
`metadata`	Optional named list describing the external method.
`overwrite`	Logical indicating whether to replace an existing reduced dimension with different values. Defaults to `FALSE`; registering the same embedding again is allowed.

Value

A `SingleCellExperiment` with the external embedding registered.

Examples

```
set.seed(1)
sce <- simulate_tata_sce(n_cells = 1000, n_features = 60)

# Stand-in for an embedding produced by an external integration tool.
external <- SingleCellExperiment::reducedDim(sce, "PCA")[, 1:2]
colnames(external) <- c("EXT_1", "EXT_2")

sce <- RegisterIntegrationEmbedding(
    sce,
    embedding = external,
    method_name = "my_method",
    metadata = list(software = "external_tool", version = "1.0")
)
print(SingleCellExperiment::reducedDimNames(sce))
record <- S4Vectors::metadata(sce)$scProkaR$integration$results
print(record$my_method)
```

RunBacQC *Compute bacterial QC metrics*

Description

Adds Seurat-style per-cell feature fractions to a SingleCellExperiment, including rRNA and ribosomal-protein content.

Usage

```
RunBacQC(
  sce,
  rrna_pattern = "^(rrs|rrl|rrf)",
  ribo_pattern = "^(rpl|rps)",
  gene_class_col = NULL,
  store = TRUE
)
```

Arguments

sce	A SingleCellExperiment.
rrna_pattern	Regular expression used to identify rRNA features.
ribo_pattern	Regular expression used to identify ribosomal protein features.
gene_class_col	Optional rowData column describing feature classes. When present, values matching "rrna" are used for rRNA metrics and values matching "ribo" or "ribosomal_protein" are used for ribosomal protein metrics.
store	If TRUE, store metrics in colData(sce) and return the modified object. If FALSE, return the QC table only.

Value

A SingleCellExperiment when store = TRUE, otherwise a data frame of QC metrics.

Examples

```
set.seed(1)
genes <- c("rrsA", "rrlB", "rrfC", "rplA", "rpsB", paste0("gene", 1:25))
counts <- matrix(
    rpois(length(genes) * 40, lambda = 4),
    nrow = length(genes),
    dimnames = list(genes, paste0("cell", seq_len(40)))
)
sce <- SingleCellExperiment::SingleCellExperiment(
    assays = list(counts = counts)
)

# rRNA and ribosomal protein features are matched on feature names
sce <- RunBacQC(sce)
head(SummarizedExperiment::colData(sce)[, c(
    "total_counts", "detected_features", "rrna_fraction", "pct_ribo"
)])

# Return the QC table only, leaving the object untouched
qc <- RunBacQC(sce, store = FALSE)
summary(qc$rrna_fraction)

# Feature classes stored in rowData() can be used instead of patterns
SummarizedExperiment::rowData(sce)$gene_class <- ifelse(
    rownames(sce) %in% c("rrsA", "rrlB", "rrfC", "gene1"),
    "rrna",
    "other"
)
qc_class <- RunBacQC(sce, gene_class_col = "gene_class", store = FALSE)
summary(qc_class$rrna_fraction)
```

RunIntegratedClustering

Cluster cells on an integrated embedding

Description

Builds a k-nearest-neighbour graph on a corrected embedding and detects communities for downstream trajectory and pseudobulk workflows.

Usage

```
RunIntegratedClustering(
  sce,
  reduction = NULL,
  cluster_col = "integrated_clusters",
  dims = NULL,
  k = 20,
  algorithm = c("louvain", "walktrap", "leiden"),
  resolution = 0.8
)
```

Arguments

sce	A SingleCellExperiment.
reduction	Integrated reduced-dimension name. If NULL, the function prefers a stored integrated_* embedding.
cluster_col	Column name used to store cluster labels in colData(sce).
dims	Optional dimensions to retain from the embedding.
k	Number of nearest neighbours used to build the graph.
algorithm	Graph clustering algorithm: "louvain" or "walktrap".
resolution	Cluster granularity parameter. This is analogous to Seurat's resolution: higher values typically produce more clusters. It is used when the selected graph clustering algorithm supports it.

Value

A SingleCellExperiment with cluster labels added to colData(sce).

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 300, n_features = 60, n_pcs = 10
)
sce <- RunIntegratedClustering(
    sce,
    reduction = "PCA",
    cluster_col = "pca_clusters",
    dims = 1:8,
    k = 15,
    algorithm = "louvain",
    resolution = 0.8
)
print(table(sce$pca_clusters))
print(table(sce$pca_clusters, sce$condition))
```

RunIntegratedUMAP *Run UMAP on an existing embedding*

Description

Computes a two-dimensional UMAP from a stored reduced dimension, which is useful for inspecting integrated embeddings returned by IntegrateBacData() or RegisterIntegrationEmbedding().

Usage

```
RunIntegratedUMAP(
  sce,
  reduction,
  umap_name = NULL,
  n_neighbors = 30,
  min_dist = 0.3,
```

```
  metric = "cosine",
  ...
)
```

Arguments

sce	A SingleCellExperiment.
reduction	Existing reduced-dimension name to use as UMAP input.
umap_name	Optional name for the output UMAP reduced dimension.
n_neighbors	Number of neighbours passed to uwot::umap().
min_dist	UMAP min_dist.
metric	Distance metric passed to uwot::umap().
...	Additional arguments passed to uwot::umap().

Value

A SingleCellExperiment with the computed UMAP stored in reducedDims(sce).

Examples

```
set.seed(1)
sce <- simulate_tata_sce(n_cells = 1000, n_features = 60)

if (requireNamespace("uwot", quietly = TRUE)) {
    sce <- RunIntegratedUMAP(
        sce,
        reduction = "PCA",
        umap_name = "umap_pca",
        n_neighbors = 15
    )
    print(SingleCellExperiment::reducedDimNames(sce))
    print(head(SingleCellExperiment::reducedDim(sce, "umap_pca"), 3))
}
```

---

run_edger_pairwise_de *Run pairwise pseudobulk differential expression with edgeR*

Description

Fit a quasi-likelihood negative-binomial model with edgeR and test pairwise contrasts between groups such as timepoints, treatments, or composite treatment-timepoint groups.

Usage

```
run_edger_pairwise_de(
  pb,
  assay_name = "counts",
  group_col = "timepoint",
  covariates = NULL,
  combine_group_cols = NULL,
  contrast_type = c("reference", "all", "manual"),
```

```
  reference_level = NULL,
  manual_pairs = NULL,
  normalization = c("TMM", "upperquartile", "none"),
  filter_by_expr = TRUE,
  robust = TRUE,
  abundance_trend = FALSE,
  fdr_cutoff = 0.05
)
```

Arguments

pb	A pseudobulk `SingleCellExperiment`.
assay_name	Assay containing pseudobulk counts.
group_col	Sample metadata column defining the primary grouping, for example `"timepoint"`.
covariates	Optional metadata columns to include as additive covariates in the design matrix, for example treatment, batch, or donor.
combine_group_cols	Optional metadata columns to combine into a single composite group via `interaction()`, for example `c("treatment", "timepoint")`.
contrast_type	Contrast construction strategy. Choices are: • `"reference"`: compare every non-reference group against one reference level. This is the default. • `"all"`: test all pairwise group contrasts. • `"manual"`: use `manual_pairs` exactly as supplied.
reference_level	Optional reference level used when `contrast_type = "reference"`. By default the earliest or first group is used.
manual_pairs	Optional two-column matrix or data frame of manual contrasts with `group1` and `group2` values. Each test is `group1 - group2`.
normalization	Library-size normalization method. Choices are: • `"TMM"`: edgeR TMM normalization. • `"upperquartile"`: edgeR upper-quartile normalization. • `"none"`: no normalization factors beyond raw library sizes.
filter_by_expr	Logical indicating whether to run `edgeR::filterByExpr()`.
robust	Logical passed to `edgeR::glmQLFit()`.
abundance_trend	Logical passed to `edgeR::glmQLFit()`.
fdr_cutoff	FDR cutoff used to mark significant genes.

Value

A list containing the fitted edgeR objects, contrast matrix, result tables, and analysis settings.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60, n_pcs = 10
)
```

```
sce$replicate <- rep(paste0("rep", 1:3), length.out = ncol(sce))

pb <- aggregate_pseudobulk(
    sce,
    sample_cols = c("condition", "timepoint", "replicate"),
    aggregation = "sum",
    min_cells = 5
)
pb <- normalize_pseudobulk(pb, method = "TMM")

# Compare every timepoint against the baseline, adjusting for drug.
de <- run_edger_pairwise_de(
    pb,
    group_col = "timepoint",
    covariates = "condition",
    contrast_type = "reference"
)
names(de$tables)
head(de$tables[["24_vs_0"]][, c("gene", "logFC", "FDR", "significant")])
table(de$combined_table$contrast, de$combined_table$significant)
```

`run_edger_spline_de` *Run pseudobulk spline or time-series differential expression with edgeR*

Description

Fit spline-based edgeR models to identify genes that change over time within each condition, then optionally return fitted expression curves for visualization.

Usage

```
run_edger_spline_de(
  pb,
  time_col = "timepoint",
  condition_col = NULL,
  covariates = NULL,
  assay_name = "counts",
  normalization = c("TMM", "upperquartile", "none"),
  df = 3,
  filter_by_expr = TRUE,
  robust = TRUE,
  abundance_trend = FALSE,
  fdr_cutoff = 0.05,
  return_curve_for = c("significant", "all"),
  curve_grid_length = 200
)
```

Arguments

`pb`	A pseudobulk `SingleCellExperiment`.
`time_col`	Sample metadata column containing numeric or ordered time.

`condition_col` Optional sample metadata column defining conditions to be analysed separately, for example treatment.

`covariates` Optional additive covariates to include in each condition-specific model.

`assay_name` Assay containing pseudobulk counts.

`normalization` Library-size normalization method. Choices are:

- `"TMM"`: edgeR TMM normalization.
- `"upperquartile"`: edgeR upper-quartile normalization.
- `"none"`: no normalization factors beyond raw library sizes.

`df` Degrees of freedom for the natural spline basis.

`filter_by_expr` Logical indicating whether to run `edgeR::filterByExpr()`.

`robust` Logical passed to `edgeR::glmQLFit()`.

`abundance_trend` Logical passed to `edgeR::glmQLFit()`.

`fdr_cutoff` FDR cutoff used to define significant genes.

`return_curve_for` Which genes should receive fitted spline curves. Choices are:

- `"significant"`: return curves only for significant genes.
- `"all"`: return curves for all tested genes.

`curve_grid_length` Number of points used when drawing fitted spline curves.

Value

A list containing condition-specific edgeR fits, tables of temporal DE genes, and fitted curve data for plotting.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60, n_pcs = 10
)
sce$replicate <- rep(paste0("rep", 1:3), length.out = ncol(sce))

pb <- aggregate_pseudobulk(
    sce,
    sample_cols = c("condition", "timepoint", "replicate"),
    aggregation = "sum",
    min_cells = 5
)
pb <- normalize_pseudobulk(pb, method = "TMM")

# Six timepoints per drug, so a natural spline in time is estimable.
spline_de <- run_edger_spline_de(
    pb,
    time_col = "timepoint",
    condition_col = "condition",
    df = 3,
    curve_grid_length = 50
)
names(spline_de$by_condition)
```

```
head(spline_de$combined_table[, c("gene", "FDR", "condition")])
table(
    spline_de$combined_table$condition,
    spline_de$combined_table$significant
)
```

`run_scproka_de` *Run a full scProkaR differential expression workflow*

Description

Aggregate single-cell counts into pseudobulk samples and then dispatch to a pairwise or time-series edgeR workflow.

Usage

```
run_scproka_de(
  sce,
  mode = c("pairwise", "time_series"),
  sample_cols,
  assay_name = "counts",
  aggregation = c("sum", "mean"),
  min_cells = 10L,
  normalization = c("TMM", "upperquartile", "none"),
  ...
)
```

Arguments

`sce`	A `SingleCellExperiment`.
`mode`	Workflow mode. Choices are: • `"pairwise"`: pseudobulk followed by pairwise edgeR contrasts. • `"time_series"`: pseudobulk followed by spline-based time-series edgeR.
`sample_cols`	Metadata columns used to define pseudobulk samples.
`assay_name`	Assay containing input counts.
`aggregation`	Pseudobulk aggregation method. Choices are: • `"sum"`: sum counts within each pseudobulk sample. • `"mean"`: average counts within each pseudobulk sample.
`min_cells`	Minimum number of cells required per pseudobulk sample.
`normalization`	Pseudobulk normalization method. Choices are: • `"TMM"`: edgeR TMM normalization. • `"upperquartile"`: edgeR upper-quartile normalization. • `"none"`: no normalization factors beyond raw library sizes.
`...`	Additional arguments passed to `run_edger_pairwise_de()` or `run_edger_spline_de()`, depending on `mode`.

Value

A list with the pseudobulk object, the DE result, and workflow settings.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60, n_pcs = 10
)
sce$replicate <- rep(paste0("rep", 1:3), length.out = ncol(sce))

# Pseudobulk aggregation plus pairwise edgeR contrasts in one call.
pairwise_res <- run_scproka_de(
    sce,
    mode = "pairwise",
    sample_cols = c("condition", "timepoint", "replicate"),
    min_cells = 5,
    group_col = "timepoint",
    covariates = "condition"
)
dim(pairwise_res$pseudobulk)
names(pairwise_res$de_result$tables)

# The same pseudobulk design analysed as a spline time series.
ts_res <- run_scproka_de(
    sce,
    mode = "time_series",
    sample_cols = c("condition", "timepoint", "replicate"),
    min_cells = 5,
    time_col = "timepoint",
    condition_col = "condition",
    curve_grid_length = 50
)
head(ts_res$de_result$combined_table[, c("gene", "FDR", "condition")])
```

`run_tata` *Run the full TATA workflow*

Description

Run Time Aware Trajectory Analysis (TATA) end-to-end on a `SingleCellExperiment`.

Usage

```
run_tata(
  sce,
  dimred = "PCA",
  dims = NULL,
  time_col = "timepoint",
  use_time = TRUE,
  cluster_col = "cluster",
  k = 15,
  cluster_method = c("louvain", "leiden", "walktrap"),
  alpha = 1,
  beta = 1,
  direction_threshold = 0.1,
  prune_threshold = 0.01,
```

```
  time_weight_mode = c("directional_confidence", "ordered"),
  do_pseudotime = TRUE,
  root_cluster = NULL,
  refine_clusters = TRUE,
  refine_min_cells = NULL,
  refine_min_split_fraction = 0.2,
  do_branch_probs = TRUE,
  expected_branches = NULL,
  n_pcs_if_missing = 20
)
```

Arguments

sce	A SingleCellExperiment.
dimred	Reduced dimension to use for graph construction.
dims	Optional subset of dimensions.
time_col	Column containing experimental time labels.
use_time	Logical indicating whether known experimental time should be used during graph abstraction. Set FALSE to run a topology-only TATA variant.
cluster_col	Name of the column where inferred cluster labels will be stored.
k	Number of neighbours for the cell-cell kNN graph.
cluster_method	Graph clustering method. Choices are: • "louvain": fast modularity-based clustering and the default option. • "leiden": Leiden community detection if available in the installed igraph; otherwise TATA falls back to Louvain. • "walktrap": random-walk clustering that can give slightly coarser states.
alpha	Exponent for topology weights.
beta	Exponent for time weights.
direction_threshold	Threshold for direction assignment.
prune_threshold	Minimum TATA weight to retain an abstract graph edge.
time_weight_mode	How the time signal contributes to the final edge score. Choices are: • "directional_confidence": use direction strength regardless of whether the label ordering is a -> b or b -> a; this is the recommended option for multi-branch graphs. • "ordered": use the original ordered-pair score (flow_score + 1) / 2, which is stricter but more sensitive to the arbitrary ordering of cluster labels.
do_pseudotime	Logical indicating whether to compute pseudotime.
root_cluster	Optional root cluster for pseudotime.
refine_clusters	Logical indicating whether to apply a conservative temporal refinement step after graph clustering.
refine_min_cells	Minimum cluster size considered for temporal refinement.

refine_min_split_fraction
: Minimum fraction of cells required in both temporal subclusters for a split to be accepted.

do_branch_probs
: Logical indicating whether to compute terminal-branch probabilities and entropy/plasticity scores.

expected_branches
: Optional expected number of terminal branches. If supplied, TATA uses it to guide terminal-state selection for downstream branch probabilities and branch assignment.

n_pcs_if_missing
: Number of PCs to compute if dimred is missing.

Value

A list containing the updated sce, cell graph, adjacency matrix, cluster graph, cluster edge table, cluster pseudotime, cell pseudotime, and parameters used.

Examples

```
sce <- simulate_tata_sce(n_cells = 1000, n_features = 60)
tata <- run_tata(sce, dimred = "PCA", k = 12)
names(tata)
```

---

sce1 *Example bacterial single-cell dataset*

Description

sce1 is a packaged SingleCellExperiment example dataset for demonstrating the core scProkaR workflow. It contains one observation per bacterium and includes three treatment groups (control, PMB0.5, and PMB2) measured across multiple time points.

Usage

```
sce1
```

Format

A SingleCellExperiment with a counts assay and cell metadata columns:

sample original sample identifier

clusters pre-existing cluster label

treatments treatment group

timepoints experimental sampling time

Source

Internal example dataset distributed with scProkaR.

Examples

```
data("sce1", package = "scProkaR")

# Size of the object and the assays it carries
dim(sce1)
SummarizedExperiment::assayNames(sce1)

# Cell metadata used by the scProkaR workflow
colnames(SummarizedExperiment::colData(sce1))
table(sce1$treatments, sce1$timepoints)

# Inspect counts on a small subset of cells
sub <- sce1[, seq_len(300)]
lib_size <- Matrix::colSums(SingleCellExperiment::counts(sub))
summary(lib_size)
```

---

select_tata_branch_cells

Select cells from a TATA branch

Description

Select cells that belong to a chosen terminal branch using either hard branch assignment or a probability threshold.

Usage

```
select_tata_branch_cells(
  tata_result,
  branch,
  selection_mode = c("soft", "hard"),
  probability_threshold = 0.25
)
```

Arguments

`tata_result`	Result list returned by `run_tata()`.
`branch`	Terminal branch or terminal cluster to select.
`selection_mode`	How to define selected cells. Choices are: • `"soft"`: select cells whose probability for the branch is at least `probability_threshold`. • `"hard"`: select cells whose hard TATA branch assignment equals `branch`.
`probability_threshold`	Minimum branch probability required for `selection_mode = "soft"`.

Value

A data frame with cell ids, branch probabilities, and a logical `selected` column.

Examples

```
set.seed(1)
sce <- simulate_tata_multidrug_sce(
    n_cells = 600, n_features = 60, n_pcs = 10
)
tata <- run_tata(sce,
    dimred = "PCA", time_col = "timepoint", k = 15,
    expected_branches = 3
)
branch <- tata$terminal_clusters[1]
soft <- select_tata_branch_cells(tata,
    branch = branch,
    selection_mode = "soft", probability_threshold = 0.3
)
head(soft)
table(soft$selected)
hard <- select_tata_branch_cells(tata,
    branch = branch,
    selection_mode = "hard"
)
table(hard$selected)
```

---

simulate_tata_multidrug_sce

Simulate a multi-drug branching TATA dataset

Description

Simulate a `SingleCellExperiment` with one shared origin trajectory that can diverge into three drug-associated fates. The three drug-associated branches emerge at different real treatment times, making the dataset useful for checking whether TATA uses time information rather than feature similarity alone when orienting cluster-level transitions.

Usage

```
simulate_tata_multidrug_sce(n_cells = 5000, n_features = 100, n_pcs = 20)
```

Arguments

`n_cells`	Number of cells to simulate.
`n_features`	Number of features to simulate.
`n_pcs`	Number of principal components to store.

Details

The simulated design uses treatment times in hours:

- `drug_A` branch emerges early, around 24 hours
- `drug_C` branch emerges around 48 hours
- `drug_B` branch emerges late, around 96 hours

The geometry is intentionally center-rooted rather than tip-rooted: drug_A and drug_B extend in opposite directions from a shared middle state, and drug_C diverges along a third arm. This is meant to stress-test whether time labels can recover direction when the root is not visually obvious from topology alone.

Value

A SingleCellExperiment.

Examples

```
sce <- simulate_tata_multidrug_sce(n_cells = 1000, n_features = 80)
table(sce$condition, sce$timepoint)
```

---

simulate_tata_sce *Simulate a divergent TATA dataset*

Description

Simulate a SingleCellExperiment containing single-bacterium observations on a divergent two-way trajectory. Time 0 is centred at the origin of the latent trajectory, and later timepoints move away from that central root into two branches. This makes the simulated dataset useful for demonstrating TATA on a divergent trajectory where the earliest state appears in the middle of a UMAP.

Usage

```
simulate_tata_sce(n_cells = 5000, n_features = 100, n_pcs = 20)

simulate_tata_divergent_sce(n_cells = 5000, n_features = 100, n_pcs = 20)
```

Arguments

n_cells	Number of cells to simulate.
n_features	Number of features to simulate.
n_pcs	Number of principal components to store.

Details

The returned object contains:

- counts and logcounts
- colData with cell_id, timepoint, condition, simulated_state, and simulated_branch
- reducedDims(sce)$PCA
- reducedDims(sce)$UMAP
- reducedDims(sce)$truth

Value

A SingleCellExperiment.

Examples

```
sce <- simulate_tata_sce(n_cells = 1000, n_features = 60)
dim(sce)
```

`simulate_tata_spiral_sce` *Simulate an extreme spiral-plus-divergent TATA dataset*

Description

Simulate a stress-test `SingleCellExperiment` whose `UMAP` embedding follows a spiral-like manifold that transitions into two divergent tails. This is useful for visually stress-testing TATA in an intentionally extreme geometry.

Usage

```
simulate_tata_spiral_sce(n_cells = 5000, n_features = 100, n_pcs = 20)
```

Arguments

`n_cells`	Number of cells to simulate.
`n_features`	Number of features to simulate.
`n_pcs`	Number of principal components to store.

Details

The embedding is constructed directly in UMAP-like space and paired with smooth expression programs so that neighboring cells along the path remain transcriptionally similar.

Value

A `SingleCellExperiment`.

Examples

```
sce <- simulate_tata_spiral_sce(n_cells = 1000, n_features = 80)
SingleCellExperiment::reducedDimNames(sce)
```

Index

∗ **datasets**
    sce1, 60
∗ **internal**
    scProkaR-package, 3

adjacent_timepoint_auc, 4
aggregate_pseudobulk, 5
anticausal_edge_mass, 6

BenchmarkIntegration, 8
BenchmarkIntegration(), *48*
branch_onset_metrics, 9
build_knn_graph, 10
build_tata_graph, 11

cluster_graph_states, 12
cluster_tata_gene_trends, 13
compute_tata_branch_probabilities, 14
compute_tata_gene_trends, 15
compute_tata_pseudotime, 16
compute_time_weights, 17
compute_topology_weights, 18
CreateBacObject, 19
CreateBacObject(), *27*

directed_reachability_concordance, 21

filter_pseudobulk_samples, 23
FilterBacCells, 22

IntegrateBacData, 24
IntegrateBacData(), *52*

local_temporal_order_metrics, 26

MergeBacObjects, 27

normalize_pseudobulk, 28

orient_pseudotime_to_time, 29

plot_pairwise_de_ma, 32
plot_pairwise_de_volcano, 34
plot_tata_branch_confusion, 35
plot_tata_branch_probabilities, 36
plot_tata_branch_selection, 37
plot_tata_cluster_graph, 38
plot_tata_edge_causality, 40
plot_tata_gene_trend_heatmap, 41
plot_tata_gene_trends, 41
plot_tata_results, 42
plot_tata_terminal_probabilities, 43
plot_tata_time_calibration, 44
plot_tata_trajectory_embedding, 45
plot_time_series_deg_curves, 46
PlotIntegrationOverview, 30
PlotReduction, 31
PlotReduction(), *30*

RegisterExistingIntegrationEmbeddings, 48
RegisterExistingIntegrationEmbeddings(), *8*
RegisterIntegrationEmbedding, 49
RegisterIntegrationEmbedding(), *8*, *52*
run_edger_pairwise_de, 53
run_edger_pairwise_de(), *32*, *34*
run_edger_spline_de, 55
run_scproka_de, 57
run_tata, 58
RunBacQC, 50
RunIntegratedClustering, 51
RunIntegratedUMAP, 52
RunIntegratedUMAP(), *31*

sce1, 60
scProkaR *(*scProkaR-package*)*, 3
scProkaR-package, 3
select_tata_branch_cells, 61
simulate_tata_divergent_sce *(*simulate_tata_sce*)*, 63
simulate_tata_multidrug_sce, 62
simulate_tata_sce, 63
simulate_tata_spiral_sce, 64