Package: scECODA (via r-universe)

May 30, 2026

Title Single-Cell Exploratory Compositional Data Analysis

Version 1.0.0

Description The scECODA R package provides a complete workflow for the analysis and visualization of compositional data, primarily focusing on cell type proportions derived from single-cell data. It implements specialized methods, such as the Centered Log-Ratio (CLR) transformation, to properly analyze proportional data while avoiding the biases introduced by the compositional constraint. The package encapsulates data management, transformation, and analysis into a single SummarizedExperiment object, offering downstream tools for dimensionality reduction via PCA, calculating critical metrics like the Adjusted Rand Index (ARI) and Modularity to quantify sample grouping quality, and generating high-quality visualizations like heatmaps and scatter plots.

biocViews Software, SingleCell, Transcriptomics, CellBasedAssays, Normalization, Preprocessing, Visualization, Clustering, DimensionReduction, FeatureExtraction, PrincipalComponent

BugReports https://github.com/carmonalab/scECODA/issues

License GPL-3 + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.5.0)

VignetteBuilder knitr

Suggests Seurat (>= 5.0.0), igraph, knitr, rmarkdown, BiocStyle, testthat, scRNAseq

Config/testthat/edition 3

URL https://github.com/carmonalab/scECODA

Imports BiocGenerics, cluster, corrplot, DESeq2, dplyr, factoextra (>= 2.0.0), ggplot2, ggpubr, ggrepel, gtools, Matrix, mclust, methods, pheatmap, plotly, rlang, rstatix, S4Vectors, stringr, SummarizedExperiment (>= 1.34.0), tidyr, vegan

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev zlib1g-dev

Repository https://bioc-release.r-universe.dev

Date/Publication 2026-04-28 13:07:02 UTC

RemoteUrl https://github.com/bioc/scECODA

RemoteRef RELEASE_3_23

RemoteSha 7eb703af84625cbe3e23ff536f29c5d589bcbc4e

Contents

`calc_anosim` *Analysis of Similarities (ANOSIM) R score*

Description

Calculates the ANOSIM R-statistic to test whether there is significant separation between two or more groups (defined by `labels`) based on the sample distance matrix (`dist_mat`).

Usage

```
calc_anosim(dist_mat, labels, permutations = 99, parallel = 1, digits = 3)
```

Arguments

`dist_mat`	Sample distance matrix.
`labels`	Vector of factors or character strings. The grouping variable (the known cluster assignments) for each row in the matrix.
`permutations`	Integer (default: `99`). The number of permutations to use when calculating the ANOSIM R-statistic and p-value.
`parallel`	Integer (optional, default: `1`). The number of parallel processes/cores to use for the permutation testing.
`digits`	Integer (default: `3`). The number of decimal places to round the R-statistic.

Details

ANOSIM compares the mean of rank dissimilarities between groups to the mean of rank dissimilarities within groups. The R-statistic ranges from -1 to 1:

- An R value close to **1** indicates clear separation of groups.
- An R value close to **0** indicates that the separation is no greater than expected by chance (i.e., poor separation).
- An R value close to **-1** indicates that within-group dissimilarities are greater than between-group dissimilarities (a very rare result).

Value

A numeric value representing the **ANOSIM R-statistic**.

Examples

```
library(SummarizedExperiment)
data(example_data)
se <- ecoda(
    data = example_data$GongSharma_full$cell_counts_highresolution,
    metadata = example_data$GongSharma_full$metadata
)
```

```
# Extract necessary components
dist_mat <- dist(t(assay(se, "clr")))
labels <- colData(se)$subject.cmv

# Run the calculation
calc_anosim(dist_mat, labels, parallel = 1)
```

`calc_ari` *Calculate Adjusted Rand Index (ARI)*

Description

Calculates the Adjusted Rand Index (ARI) to measure the agreement between the known cluster assignments (`labels`) and the cluster assignments derived from two unsupervised clustering methods: Hierarchical Clustering (`hclust`) and Partitioning Around Medoids (`pam`).

Usage

```
calc_ari(dist_mat, labels, nclusts = NULL, digits = 3, return_mean = TRUE)
```

Arguments

`dist_mat`	Sample distance matrix.
`labels`	Vector of factors or character strings. The true or known cluster assignments for each row in the matrix.
`nclusts`	Integer (optional, default: `NULL`). The target number of clusters (`k`) to use for `hclust` and `pam`. If `NULL`, it defaults to the number of unique levels in `labels`.
`digits`	Integer (default: `3`). The number of decimal places to round the result.
`return_mean`	Logical (default: `TRUE`). If `TRUE`, returns the mean of the ARI scores from `hclust` and `pam`. If `FALSE`, returns a named list with both individual scores.

Value

A numeric value (mean ARI) or a list of two ARI scores. ARI ranges from -1 (disagreement) to +1 (perfect agreement).

Examples

```
library(SummarizedExperiment)
data(example_data)
se <- ecoda(
    data = example_data$GongSharma_full$cell_counts_highresolution,
    metadata = example_data$GongSharma_full$metadata
)

# Extract necessary components
dist_mat <- dist(t(assay(se, "clr")))
labels <- colData(se)$subject.cmv
```

```
# Run the calculation
calc_ari(dist_mat, labels)
```

`calc_clr` *Perform the Centered Log-Ratio (CLR) transformation.*

Description

The CLR transformation is a common technique used for compositional data analysis, mapping the data from the simplex to Euclidean space. It is defined as the log of the ratio between each component and the geometric mean of all components in that sample. **Note:** This function assumes the input data is strictly positive (i.e., does not contain zeros). Use an imputation or pseudocount method prior to this function if zeros are present.

Usage

```
calc_clr(df)
```

Arguments

`df`	A data frame or matrix of strictly positive relative frequencies or counts (samples/observations as columns, components as rows).

Value

A data frame of the same dimensions containing the CLR-transformed values.

Examples

```
freq_imp <- data.frame(A = c(10.1, 89.9), B = c(50.1, 49.9))
calc_clr(freq_imp)
```

`calc_freq` *Calculate relative frequencies (percentages) column-wise.*

Description

This function takes a matrix or data frame of counts and transforms each column (assumed to be a sample or observation) into relative frequencies, expressing each component as a percentage of the column's total sum.

Usage

```
calc_freq(df)
```

Arguments

df	A data frame or matrix of counts (samples/observations as columns, components as rows). Must contain only numeric, non-negative values.

Value

A data frame of the same dimensions, where each column sums to 100.

Examples

```
counts <- data.frame(A = c(10, 90), B = c(50, 50))
calc_freq(counts)
```

---

calc_modularity *Calculate Adjusted Modularity Score*

Description

Calculates the Modularity score for a given clustering (`labels`) based on a Shared Nearest Neighbor (SNN) graph.The score is adjusted by the theoretical maximum modularity for the number of groups to always be between -0.5 (poor clustering) and +1 (excellent clustering)).

Usage

```
calc_modularity(dist_mat, labels, knn_k = 3, digits = 3)
```

Arguments

dist_mat	Sample distance matrix.
labels	Vector of factors or character strings. The cluster assignments for each row in `dist_mat`.
knn_k	Integer (optional, default: `NULL`). The number of nearest neighbors (`k`) used for SNN graph construction. If `NULL`, it defaults to `max(3, round(sqrt(N)))`, where `N` is the number of samples.
digits	Integer (default: `3`). The number of decimal places to round the final adjusted score.

Value

A numeric value representing the adjusted modularity score, ranging from -0.5 to 1.0.

References

Brandes, Ulrik, et al. "On finding graph clusterings with maximum modularity." European Symposium on Algorithms. Springer, Berlin, Heidelberg, 2007.

Examples

```
library(SummarizedExperiment)
data(example_data)
se <- ecoda(
    data = example_data$GongSharma_full$cell_counts_highresolution,
    metadata = example_data$GongSharma_full$metadata
)

# Extract necessary components
dist_mat <- dist(t(assay(se, "clr")))
labels <- colData(se)$subject.cmv

# Run the calculation
## Not run:
calc_modularity(dist_mat, labels)

## End(Not run)
```

`calc_sil` *Calculate Average Silhouette Width*

Description

Calculates the average silhouette width for a given feature matrix, using pre-defined cluster assignments (`labels`). This metric assesses the quality of the clustering.

Usage

```
calc_sil(dist_mat, labels, digits = 3)
```

Arguments

`dist_mat`	Sample distance matrix.
`labels`	Vector of factors or character strings. The cluster assignments for each row in `dist_mat` (i.e., the grouping variable).
`digits`	Integer (default: `3`). The number of decimal places to round the final score.

Value

A numeric value representing the mean silhouette width, typically ranging from -1 (poor clustering) to +1 (excellent clustering).

Examples

```
library(SummarizedExperiment)
data(example_data)
se <- ecoda(
    data = example_data$GongSharma_full$cell_counts_highresolution,
    metadata = example_data$GongSharma_full$metadata
)

# Extract necessary components
dist_mat <- dist(t(assay(se, "clr")))
labels <- colData(se)$subject.cmv

# Run the calculation
calc_sil(dist_mat, labels)
```

`calculate_pseudobulk` *Calculate Pseudobulk from Count Matrix*

Description

This function aggregates single-cell count data into a "pseudobulk" matrix by summing the counts for all cells belonging to the same sample ID. It is robust to both dense and sparse count matrices. It also includes filtering logic to exclude samples that do not meet a minimum cell count threshold.

Usage

```
calculate_pseudobulk(count_matrix, sample_ids, min_cells = 10)
```

Arguments

`count_matrix`	A gene x cell count matrix (can be a dense matrix or sparse Matrix). Gene identifiers should be row names and cell barcodes should be column names.
`sample_ids`	A vector of sample identifiers, one for each column (cell) in `count_matrix`. The length must equal `ncol(count_matrix)`. Must not contain `NA` values.
`min_cells`	Minimum number of cells required per sample (default: 1). Samples with fewer cells than this threshold will be excluded from the final pseudobulk matrix.

Value

A gene x sample pseudobulk count matrix. The columns correspond to the unique sample IDs, and the rows correspond to the genes.

Examples

```
nrow <- 100
ncol <- 100
vals <- nrow * ncol
mat <- round(matrix(exp(rpois(vals, lambda = 3)), nrow = nrow, ncol = ncol))
rownames(mat) <- paste0("Gene", seq_len(ncol))
colnames(mat) <- paste0("Cell", seq_len(nrow))
ids <- rep(c("Sample1", "Sample2"), each = nrow / 2)
pb <- calculate_pseudobulk(count_matrix = mat, sample_ids = ids)
```

---

compute_KNN_from_dist *Compute K-Nearest Neighbors (KNN) from Distance Matrix*

Description

Identifies the indices of the K-nearest neighbors for each sample based on a pre-computed distance matrix.

Usage

```
compute_KNN_from_dist(dist_mat, knn_k)
```

Arguments

dist_mat	A square distance matrix or an object of class dist.
knn_k	Integer. The number of neighbors (k) to return for each sample.

Value

A matrix with nrow(dist_mat) rows and knn_k columns, where each row contains the indices of the nearest neighbors.

---

compute_snn_graph *Compute Shared Nearest Neighbor (SNN) Graph*

Description

Constructs a graph where nodes represent samples and edges are weighted by the number of shared nearest neighbors (SNN) between them.

Usage

```
compute_snn_graph(knn)
```

Arguments

knn	A matrix of nearest neighbor indices, typically generated by compute_KNN_from_dist.

Value

An igraph graph object where edge weights correspond to the count of shared neighbors between nodes.

`create_long_data` *Reshapes ECODA data into a long format for plotting and analysis.*

Description

This function takes either the relative abundance (`freq`) or CLR-transformed abundance (`clr`) matrix from an SummarizedExperiment object, converts it from a wide (cell types x samples) to a long (sample, cell type, value) format for plotting, and optionally joins it with a specified column from the sample metadata.

Usage

```
create_long_data(
  se,
  assay = c("clr", "freq", "asin_sqrt", "clr_hvc"),
  label_col = NULL
)
```

Arguments

`se`	An initialized SummarizedExperiment object.
`assay`	Character string (default: `"clr"`). The name of the assay in the `SummarizedExperiment` object to use for plotting. Must be one of: `"clr"` (CLR-transformed abundances, default), `"clr_hvc"` (CLR-transformed abundances of only the most highly variable cell types (HVCs)), `"counts"` (raw counts), `"counts_imp"` (imputed counts), `"freq"` (relative frequencies), `"freq_imp"` (imputed frequencies), or `"asin_sqrt"` (arcsin-square root transformed data).
`label_col`	Character string (optional, default: `NULL`). The name of a column in `colData(se)` to merge into the long data frame (e.g., "Disease_State" or "Batch"). If `NULL`, only the abundance data and sample/celltype IDs are returned.

Value

A tidy, long format data frame with columns:

- `sample_id`: Sample identifier.
- `celltype`: Cell type name.
- `rel_abundance` or `clr_abundance`: The quantitative value depending on the chosen `data_slot`.
- `...`: Additional column specified by `label_col` (if provided).

See Also

SummarizedExperiment

deseq2_normalize *DESeq2 Normalization of Pseudobulk Data*

Description

This function normalizes a gene x sample pseudobulk count matrix using the Variance Stabilizing Transformation (VST) from the DESeq2 package. It estimates size factors and variance for all genes, performs the VST, and then subsets the results to include only highly variable genes (HVCs), either specified by the user or automatically selected based on variance.

Usage

```
deseq2_normalize(
  pb,
  metadata = NULL,
  deseq2_design = NULL,
  hvg = NULL,
  nvar_genes = 2000
)
```

Arguments

pb	A gene x sample pseudobulk count matrix (Genes as rows, Samples as columns). Must contain non-negative integer counts.
metadata	A data.frame with sample-level metadata. Row names must exactly match the column names of pb. The function will reorder the metadata to match pb.
deseq2_design	Optional: A formula object specifying the design used by DESeq2 to estimate size factors and variance for the VST. This formula should reference columns in the metadata data frame. If NULL (the default), the minimal design ~ 1 is used. Note: the design will be considered for pseudobulk normalization as the function uses vst(blind = FALSE, ...) internally.
hvg	Optional character vector of gene names to use as highly variable genes. If provided, only these genes will be returned after VST.
nvar_genes	Number of top variable genes to select (default: 2000). Only used if hvg = NULL; the function will select the top nvar_genes by variance after VST.

Details

Note: If deseq2_design is not specified, the VST is performed using a minimal design formula (~ 1) for size factor and variance estimation.

Value

A normalized expression matrix (VST-transformed) with\ **samples as columns** and **genes as rows**.

Examples

```
nrow <- 100
ncol <- 100
vals <- nrow * ncol
mat <- round(matrix(exp(rpois(vals, lambda = 3)), nrow = nrow, ncol = ncol))
rownames(mat) <- paste0("Gene", seq_len(ncol))
colnames(mat) <- paste0("Cell", seq_len(nrow))
ids <- rep(c("Sample1", "Sample2"), each = nrow / 2)
pb <- calculate_pseudobulk(count_matrix = mat, sample_ids = ids)
pb_norm <- deseq2_normalize(pb)
```

`ecoda` *Create an SummarizedExperiment object from various data types*

Description

This is a smart constructor function used to initialize an SummarizedExperiment object. It handles the processing of single-cell objects (`Seurat` or `SingleCellExperiment`) or raw data frames to extract cell type counts, calculate sample metadata, and optionally generate DESeq2-normalized pseudobulk data.

Usage

```
ecoda(
  data = NULL,
  data_is_freq = FALSE,
  metadata = NULL,
  sample_col = NULL,
  celltype_col = NULL,
  get_pb = FALSE,
  variance_explained = 0.5,
  top_n_hvcs = NULL,
  pseudo_count = 0.5,
  pseudo_frac_min = 2/3,
  add_to = NULL
)
```

Arguments

`data`	The primary input, which can be: • A `Seurat` object. • A `SingleCellExperiment` object. • A cell type x sample matrix or data frame (counts or frequencies).
`data_is_freq`	Logical (default: `FALSE`). If `TRUE`, the input `data` is treated as relative frequencies (proportions) rather than raw counts.

metadata	A data frame containing sample-level metadata. Required if data is a matrix/data frame. If data is a single-cell object, metadata is extracted automatically.
sample_col	The metadata column name defining unique sample IDs. Required if data is a single-cell object.
celltype_col	The metadata column name defining cell type annotations. Required if data is a single-cell object.
get_pb	Logical (default: FALSE). If TRUE, calculates and stores DESeq2-normalized pseudobulk data in metadata(se)$pb
variance_explained	Numeric (default: 0.5). Proportion of variance used to determine the number of highly variable cell types (HVCs).
top_n_hvcs	Integer (optional). If provided, specifies the exact number of top HVCs to select, overriding variance_explained.
pseudo_count	Numeric (default: 0.5). Used if rep_method = "counts".
pseudo_frac_min	Numeric (default: 2/3). The multiplier for the minimum non-zero value when rep_method = "frac_min".
add_to	Character string. Should the value be added to "all" cells or just the "zeros"?

Value

A new SummarizedExperiment object.

Examples

```
data(example_data)
Zhang <- example_data$Zhang
counts <- Zhang$cell_counts_lowresolution
freq <- calc_freq(counts)
metadata <- Zhang$metadata

se <- ecoda(data = counts, metadata = metadata)

se <- ecoda(data = freq, metadata = metadata)
```

---

ecoda_helper *Core constructor for SummarizedExperiment objects from count/frequency matrices*

Description

This is the internal engine that initializes an SummarizedExperiment object. It performs zero-imputation, Centered Log-Ratio (CLR) transformation, calculates sample distances, and identifies highly variable cell types (HVCs).

Usage

```
ecoda_helper(
  data = NULL,
  data_is_freq,
  variance_explained = 0.5,
  top_n_hvcs = NULL,
  pseudo_count = 0.5,
  pseudo_frac_min = 2/3,
  add_to = c("all", "zeros")
)
```

Arguments

`data`	A matrix or data frame where **columns are samples** and **rows are cell types**.
`data_is_freq`	Logical. If `TRUE`, `data` is treated as proportions (0-1 or 0-100). If `FALSE`, treated as raw integer counts.
`variance_explained`	Numeric (default: 0.5). Target variance for HVCs.
`top_n_hvcs`	Integer (optional). Number of top HVCs to select.
`pseudo_count`	Numeric (default: 0.5). Used if `rep_method = "counts"`.
`pseudo_frac_min`	Numeric (default: 2/3). The multiplier for the minimum non-zero value when `rep_method = "frac_min"`.
`add_to`	Character string. Should the value be added to `"all"` cells or just the `"zeros"`?

Value

A fully initialized SummarizedExperiment object.

`example_data` *Example Data for scECODA*

Description

A collection of single-cell compositional data from multiple studies, structured to demonstrate the full scECODA workflow.

Usage

```
data(example_data)
```

Format

A named `list` where each element corresponds to a different study (e.g., `$Adams`). Each study element is a `list` containing the following:

- `cell_counts_highresolution`: Data frame of cell type counts at high resolution annotation (samples as rows, cell types as columns).
- `cell_counts_lowresolution`: Data frame of cell type counts at low resolution annotation (samples as rows, cell types as columns).
- `metadata`: Data frame of sample-level metadata (samples as rows).
- `main_biologicalcondition_columnname`: Character string, indicating the main biological condition column name in the metadata.

Source

Data originally derived from multiple public single-cell datasets and re-processed as described in the accompanying scECODA manuscript (Halter et al., *in preparation*).

`find_hvcs` *Identifies and stores Highly Variable Cell Types (HVCs) in an SummarizedExperiment object.*

Description

This function calculates the variance of each cell type across all samples based on the Centered Log-Ratio (CLR) transformed data. It then selects the subset of highly variable cell types (HVCs) that collectively explain a specified proportion of the total variance, or a fixed number of top cell types.

Usage

```
find_hvcs(se, variance_explained = 0.5, top_n_hvcs = NULL)
```

Arguments

`se`	An initialized SummarizedExperiment object containing the CLR-transformed data in the `clr` assay.
`variance_explained`	Numeric (default: 0.5). The target cumulative proportion of total variance to be explained by the selected HVCs. The function stops selecting cell types once this threshold is met.
`top_n_hvcs`	Integer (optional). If provided, this overrides `variance_explained` and selects exactly the top N cell types with the highest variance.

Value

The updated SummarizedExperiment object with the following metadata populated:

- `celltype_variances`: Data frame of cell type variances.
- `variance_explained`: The exact variance proportion captured by the selected HVCs.
- `top_n_hvcs`: The number of HVCs selected.
- `hvcs`: Character vector of the names of the selected HVCs.

See Also

SummarizedExperiment, `get_celltype_variances`, `get_hvcs`

Examples

```
data(example_data)
se <- ecoda(
    data = example_data$GongSharma_full$cell_counts_highresolution,
    metadata = example_data$GongSharma_full$metadata
)

# Select HVCs that explain 75% of the total variance
se <- find_hvcs(se, variance_explained = 0.75)

# Select exactly the top 5 most variable cell types
se <- find_hvcs(se, top_n_hvcs = 5)
```

`get_celltype_counts` *Get the cell type counts from a long data frame (e.g. seurat object metadata) where each cell is a row.*

Description

Get the cell type counts from a long data frame (e.g. seurat object metadata) where each cell is a row.

Usage

```
get_celltype_counts(cell_data_df, sample_col, celltype_col)
```

Arguments

`cell_data_df`	A data frame where each row represents a single cell, and columns contain cell-level information, including sample ID and cell type annotation.
`sample_col`	The column that defines the sample ID for each cell
`celltype_col`	The column that defines the cell type annotation for each cell

Value

A data frame with cell types as rows and samples as columns, containing the count of each cell type per sample.

Examples

```
# Create example data frame
cell_data_df <- data.frame(
    Cell_ID = paste0("C", seq(10)),
    Sample_Name = c(rep("S1", 5), rep("S2", 5)),
    Cluster_Annotation = factor(c(
        "B_Cell", "T_Cell", "B_Cell", NA, "T_Cell",
        "T_Cell", "Macrophage", "B_Cell", "T_Cell", "T_Cell"
    ))
)
#
# Calculate cell type counts per sample
celltype_counts <- get_celltype_counts(
    cell_data_df = cell_data_df,
    sample_col = "Sample_Name",
    celltype_col = "Cluster_Annotation"
)
#
print(celltype_counts)
# Note how the NA cell type count is handled and renamed to "NA".
```

---

get_celltype_variances

Calculates the variance of cell types across samples.

Description

This function takes the Centered Log-Ratio (CLR) transformed cell type abundance data from an SummarizedExperiment object, calculates the mean CLR abundance and variance for each cell type. It also calculates the cumulative variance explained by the cell types when ranked by variance.

Usage

```
get_celltype_variances(se, descending = TRUE)
```

Arguments

se	An initialized SummarizedExperiment object containing CLR-transformed data in the clr assay
descending	Logical (default: TRUE). If TRUE, the returned data frame is sorted by variance in descending order.

Value

A data frame where each row is a cell type, containing:

- `celltype`: The name of the cell type.
- `avg_clr_abundance`: The mean CLR value for that cell type across all samples.
- `Variance`: The variance of the CLR values for that cell type across all samples.
- `cumulative_variance`: Cumulative sum of the variance, ranked by variance.
- `variance_exp`: The proportion of total variance explained cumulatively.

See Also

SummarizedExperiment, `plot_varmean`

Examples

```
data(example_data)
se <- ecoda(
    data = example_data$GongSharma_full$cell_counts_highresolution,
    metadata = example_data$GongSharma_full$metadata
)

# Calculate variances without plotting, sorted by ascending variance
df_var_asc <- get_celltype_variances(
    se,
    descending = FALSE
)
```

`get_ecoda_assay` *Helper to get assay for ecoda SummarizedExperiment objects*

Description

Because not all data can be stored in assays (e.g. pseudobulk and clr_hvc have different number of features (rows) than counts etc.)

Usage

```
get_ecoda_assay(se, assay)
```

Arguments

`se`	A SummarizedExperiment object.
`assay`	Character string, the name of the assay.

Value

A data frame containing the assay data.

get_hvcs *Selects Highly Variable Cell Types (HVCs) based on variance or count threshold.*

Description

This is a utility function that selects the most variable cell types from a variance-ranked data frame. Selection can be controlled either by defining the cumulative variance to be explained or by specifying a fixed number of cell types.

Usage

```
get_hvcs(df_var, variance_explained = 0.5, top_n_hvcs = NULL)
```

Arguments

df_var	A data frame (typically the output of get_celltype_variances) that must contain at least the columns celltype and variance_exp (cumulative variance explained) and must be sorted by variance in descending order.
variance_explained	Numeric (default: 0.5). The cumulative proportion of total variance that the selected HVCs must explain. This parameter is ignored if top_n_hvcs is set.
top_n_hvcs	Integer or Numeric (optional). • If an integer (>= 1), it specifies the exact number of top cell types to select. • If a numeric value between 0 and 1, it specifies the proportion of cell types to select (e.g., 0.1 for the top 10%).

Value

A character vector containing the names of the selected highly variable cell types. The function ensures that at least two cell types are always returned.

See Also

find_hvcs, get_celltype_variances

Examples

```
data(example_data)
ecoda_object <- ecoda(
    data = example_data$GongSharma_full$cell_counts_highresolution,
    metadata = example_data$GongSharma_full$metadata
)

# Calculate variances
df_var <- get_celltype_variances(ecoda_object)

# Select cell types explaining at least 60% of variance:
```

```
hvcs_60 <- get_hvcs(df_var, variance_explained = 0.6)

# Select the top 10 cell types:
hvcs_top10 <- get_hvcs(df_var, top_n_hvcs = 10)

# Select the top 5% of cell types:
hvcs_prop <- get_hvcs(df_var, top_n_hvcs = 0.05)
```

`get_sample_metadata` *Extracts constant metadata for each sample from a cell-level data frame.*

Description

This function identifies columns in a cell-level metadata data frame that have a **constant** value for all cells belonging to the same sample. It aggregates this constant information, returning a new data frame where each row represents a unique sample. Columns that vary within any sample are excluded from the output.

Usage

```
get_sample_metadata(cell_data_df, sample_col)
```

Arguments

`cell_data_df`	A data frame containing cell-level metadata. This should include the sample ID column and all potential metadata columns.
`sample_col`	A character string specifying the name of the column that defines the unique sample ID for each cell (e.g., "Sample_ID").

Value

A new data frame where:

- Each row corresponds to a unique sample from the input data.
- The row names are set to the values of the input `sample_col`.
- Columns contain only the metadata fields that were constant across all cells within **each** sample. The `sample_col` itself is excluded from the final columns but used for row names.

Examples

```
# Assuming you have a data frame 'cell_df'
cell_df <- data.frame(
    Cell_ID = paste0("C", seq(10)),
    Sample_ID = c(rep("S1", 5), rep("S2", 5)),
    Age = c(rep(30, 5), rep(45, 5)),
    Gender = c(rep("M", 5), rep("F", 5)),
    Cell_Type = c(rep("A", 3), rep("B", 2), rep("A", 3), rep("B", 2))
```

```
)

# The 'Age' and 'Gender' columns are constant within each sample (S1 and S2).
# The 'Cell_ID' and 'Cell_Type' columns vary within sample S1 and/or S2.

sample_meta <- get_sample_metadata(cell_df, "Sample_ID")
print(sample_meta)
# Output will have 'Age' and 'Gender' as columns,
# with row names 'S1' and 'S2'.
```

`plot_barplot` *Generates a Stacked Bar Plot of Cell Type Relative Abundance.*

Description

This function visualizes the relative abundance (composition) of cell types across samples or across aggregated groups. It automatically handles data preparation and ordering based on the provided parameters.

Usage

```
plot_barplot(
  se,
  label_col = NULL,
  plot_by = c("sample", "group"),
  custom_sample_order = NULL,
  title = "",
  facet_by_label_col = TRUE
)
```

Arguments

`se`	A SummarizedExperiment object containing cell type relative frequencies in the `freq` assay.
`label_col`	Character string (optional, default: `NULL`). The name of a column in `colData(se)`) used to define grouping or groups (required if `plot_by = "group"`).
`plot_by`	Character string (default: `"sample"`). Specifies whether to plot the relative abundance for each individual sample (`"sample"`) or the average relative abundance aggregated by a group (`"group"`) defined by `label_col`.
`custom_sample_order`	Character vector (optional, default: `NULL`). A vector of sample IDs to enforce a specific order when `plot_by = "sample"`. If `NULL`, samples are ordered first by `label_col` (if provided) and then naturally.
`title`	Character string (default: `""`). The main title for the plot.
`facet_by_label_col`	Logical (default: `TRUE`). If `TRUE` and `plot_by = "sample"`, the plot will be faceted (split) horizontally by the categories in `label_col`.

Value

A ggplot object representing the stacked bar plot.

See Also

create_long_data, SummarizedExperiment

Examples

```
data(example_data)
se <- ecoda(
    data = example_data$Zhang$cell_counts_lowresolution,
    metadata = example_data$Zhang$metadata,
)

plot_barplot(se)

# Plotting average cell type abundance by experimental group
plot_barplot(
    se,
    label_col = "Tissue",
    plot_by = "group",
    title = "Mean Relative Abundance by Condition"
)

# Plotting cell type abundance for each sample separately
plot_barplot(
    se,
    label_col = "Tissue",
    plot_by = "sample",
    title = "Relative Abundance for Each Sample"
)
```

---

plot_boxplot *Generates Boxplots for CLR-transformed Cell Type Abundances with Optional Group Comparison.*

Description

This function visualizes the distribution of CLR-transformed abundance for each cell type using boxplots, optionally splitting the data by a sample metadata column and performing statistical tests for comparison between groups.

Usage

```
plot_boxplot(
  se,
  assay = c("clr", "clr_hvc", "asin_sqrt"),
  label_col = NULL,
```

```
  selected_celltypes = NULL,
  title = "",
  stat_method = "wilcox.test",
  paired = FALSE,
  signif_label = c("p.signif", "p.format")
)
```

Arguments

se	An SummarizedExperiment object containing the CLR-transformed abundances in the clr.
assay	Character string (default: "clr"). The name of the assay in the SummarizedExperiment object to use for plotting. Must be one of: "clr" (CLR-transformed abundances, default), "clr_hvc" (CLR-transformed abundances of only the most highly variable cell types (HVCs)) or "asin_sqrt" (arcsin-square root transformed data).
label_col	Character string (optional, default: NULL). The name of a column in colData(se) used to define groups for comparison. If NULL, a single boxplot is generated per cell type.
selected_celltypes	Specify selected celltypes you want to plot instead of plotting boxplots for all. Note: this does not recalculate CLR-values on subset.
title	Character string (default: ""). The main title for the plot.
stat_method	Character string (default: "wilcox.test"). The statistical method used for comparisons between 2 groups (e.g., "t.test", "wilcox.test"). Note: This is overridden by "kruskal.test" for 3+ groups.
paired	Logical (default: FALSE). If TRUE, performs a paired statistical test (e.g., paired t-test or paired Wilcoxon test). Only applicable for 2-group comparisons.
signif_label	Character string (default: "p.signif"). Controls how p-values are displayed for 2-group comparisons (e.g., "p.signif" for stars, "p.format" for numeric p-value).

Details

Statistical Test Logic:

- If the number of groups (label_col levels) is **2**, the function uses the specified stat_method (default: "wilcox.test") and displays pairwise significance labels or stars.
- If the number of groups is **3 or more**, and stat_method is "wilcox.test" or "t.test", the function automatically switches to the global non-parametric test, **"kruskal.test"**, and displays the overall p-value for the comparison across all groups.

Value

A ggplot object (enhanced by ggpubr) representing the CLR abundance boxplot, including jittered data points and dynamic significance information (pairwise stars for 2 groups, overall p-value for 3+ groups).

See Also

create_long_data, SummarizedExperiment

Examples

```
data(example_data)
se <- ecoda(
    data = example_data$Zhang$cell_counts_lowresolution,
    metadata = example_data$Zhang$metadata,
)

# 1. Boxplots for CLR abundance without grouping (no stats calculated):
plot_boxplot(se)

# 2. Boxplots grouped by 'Treatment' (2 groups) and applying Wilcoxon test:
plot_boxplot(
    se,
    label_col = "Tissue",
    stat_method = "wilcox.test",
    title = "CLR Abundance by Tissue (with Wilcoxon Test)"
)
```

---

plot_corr *Plot Cell Type Correlation Matrix*

Description

Calculates the pairwise Pearson correlation matrix for all cell types (columns) using the Centered Log-Ratio (CLR) transformed abundances stored in `assay(se, "clr")`. It then visualizes this matrix as a heatmap using `corrplot::corrplot`.

Usage

```
plot_corr(
  se,
  assay = c("clr", "counts", "counts_imp", "freq", "freq_imp", "asin_sqrt", "clr_hvc",
    "pb"),
  order = "hclust",
  hclust.method = "ward.D2",
  ...
)
```

Arguments

se	A SummarizedExperiment object.

assay	Character string (default: "clr"). The name of the assay in the SummarizedExperiment object to use for the correlation plot Must be one of: "clr" (CLR-transformed abundances, default), "clr_hvc" (CLR-transformed abundances of only the most highly variable cell types (HVCs)), "pb" (pseudobulk gene expression), "counts" (raw counts), "counts_imp" (imputed counts), "freq" (relative frequencies), "freq_imp" (imputed frequencies), or "asin_sqrt" (arcsin-square root transformed data).
order	Character string (default: "hclust"). The ordering method for the correlation matrix. Common options include: • "original" (no reordering) • "hclust" (hierarchical clustering) • "FPC" (first principal component order)
hclust.method	Character string (default: "ward.D2"). The hierarchical clustering method to use if order is set to "hclust".
...	Additional arguments passed to corrplot::corrplot for plot customization (e.g., method, type, tl.col).

Details

The function uses the CLR matrix, where high correlation between two cell types suggests they vary together across samples, indicating potential co-occurrence or co-regulation.

Value

A plot object generated by corrplot::corrplot, which is a base R plot or a grid object depending on the corrplot version and options.

Examples

```
data(example_data)
# Example of a large cohort with 868 samples and 69 cell types
se <- ecoda(
    data = example_data$GongSharma_full$cell_counts_highresolution,
    metadata = example_data$GongSharma_full$metadata
)
plot_corr(se)
```

---

plot_heatmap *Generates a Heatmap of Cell Abundance Data from an SummarizedExperiment assay.*

Description

This function visualizes a data matrix from a specified slot (e.g., CLR-transformed, frequency, or pseudobulk data) after mean-centering. It supports optional filtering to only Highly Variable Cell Types (HVCs) and includes a sample annotation sidebar based on a specified metadata column.

Usage

```
plot_heatmap(
  se,
 assay = c("clr", "counts", "counts_imp", "freq", "freq_imp", "asin_sqrt", "clr_hvc",
    "pb"),
  label_col,
  cluster_rows = TRUE,
  cluster_cols = TRUE,
  scale = "none",
  clustering_method = "ward.D2",
  angle_col = "90",
  ...
)
```

Arguments

se	A SummarizedExperiment object.
assay	Character string (default: "clr"). The name of the assay in the SummarizedExperiment object to use for the heatmap. Must be one of: "clr" (CLR-transformed abundances, default), "clr_hvc" (CLR-transformed abundances of only the most highly variable cell types (HVCs)), "pb" (pseudobulk gene expression), "counts" (raw counts), "counts_imp" (imputed counts), "freq" (relative frequencies), "freq_imp" (imputed frequencies), or "asin_sqrt" (arcsin-square root transformed data).
label_col	Character string. The name of the column in colData(se) to use for annotating the samples (columns) of the heatmap.
cluster_rows	Logical (default: TRUE). Whether to apply hierarchical clustering to the cell types (rows).
cluster_cols	Logical (default: TRUE). Whether to apply hierarchical clustering to the samples (columns).
scale	Character string (default: "none"). Method for scaling the abundance values within the heatmap using pheatmap. Options include "none", "row", or "column". Note: The data is internally **mean-centered** (scale(center=TRUE, scale=FALSE)) across samples before being passed to pheatmap, regardless of this argument.
clustering_method	Character string (default: "ward.D2"). The clustering method to use for hierarchical clustering. Options are passed directly to hclust (e.g., "complete", "average", "ward.D2").
angle_col	Character string (default: "90"). Angle of the sample labels (columns).
...	Additional arguments passed directly to the pheatmap function.

Value

A pheatmap object, which is typically visualized automatically when called interactively.

See Also

SummarizedExperiment, pheatmap

Examples

```
# Example for a simple dataset:
data(example_data)
se <- ecoda(
    data = example_data$Zhang$cell_counts_lowresolution,
    metadata = example_data$Zhang$metadata,
)

plot_heatmap(se, label_col = c("Clinical.efficacy.", "Tissue"))

plot_heatmap(
    se,
    label_col = c("Clinical.efficacy.", "Tissue"),
    # Additional arguments for pheatmap:
    cutree_rows = 3,
    cutree_cols = 3
)

# Example of a large cohort with 868 samples and 69 cell types
se <- ecoda(
    data = example_data$GongSharma_full$cell_counts_highresolution,
    metadata = example_data$GongSharma_full$metadata
)

plot_heatmap(
    se,
    label_col = c("subject.cmv", "age_group"),
    cutree_rows = 3,
    cutree_cols = 5,
    show_colnames = FALSE
)

# Using only the most highly variable cell types (HVCs)
plot_heatmap(
    se,
    assay = "clr_hvc",
    label_col = c("subject.cmv", "age_group"),
    cutree_rows = 3,
    cutree_cols = 4,
    show_colnames = FALSE
)
```

---

plot_pca *Plot Principal Component Analysis and Calculate Clustering Scores*

Description

Performs Principal Component Analysis (PCA) on a selected data matrix from the SummarizedExperiment object (default: CLR-transformed abundances, clr) and visualizes the results. It can also calculate and display several metrics to evaluate the separation of groups defined by label_col. It uses factoextra.

Usage

```
plot_pca(
  se,
  assay = c("clr", "counts", "counts_imp", "freq", "freq_imp", "asin_sqrt", "clr_hvc",
    "pb"),
  label_col = NULL,
  scale. = FALSE,
  title = NULL,
  title_show_n_features = TRUE,
  legend_title = "Group",
  show_label_samples = FALSE,
  score_digits = 3,
  cluster_score = FALSE,
  mod_score = FALSE,
  sil_score = FALSE,
  anosim_score = TRUE,
  anosim_permutations = 99,
  anosim_parallel = 1,
  ari_nclusts = NULL,
  knn_k = 3,
  pointsize = 3,
  labelsize = 4,
  coord_equal = TRUE,
  axes = c(1, 2),
  invisible = c("var", "quali"),
  geom = "point",
  n_hv_feat_show = Inf,
  repel = TRUE
)
```

Arguments

se	A SummarizedExperiment object.
assay	Character string (default: "clr"). The name of the assay in the SummarizedExperiment object to use for PCA. Must be one of: "clr" (CLR-transformed abundances, default), "clr_hvc" (CLR-transformed abundances of only the most highly variable cell types (HVCs)), "pb" (pseudobulk gene expression), "counts" (raw counts), "counts_imp" (imputed counts), "freq" (relative frequencies), "freq_imp" (imputed frequencies), or "asin_sqrt" (arcsin-square root transformed data).
label_col	Character string (optional, default: NULL). The name of a column in colData(sd) used to color and group samples in the plot, and for calculating clustering scores.

scale. Logical (default: FALSE). A value indicating whether the variables should be scaled to have unit variance before the PCA.

title Character string (optional, default: NULL). The main title for the plot. If clustering scores are calculated, they are appended to this title.

title_show_n_features
Logical (optional, default: TRUE) Show the number of features (cell types or genes) used.

legend_title Character string (default: "Group"). The title for the color legend in the plot when a grouping column (label_col) is provided.

show_label_samples
Logical (default: FALSE). If TRUE, sample names are displayed next to the points in the plot. This automatically adds "text" to the geom parameter if it is not already present.

score_digits Integer (default: 3). The number of decimal places to round the clustering and ANOSIM scores appended to the plot title.

cluster_score Logical (default: TRUE). If TRUE, calculates the Adjusted Rand Index (ARI) using calc_ari.

mod_score Logical (default: TRUE). If TRUE, calculates the adjusted Modularity score using calc_modularity.

sil_score Logical (default: FALSE). If TRUE, calculates the average Silhouette width using calc_sil.

anosim_score Logical (default: TRUE). If TRUE, calculates the ANOSIM statistic (R) using vegan::anosim.

anosim_permutations
Integer (default: 99). The number of permutations to use when calculating the ANOSIM statistic.

anosim_parallel
Integer (default: 1). The number of parallel processes/cores to use for the ANOSIM calculation.

ari_nclusts Integer (optional, default: NULL). The target number of clusters (k) to use for hclust and pam. If NULL, it defaults to the number of unique levels in labels.

knn_k Integer (optional, default: 3). The number of nearest neighbors (k) to use for the Shared Nearest Neighbor (SNN) graph construction, required for Modularity score calculation. If NULL, it defaults to max(3, round(sqrt(N))), where N is the number of samples.

pointsize Numeric (default: 3). Size of the points in the plot.

labelsize Numeric (default: 4). Size of the variable labels in the plot.

coord_equal Logical (default: TRUE). If TRUE, forces the aspect ratio of the plot to be equal.

axes Numeric vector (default: c(1, 2)). The principal components to plot (e.g., c(1, 2) for PC1 vs PC2).

invisible Character vector (default: c("var", "quali")). Elements to hide. Can include "var" (variables/cell types), "ind" (samples), or "quali" (group centroids). see the description of factoextra for details.

geom	Character string or vector (default: "point"). The geometry to be used for the plot. Allowed values are combinations of: • "point" to show points for individuals (samples) • "text" to show labels for individuals (samples) • "arrow" to show vectors for variables (features) The default "point" plots points for both individuals and variables. Use c("point", "text") to show both points and labels for samples.
n_hv_feat_show	Integer (default: Inf). Number of most highly variable features to show based on their contribution to the selected axes.
repel	Logical (default: TRUE). Whether to use ggrepel to prevent label overlap for variable names.

Details

The clustering metrics (ARI, Modularity, Silhouette, ANOSIM) assess how well the sample groupings (labels) align with the underlying data structure in the feature space defined by the PCA.

Value

A ggplot object via factoextra visualizing the PCA results.

Examples

```
data(example_data)
se <- ecoda(
    data = example_data$Zhang$cell_counts_lowresolution,
    metadata = example_data$Zhang$metadata,
)

plot_pca(
    se,
    label_col = "Tissue",
    title = "PCA based on cell type composition",
    anosim_parallel = 1,
    n_hv_feat_show = 5 # Shows the most highly variable features (cell types)
)

# Using only the most highly variable cell types
plot_pca(
    se,
    assay = "clr_hvc",
    label_col = "Tissue",
    title = "PCA based on highly variable cell types",
    anosim_parallel = 1,
    n_hv_feat_show = nrow(S4Vectors::metadata(se)$clr_hvc)
)
```

plot_pca3d *Plot 3-dimensional interactive Principal Component Analysis plot*

Description

Performs Principal Component Analysis (PCA) on a selected data matrix from the ECODA object (default: CLR-transformed abundances, clr) and visualizes the results in 3D colored by label_col.

Usage

```
plot_pca3d(
  se,
  assay = c("clr", "counts", "counts_imp", "freq", "freq_imp", "asin_sqrt", "clr_hvc",
    "pb"),
  label_col = NULL,
  scale. = FALSE
)
```

Arguments

se	A SummarizedExperiment object.
assay	Character string (default: "clr"). The name of the assay in the SummarizedExperiment object to use for PCA. Must be one of: "clr" (CLR-transformed abundances, default), "clr_hvc" (CLR-transformed abundances of only the most highly variable cell types (HVCs)), "pb" (pseudobulk gene expression), "counts" (raw counts), "counts_imp" (imputed counts), "freq" (relative frequencies), "freq_imp" (imputed frequencies), or "asin_sqrt" (arcsin-square root transformed data).
label_col	Character string (optional, default: NULL). The name of a column in colData(se) used to color and group samples in the plot, and for calculating clustering scores.
scale.	Logical (default: FALSE). A value indicating whether the variables should be scaled to have unit variance before the PCA.

Value

An interactive 3D plotly object visualizing the PCA results.

Examples

```
data(example_data)
se <- ecoda(
    data = example_data$Zhang$cell_counts_lowresolution,
    metadata = example_data$Zhang$metadata,
)

plot_pca3d(se, label_col = "Tissue")
```

plot_varmean *Generates a Mean-Variance Plot for CLR-transformed Cell Type Data.*

Description

This function visualizes the relationship between the mean abundance (CLR) and the variance (CLR) for each cell type, which is typically used to identify Highly Variable Cell Types (HVCs).

Usage

```
plot_varmean(
  se,
  plot_title = "",
  highlight_hvcs = TRUE,
  labels = c("only_hvc", "all", "none"),
  plot_fit_line = FALSE,
  smooth_method = "lm"
)
```

Arguments

se	A SummarizedExperiment object containing pre-calculated cell type variances in `metadata(se)$celltype_variances` and the HVC list in `metadata(se)$hvcs`
plot_title	Character string (default: ""). The title for the plot.
highlight_hvcs	Logical (default: TRUE). If TRUE, the points corresponding to the Highly Variable Cell Types (HVCs) stored in the SummarizedExperiment object are colored red.
labels	Character (default: "only_hvc"). Options: "all" (label every point), "none" (no labels), or "only_hvc" (label only the highly variable cell types).
plot_fit_line	Logical (default: FALSE). If TRUE, a smoothing regression line is added to the plot.
smooth_method	Character string (default: "lm"). The smoothing method to use for the regression line if `plot_fit_line` is TRUE (e.g., "lm" for linear model, "loess" for local regression).

Details

If `highlight_hvcs` is TRUE, cell types previously identified and stored in `metadata(se)$hvcs` will be highlighted in red on the plot.

Value

A `ggplot` object representing the mean-variance plot.

See Also

SummarizedExperiment, `get_celltype_variances`

Examples

```
data(example_data)
se <- ecoda(
    data = example_data$GongSharma_full$cell_counts_highresolution,
    metadata = example_data$GongSharma_full$metadata
)

# 1. Generate the plot, highlighting HVCs (default)
plot_varmean(se, plot_title = "HVCs on Mean-Variance Plot")

# 2. Generate the plot without highlighting HVCs and add a fit line
plot_varmean(se,
    highlight_hvcs = FALSE,
    plot_fit_line = TRUE,
    smooth_method = "loess"
)
```

---

replace_zeros *Replace zero values in count or frequency data*

Description

This function replaces zero values in a data frame or matrix to facilitate downstream transformations that require strictly positive values, such as the Centered Log-Ratio (CLR) transformation.

Usage

```
replace_zeros(
  df,
  rep_method = c("counts", "frac_min"),
  pseudo_count = 0.5,
  pseudo_frac_min = 2/3,
  add_to = NULL
)
```

Arguments

df	A data frame or matrix where zeros need to be replaced.
rep_method	Character string specifying the replacement strategy. One of "counts", or "frac_min".
pseudo_count	Numeric (default: 0.5). Used if rep_method = "counts".
pseudo_frac_min	Numeric (default: 2/3). The multiplier for the minimum non-zero value when rep_method = "frac_min".
add_to	Character string. Should the value be added to "all" cells or just the "zeros"?

Details

The function provides two methods for handling zeros:

- **counts:** Adds a fixed value (`pseudo_count`).
- **frac_min:** Replaces zeros with a fraction (`pseudo_frac_min`) of the smallest observed non-zero value in the dataset.

Value

A data frame or matrix of the same dimensions as `df` with all zero values replaced by the calculated replacement value.

Examples

```
# Replace zeros in a count matrix
counts_df <- data.frame(A = c(10, 0, 5), B = c(20, 10, 0))
replace_zeros(counts_df, rep_method = "counts", add_to = "all")

# Replace zeros in a frequency matrix
freq_df <- data.frame(A = c(0.5, 0.5), B = c(0.2, 0.8), C = c(0.0, 1.0))
replace_zeros(freq_df, rep_method = "frac_min", add_to = "zeros")
```

Index