# Package 'SETA'

February 2, 2026

**Title** Single Cell Ecological Taxonomic Analysis

**Version** 1.1.0

**Description** Tools for compositional and other sample-level ecological analyses and visualizations tailored for single-cell RNA-seq data. SETA includes functions for taxonomizing celltypes, normalizing data, performing statistical tests, and visualizing results. Several tutorials are included to guide users and introduce them to key concepts. SETA is meant to teach users about statistical concepts underlying ecological analysis methods so they can apply them to their own single-cell data.

**URL** https://github.com/kkimler/SETA

**BugReports** https://github.com/kkimler/SETA/issues

**License** MIT + file LICENSE

**Encoding** UTF-8

**Depends** R (>= 4.5.0)

**Imports** dplyr, MASS, Matrix, SingleCellExperiment (>= 1.30.1), stats, tidygraph, rlang, utils

**Suggests** BiocStyle, caret, glmnet, corrplot, ggplot2, ggraph, knitr, methods, patchwork, reshape2, rmarkdown, SeuratObject, Seurat, SummarizedExperiment, TabulaMurisSenisData, tidyr, tidytext, testthat (>= 3.0.0)

**VignetteBuilder** knitr

**biocViews** SingleCell, Transcriptomics, RNASeq, GeneExpression, StatisticalMethod, DimensionReduction, Visualization, Normalization, DataRepresentation, SystemsBiology

**RoxygenNote** 7.3.2

**git_url** https://git.bioconductor.org/packages/SETA

**git_branch** devel

**git_last_commit** 4f2c997

**git_last_commit_date** 2025-10-29

**Repository** Bioconductor 3.23

**Date/Publication** 2026-02-01

**Author** Kyle Kimler [aut, cre] (ORCID: <https://orcid.org/0000-0003-4735-9064>),
Marc Elosua-Bayes [aut]

**Maintainer** Kyle Kimler <kkimler@broadinstitute.org>

# Contents



---

`SETA-package` *Single Cell Ecological Taxonomic Analysis*

---

## Description

SETA provides tools for compositional analysis of single-cell RNA-seq data, applying ecological principles and compositional data analysis (CoDA) methods to understand cell-type abundance patterns. The package offers a comprehensive workflow for extracting cell-type counts, applying various compositional transforms, performing latent space analysis, and visualizing results.

## Details

SETA treats cell-type abundance data as compositional data, similar to how ecologists analyze species abundance in environmental samples. This approach is appropriate because cell-type proportions sum to 1 (or 100 in one cell type affect all others.

The package implements several compositional transforms:

- **CLR (Centered Log-Ratio)**: Centers log-transformed data around the geometric mean
- **ALR (Additive Log-Ratio)**: Uses a reference cell type as denominator
- **ILR (Isometric Log-Ratio)**: Projects data onto orthonormal basis

- **Balance transforms**: User-defined log-ratios between groups of cell types

SETA also provides multi-resolution analysis capabilities, allowing users to analyze data at different taxonomic levels (e.g., broad cell types vs. specific subtypes).

## Value

This package provides functions that return various data structures:

- `setaCounts()`: Returns a sample-by-cell-type count matrix
- `setaTransform()`: Returns a list with transformed counts and method information
- `setaLatent()`: Returns a list with latent space coordinates, loadings, and variance explained
- `setaDistances()`: Returns a data frame with pairwise distances between samples
- `setaTaxonomyDF()`: Returns a data frame with hierarchical taxonomy information
- `taxonomy_to_tbl_graph()`: Returns a `tbl_graph` object for visualization

Key functions include:

- `setaCounts`: Extract cell-type count matrices from single-cell objects
- `setaTransform`: Apply compositional transforms (CLR, ALR, ILR, balance)
- `setaLatent`: Perform dimensionality reduction (PCA, PCoA, NMDS)
- `setaDistances`: Calculate compositional distances between samples
- `setaTaxonomyDF`: Create hierarchical taxonomies for multi-resolution analysis
- `taxonomy_to_tbl_graph`: Convert taxonomies to graph objects for visualization

For detailed examples, see the package vignettes:

- `vignette("introductory_vignette", package = "SETA")`
- `vignette("comparing_samples", package = "SETA")`
- `vignette("reference_frames", package = "SETA")`

The package is designed to be educational, teaching users about compositional data analysis principles while providing practical tools for single-cell research.

## Author(s)

Kyle Kimler <kkimler@broadinstitute.org> (aut, cre)

Marc Elosua-Bayes (aut)

## References

Aitchison, J. (1982). The Statistical Analysis of Compositional Data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 44(2), 139-177.

Greenacre, M. (2018). Compositional Data Analysis in Practice. CRC Press.

Pawlowsky-Glahn, V., Egozcue, J. J., & Tolosana-Delgado, R. (2015). *Modeling and Analysis of Compositional Data*. John Wiley & Sons.

## See Also

Useful links:

- https://github.com/CellDiscoveryNetwork/SETA
- Report bugs at https://github.com/CellDiscoveryNetwork/SETA/issues

---

`.extractMetadata` *Extract metadata from single-cell objects*

---

## Description

This function extracts metadata from various single-cell objects and converts them to data.frames for use with other SETA functions.

## Usage

```
.extractMetadata(obj)
```

## Arguments

| | |
|---|---|
| `obj` | A single-cell object (Seurat, SingleCellExperiment, or data.frame) |

## Value

A data.frame containing cell metadata

---

`data` *Synthetic single-cell, mixture and marker data*

---

## Description

`mockSeurat/mockSCE/mockLong` are designed to generate synthetic single-cell data. These data are not meant to represent biologically meaningful use-cases, but are solely intended for use in examples, for unit-testing, and to demonstrate `SETA`'s general functionality. Please don't use it in real life.

## Usage

```
mockSeurat(ng = 200, nc = 50, nt = 3, ns = 4, nb = 2)

mockLong(nc = 50, nt = 3, ns = 4, nb = 2, useBatch = TRUE)

mockCount(df = mockLong())

mockSCE(nc = 500, nt = 3, ns = 4, nb = 2, useBatch = TRUE)

makeTypeHierarchy(type_levels)
```

### Arguments

| | |
|---|---|
| ng, nc, nt, ns, nb | integer scalar specifying the number of genes, cells, types (groups), samples, and batches to simulate. |
| useBatch | logical scalar indicating whether to include a batch metadata column |
| df | data.frame in the format of 'mockLong()'. |
| type_levels | character vector of type levels representing cell types to be assigned to fine, mid and broad annotations |

### Value

- mockSeurat returns a Seurat object with rows = genes, columns = single cells, and cell metadata column type containing group identifiers.
- mockLong returns a data.frame object with rows = cells, columns = cell metadata column fine_type, mid_type, broad_type containing group identifiers at different hierarchical levels.
- mockCount returns a data.frame object with rows = type x sample, columns = metadata column bc containing the number of cells per type x sample.
- mockSCE returns a SingleCellExperiment object with rows = genes, columns = single cells, and cell metadata column type containing group identifiers.
- makeTypeHierarchy returns a list of as many elements as levels in the hierarchy, with names corresponding to the type levels and values containing the corresponding type identifiers at that level.

### Examples

```
seu <- mockSeurat()
sce <- mockSCE()
hierarchy <- makeTypeHierarchy(c("Lineage", "Type", "State"))
```

---

## resolveGroup

*'resolveGroup()' converts a user supplied *group specification* into the column indices of the corresponding leaves in a **counts** taxa matrix. A group specification can be:*

---

### Description

* **character vector of leaf names** present in 'colnames(counts)' * **character vector of higher level labels** that appear in a column of 'taxonomyDF' ('taxonomy_col') * **numeric vector of column indices**

### Usage

```
resolveGroup(spec, counts, taxonomyDF = NULL, taxonomy_col = NULL)
```

### Arguments

| | |
|---|---|
| spec | A character or numeric vector specifying a group. See *Details*. |
| counts | Numeric matrix: samples × taxa. Column names are treated as leaf (finest level) labels. |
| taxonomyDF | A data.frame returned by setaTaxonomyDF (optional). |
| taxonomy_col | Character. Which column of taxonomyDF to search when spec contains higher level labels (optional). |

### Details

If higher level labels are supplied, the function returns *all leaves* (finest level labels = 'row-names(taxonomyDF)') whose 'taxonomy_col' entry matches the requested label(s).

### Value

An integer vector of column indices inside 'counts'.

### Examples

```
## Mini counts matrix
mat <- matrix(1, 3, 4, dimnames = list(NULL,
             c("AT1", "AT2", "Fib1", "Fib2")))

## Taxonomy table
taxDF <- data.frame(
  broad_type = c("Epi","Epi","Stroma","Stroma"),
  row.names  = c("AT1","AT2","Fib1","Fib2")
)

## Resolve by leaf names
resolveGroup(c("AT1","AT2"), mat, taxDF, "broad_type")

## Resolve by higher level label
resolveGroup("Stroma", mat, taxDF, "broad_type")
```

---

## setaALR *Additive Log-Ratio (ALR) Transform*

---

### Description

Applies the ALR transform to an integer matrix of counts using a specified reference taxon. Samples are in rows and taxa in columns.

### Usage

```
setaALR(counts, ref, pseudocount = 1)
```

### Arguments

| | |
|---|---|
| counts | A numeric matrix with rows as samples and columns as taxa. |
| ref | Either the reference taxon name (a character string, which must appear in `colnames(counts)`) or the column index of the reference. |
| pseudocount | Numeric. Added to every count to avoid `log(0)`. Default is 1. |

### Details

For each sample, the transform computes $\mathrm{ALR}(x)_i = \log\big((x_i + c)/(x_{ref} + c)\big)$, where $c$ is the pseudocount, for all taxa $i$ except the reference.

### Value

A list with:

**method** A string indicating the ALR transform with the reference taxon.

**counts** A matrix with one row per sample and `(n_taxa - 1)` columns.

### Examples

```
# Example with 2 samples and 2 taxa:
mat <- matrix(c(1, 2, 4, 8), nrow = 2, byrow = TRUE)
colnames(mat) <- c("TaxonA", "TaxonB")
# Using TaxonA as the reference.
out <- setaALR(mat, ref = "TaxonA", pseudocount = 0)
out$counts
```

---

## setaBalance *User-defined balance transform (geometric-mean log-ratio)*

---

### Description

‘setaBalance()‘ computes *one or more* biologically meaningful balances (log-ratios) from a count matrix. Each balance is defined by two groups of taxa: **numerator** (‘num‘) and **denominator** (‘denom‘). Groups may be given as leaf names, higher-level labels (resolved through a ‘taxonomyDF‘), or column indices. The resulting balance will be positive if weighted in the numerator direction, and negative toward the denominator.

### Usage

```
setaBalance(
  counts,
  balances,
  taxonomyDF = NULL,
  taxonomy_col = NULL,
  normalize_to_parent = FALSE,
  pseudocount = 1
)
```

## Arguments

| | |
|---|---|
| counts | Numeric matrix with rows = samples and columns = taxa. |
| balances | A single balance (list with ‘num‘, ‘denom‘) **or** a named list of such lists for multiple balances. |
| taxonomyDF | Optional. A data frame from [setaTaxonomyDF()] used to expand higher-level labels into their descendant leaves. |
| taxonomy_col | Character. Column in ‘taxonomyDF‘ whose values should match any higher-level labels given in ‘balances‘. |
| normalize_to_parent | Logical (default ‘FALSE‘). If ‘TRUE‘, each sample is re-closed to the sub-composition formed by the union of num and denom before taking the log-ratio, i.e., the balance is within the parent total. |
| pseudocount | Numeric. Value added to every count to avoid ‘log(0)‘. Default ‘1‘. |

## Details

For every balance and every sample the function returns

$$\log\big(\mathrm{GM}(\mathrm{num})/\mathrm{GM}(\mathrm{denom})\big),$$

where $\mathrm{GM}(\cdot)$ is the geometric mean of the (pseudocount-adjusted) counts in the respective group.

## Value

A list with

**method** "balance".

**counts** Matrix with dimensions samples $\times$ balances. Column names are the balance names (or "Balance1" if unnamed).

## Examples

```
# Toy metadata & taxonomy table (from setaTaxonomyDF documentation)
meta <- data.frame(
  bc         = paste0("cell", 1:6),
  fine_type  = c("AT1","AT2","AT1","Fib1","Fib1","AT2"),
  mid_type   = c("Alv","Alv","Alv","Fib","Fib","Alv"),
  broad_type = c("Epi","Epi","Epi","Stroma","Stroma","Epi")
)
taxDF <- setaTaxonomyDF(meta,
  resolution_cols = c("broad_type","mid_type","fine_type"))

# Fake counts (2 samples x n_taxa leaves)
set.seed(687)
cnt <- matrix(rpois(2 * 3, 10), nrow = 2)
colnames(cnt) <- rownames(taxDF)

# (a) One balance: Epi vs Stroma (broad_type level)
bal1 <- list(num = "Epi", denom = "Stroma")
```

```
out1 <- setaBalance(cnt, bal1,
  taxonomyDF = taxDF, taxonomy_col = "broad_type")
out1$counts

# (b) Two balances in one call
bals <- list(
  epi_vs_stroma = list(num = "Epi", denom = "Stroma"),
  AT1_vs_AT2    = list(num = "AT1", denom = "AT2")
)
out2 <- setaBalance(cnt, bals,
  taxonomyDF = taxDF, taxonomy_col = "fine_type")
out2$counts
```

---

`setaCLR` *Centered Log-Ratio (CLR) Transform Applies a CLR transform to a matrix of counts. Samples should be in rows and taxa (cell types) in columns. For each sample, the transform computes* $\mathrm{CLR}(x)\_i = \log\big((x\_i + c)/g(x+c)\big)$*, where* $g(x+c)$ *is the geometric mean of the row.*

---

## Description

Centered Log-Ratio (CLR) Transform Applies a CLR transform to a matrix of counts. Samples should be in rows and taxa (cell types) in columns. For each sample, the transform computes $\mathrm{CLR}(x)_i = \log\big((x_i + c)/g(x+c)\big)$, where $g(x+c)$ is the geometric mean of the row.

## Usage

```
setaCLR(counts, pseudocount = 1)
```

## Arguments

| | |
|---|---|
| `counts` | An integer matrix of cell-type counts with samples in rows. |
| `pseudocount` | Numeric. Added to all entries to avoid `log(0)`. Default is 1. |

## Details

The CLR transform is defined sample-wise as

$$\mathrm{CLR}(x)_{ij} = \log\big((x_{ij} + c)/g_i\big),$$

where

$$g_i = \exp\big(\tfrac{1}{p}\sum_{j=1}^{p}\log(x_{ij} + c)\big)$$

for sample $i$, and $p$ is the number of taxa. Here $c$ is the pseudocount.

## Value

A list with:

**method** A string indicating the transform (`"CLR"`).

**counts** A matrix of the same dimensions as the input after the CLR transform.

## References

Aitchison, J. (1982). The Statistical Analysis of Compositional Data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 44(2), 139–177.

## Examples

```
mat <- matrix(c(1, 2, 4, 8), nrow = 2, byrow = TRUE)
colnames(mat) <- c("Taxon1", "Taxon2")
out <- setaCLR(mat, pseudocount = 0)
out$counts
```

---

`setaCounts` *Extract Taxonomic Counts from Various Single Cell Objects*

---

## Description

Given a long-form `data.frame`, creates a type-by-sample matrix of cell counts. Users can specify the column names for cell types, samples, and barcodes.

## Usage

```
setaCounts(obj, cell_type_col = "type", sample_col = "sample", bc_col = "bc")
```

## Arguments

| | |
|---|---|
| `obj` | A long-form data.frame, Seurat object, or SingleCellExperiment object. |
| `cell_type_col` | Column name for cell types (default "type") |
| `sample_col` | Column name for sample IDs (default "sample") |
| `bc_col` | Column name for barcodes (default "bc") Use ‘"rownames"‘ to extract barcodes from row names. |

## Value

A sample-by-celltype matrix of counts.

## Examples

```
# For a data.frame with custom column names:
set.seed(687)
df <- data.frame(
  barcode = paste0("cell", 1:10),
  cellType = sample(c("Tcell", "Bcell"), 10, TRUE),
  sampleID = sample(c("sample1","sample2"), 10, TRUE)
)
cmat <- setaCounts(df,
                   cell_type_col = "cellType",
                   sample_col = "sampleID",
                   bc_col = "barcode")
print(head(cmat))
```

---

`setaDistances` *Compute Distance Matrix between Samples*

---

## Description

Calculates a pairwise distance matrix between samples based on user-specified or default (`"euclidean"`) distance metrics. If used on CLR-transformed data, the default Euclidean distance is the *Aitchison distance*, which is commonly used in compositional data analysis (CoDA).

## Usage

```
setaDistances(transformed_counts, method = "euclidean")
```

## Arguments

| | |
|---|---|
| `transformed_counts` | Numeric matrix: rows as samples and columns as taxa (e.g., output from `setaCLR`, `setaTransform`, etc.). |
| `method` | Character. Distance metric for `dist`. Default: `"euclidean"` See `dist` for options. |

## Details

This function calculates distances **between samples**.

Output is a long-form structure convenient to merge with sample-level metadata using `merge` or `left_join`.

## Value

A long-form `data.frame` with three columns:

**from** Sample ID of the first sample in the pairwise comparison.

**to** Sample ID of the second sample in the pairwise comparison.

**distance** Numeric distance between the two samples.

## References

Aitchison, J. (1982). The Statistical Analysis of Compositional Data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 44(2), 139-177.

## Examples

```
# Example CLR transformed data (2 samples, 3 taxa)
mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, byrow = TRUE)
colnames(mat) <- c("Taxon1", "Taxon2", "Taxon3")
rownames(mat) <- c("SampleA", "SampleB")

clr_mat <- setaCLR(mat, pseudocount = 0)

# Calculate Euclidean (Aitchison) distance
dist_df <- setaDistances(clr_mat)
print(head(dist_df))
```

---

`setaILR` *Isometric Log-Ratio (ILR) Transform Applies the ILR transform to an integer counts matrix. For each sample (row), the data are log-transformed (with an optional Box Cox like transformation) then projected onto an orthonormal Helmert basis, reducing dimensionality by one.*

---

## Description

Isometric Log-Ratio (ILR) Transform Applies the ILR transform to an integer counts matrix. For each sample (row), the data are log-transformed (with an optional Box Cox like transformation) then projected onto an orthonormal Helmert basis, reducing dimensionality by one.

## Usage

```
setaILR(counts, boxcox_p = 0, taxTree = NULL, pseudocount = 1)
```

## Arguments

| | |
|---|---|
| `counts` | An integer matrix of celltype counts with samples in rows. |
| `boxcox_p` | Numeric. If nonzero, a Box Cox type transform is applied to the log-values. Default is 0 (no Box Cox transformation). |
| `taxTree` | Unused. Reserved for future taxonomic-balance approaches. |
| `pseudocount` | Numeric. Added to avoid `log(0)`. Default is 1. |

## Details

The ILR transform is computed as follows:

1. Add a pseudocount and take the natural logarithm:

$$y = \log(x + \text{pseudocount})$$

2. If `boxcox_p != 0`, apply the Box Cox like transform:

$$y = \frac{\exp(p\,y) - 1}{p}$$

3. Project the log-transformed data onto an orthonormal Helmert basis computed via QR decomposition.

## Value

A list with:

**method** A string indicating the ILR transform. If `boxcox_p` is nonzero, the value is indicated in the method string.

**counts** A matrix of ILR-transformed values with `ncol(counts) - 1` columns and the same number of rows (samples) as the input.

## References

Aitchison, J. (1982). The Statistical Analysis of Compositional Data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 44(2), 139-177.

## Examples

```
# Example matrix: rows are samples, columns are cell types.
mat <- matrix(c(1, 2, 4, 8), nrow = 2, byrow = TRUE)
colnames(mat) <- c("A", "B")
# ILR transformation reduces the dimension by 1.
out <- setaILR(mat, boxcox_p = 0, pseudocount = 1)
out$counts
```

---

`setaLatent` *Compute a Latent Space from Transformed Counts*

---

## Description

Given an object produced by one of the `seta*` transform functions (e.g., `setaCLR`), this function applies a dimension reduction method (PCA, PCoA, or NMDS) to `transform_obj$counts`.

## Usage

```
setaLatent(transform_obj, method = c("PCA", "PCoA", "NMDS"), dims = 2)
```

### Arguments

| | |
|---|---|
| transform_obj | A list returned by, e.g., setaCLR, setaILR containing a counts matrix, where rows are samples and columns are features (taxa or cell types). |
| method | A string specifying the dimension reduction method. One of "PCA", "PCoA", or "NMDS". |
| dims | Integer number of dimensions to return. Default is 2. |

### Details

- **PCA**: Uses stats::prcomp on the rows of transform_obj$counts.
- **PCoA**: Computes a distance matrix via stats::dist, then applies classical multidimensional scaling (stats::cmdscale).
- **NMDS**: Uses MASS::isoMDS to compute non-metric MDS from the distance matrix.

Each method returns a data frame of coordinates in latentSpace, plus additional information specific to that method.

### Value

A list containing:

**method** The chosen latent space method.

**latentSpace** A data frame of coordinates in the chosen latent space with dims columns.

**loadings** For PCA, the loadings matrix. Otherwise NA.

**varExplained** Variance explained (for PCA or PCoA) or stress (for NMDS).

### Examples

```
set.seed(687)
mat <- matrix(rpois(20, lambda=5), nrow=4)  # small 4x5 matrix
colnames(mat) <- paste0("C", 1:5)
clr_out <- setaCLR(mat)
latent_pca <- setaLatent(clr_out, method="PCA", dims=2)
latent_pca$latentSpace
```

---

## setaLogCPM *log2(CPM) Transform Computes the log2 counts-per-million (CPM) for each sample. Samples are in rows and taxa in columns.*

---

### Description

log2(CPM) Transform Computes the log2 counts-per-million (CPM) for each sample. Samples are in rows and taxa in columns.

### Usage

```
setaLogCPM(counts, pseudocount = 1, size_factors = NULL, scale_factor = 1e+06)
```

### Arguments

| | |
|---|---|
| counts | A numeric matrix with rows as samples and columns as taxa. |
| pseudocount | Numeric. Added to counts to avoid log2(0). Default is 1. |
| size_factors | Optional numeric vector of library sizes for each sample. If NULL, the row sums are used. |
| scale_factor | Numeric. The scaling factor, typically 1e6 for CPM. Default is 1e6. |

### Details

The transform is

$$\log_2\big(((x+c)/L)\times s\big),$$

where $c$ is the pseudocount, $L$ is the per-sample library size, and $s$ is scale_factor.

### Value

A list with:

**method** The string "logCPM".

**counts** A matrix of the same dimensions with log2-transformed CPM values.

### Examples

```
mat <- matrix(c(10, 20, 100, 200), nrow = 2, byrow = TRUE)
out <- setaLogCPM(mat, pseudocount = 1)
out$counts
```

---

## setaMetadata *Extract Sample-Level Metadata from Various Objects*

---

### Description

This function extracts sample-level metadata from a dataframe. It ensures that each metadata column contains unique values per sample. If a metadata column contains non-unique values within any sample, that column is excluded from the output, and the user is notified via a warning. Useful when preparing metadata for visualizations or analyses where sample-level inspection is required.

### Usage

```
setaMetadata(x, sample_col = "Sample ID", meta_cols = NULL)
```

### Arguments

| | |
|---|---|
| x | An object of class dataframe which contains cell-level or sample-level metadata. |
| sample_col | Character. The sample identifier column name in source_obj. Default is 'sample_id'. This column is used to group the metadata. |
| meta_cols | Character vector. Names of metadata columns to retain. If NULL, all columns present in the source object are considered. However, only those columns where all entries are identical within each sample are included in the final output. |

### Value

A dataframe where each row corresponds to a unique sample and each column represents a metadata variable that has uniform values within samples. Columns with non-unique values within any sample are excluded, and a warning lists these columns.

### Examples

```
# Using a Seurat object

# if (requireNamespace("SeuratObject", quietly = TRUE)) {
# meta_df <- setaMetadata(seurat_obj[[]],
#                         sample_col="donor_id",
#                         meta_cols=c("disease", "Severity"))
# }
#
# Using a SingleCellExperiment object with default parameters
# if (requireNamespace("SingleCellExperiment", quietly = TRUE)) {
# meta_df <- setaMetadata(data.frame(colData(sce_obj)))
#
# Using a dataframe and extracting all possible metadata columns
# meta_df <- setaMetadata(df)
# }
```

---

## setaPercent *Percentage Transform Converts each row (sample) of a counts matrix to percentages of its row sum.*

---

### Description

Percentage Transform Converts each row (sample) of a counts matrix to percentages of its row sum.

### Usage

```
setaPercent(counts)
```

### Arguments

| | |
|---|---|
| counts | A numeric matrix with rows as samples and columns as taxa. |

## Details

Useful for simplified comparisons and as an input to non-parametric tests.

## Value

A list with:

**method** The string `"percent"`.

**counts** A matrix of the same dimensions as `counts`, where each row sums to 100.

## Examples

```
mat <- matrix(c(1,2,4,8), nrow = 2, byrow = TRUE)
out <- setaPercent(mat)
out$counts
```

---

`setaTaxonomyDF` *Build a taxonomy data frame at multiple resolutions*

---

## Description

setaTaxonomyDF() converts **one long-form metadata data.frame**-typically colData(sce), seu[[]], or any frame you already have, into a tidy taxonomy table. Each row corresponds to a unique value of the *finest* label (the **last** element of ‘resolution_cols‘), and every coarser label sits in its own column.

## Usage

```
setaTaxonomyDF(
  obj,
  resolution_cols = c("broad_type", "mid_type", "fine_type"),
  bc_col = "bc"
)
```

## Arguments

| | |
|---|---|
| `obj` | A data.frame or similar object containing cell metadata. |
| `resolution_cols` | A character vector of column names indicating hierarchical taxonomy (from broad to fine). |
| `bc_col` | Optional. The name of the column containing barcodes, or "rownames" if they are row names. |

## Details

## What the input must contain * exactly **one row per cell** * at least one **barcode** column (default ‘"bc"‘). Pass ‘bc_col = "rownames"‘ if barcodes live in ‘rownames(obj)‘. * **all** columns listed in ‘resolution_cols‘

No ‘Seurat‘/‘SingleCellExperiment‘ objects are accepted here: extract their metadata/colData first, then hand it in as a ‘data.frame‘

## Value A ‘data.frame‘ whose **rownames** are the finest label. If any finest label maps to more than one set of coarser labels the function should stop with an informative error.

## Value

A ‘data.frame‘ with one row per unique value of the finest label (the last entry in ‘resolution_cols‘), and one column for each resolution level. The row names are set to the finest label values. If any finest label maps to more than one combination of coarser labels, the function stops with an informative error.

## Examples

```
meta <- data.frame(
  bc          = paste0("cell", 1:6),
  broad_type   = c("Epi","Epi","Epi","Stroma","Stroma","Epi"),
  mid_type    = c("Alv","Alv","Alv","Fib","Fib","Alv"),
  fine_type  = c("AT1","AT2","AT1","Fib1","Fib1","AT2")
)
setaTaxonomyDF(meta,
               resolution_cols = c("broad_type","mid_type","fine_type"))

## barcodes can be in rownames with bc_col = "rownames" (as in Seurat Object)
rownames(meta) <- meta$bc
meta$bc <- NULL
setaTaxonomyDF(meta,
               resolution_cols = c("broad_type","mid_type","fine_type"),
               bc_col = "rownames")
```

---

`setaTransform` *Wrapper for Compositional Transforms with Optional Within-Lineage Resolutions A convenience function that dispatches to one of the transforms: CLR, ALR, ILR, percent, or logCPM. Note that the input* `counts` *matrix should have rows as samples and columns as taxa. Optionally, you can supply a taxonomy data frame to perform a within-lineage transform at a specified resolution.*

---

## Description

Wrapper for Compositional Transforms with Optional Within-Lineage Resolutions A convenience function that dispatches to one of the transforms: CLR, ALR, ILR, percent, or logCPM. Note that the input `counts` matrix should have rows as samples and columns as taxa. Optionally, you can supply a taxonomy data frame to perform a within-lineage transform at a specified resolution.

## Usage

```
setaTransform(
  counts,
  method = c("CLR", "ALR", "ILR", "percent", "logCPM", "balance"),
  ref = NULL,
  taxTree = NULL,
  pseudocount = 1,
  size_factors = NULL,
  taxonomyDF = NULL,
  taxonomy_col = NULL,
  within_resolution = FALSE,
  balances = NULL,
  normalize_to_parent = FALSE
)
```

## Arguments

| | |
|---|---|
| counts | A numeric matrix with rows as samples and columns as taxa. |
| method | A character string specifying which transform to apply. One of "CLR", "ALR", "ILR", "percent", "logCPM" or "balance". |
| ref | Reference taxon (only used if method = "ALR"). This can be a taxon name or a column index. |
| taxTree | Optional tree for ILR (not yet implemented). |
| pseudocount | Numeric, used by CLR, ALR, ILR, and logCPM. Default is 1. |
| size_factors | For logCPM scaling. If NULL, uses row sums. |
| taxonomyDF | Optional data frame specifying higher-level groupings for each taxon. Row names of taxonomyDF should match colnames(counts). |
| taxonomy_col | The column of taxonomyDF indicating which lineage each taxon belongs to. Only used if within_resolution = TRUE. |
| within_resolution | Logical. If TRUE, applies the transform within each lineage of taxa defined by taxonomyDF[[taxonomy_col]] separately, then merges them back into the original matrix structure. Default is FALSE. Ignored for ‘"balance"‘. |
| balances | For ‘"balance"‘: a single balance list or a named list; |
| normalize_to_parent | Logical, passed to [setaBalance()]. |

## Value

A list with the following elements:

**transform_method** The core transform, e.g. \"CLR\", \"ALR\", etc.

**within_resolution** Logical indicating if a within-lineage transform was used.

**grouping_var** The name of the column in taxonomyDF used for grouping (lineages) if within_resolution = TRUE, otherwise NULL.

**counts** The resulting matrix after transformation, with the same dimensions as the input counts.

## Examples

```
mat <- matrix(c(1, 2, 4, 8, 3, 6, 9, 12),
              nrow = 2, byrow = TRUE)
colnames(mat) <- c("TaxonA1", "TaxonA2", "TaxonB1", "TaxonB2")

# Build a taxonomy data frame labeling lineages
df_lineage <- data.frame(
    Lineage = c("LineageA", "LineageA", "LineageB", "LineageB"),
    row.names = colnames(mat)
)

# Apply CLR transform to all columns together
out1 <- setaTransform(mat, method = "CLR")

# Apply CLR within each Lineage
out2 <- setaTransform(
    mat,
    method = "CLR",
    taxonomyDF = df_lineage,
    taxonomy_col = "Lineage",
    within_resolution = TRUE
)
```

---

# taxonomy_to_tbl_graph *Convert Multi-Column Taxonomy to a Single-Root tbl_graph (with node metadata)*

---

## Description

This function takes a data frame describing a hierarchical taxonomy across multiple columns (e.g., broad -> mid -> fine). Each row represents a unique path through the hierarchy. The function introduces a single root node (named `root_name`) above the first hierarchy column, then constructs a directed tree in which each level connects to the next. After building the graph, it appends node-level metadata by looking up which rows (and columns) in `tax_df` contain each node. This allows you to color or facet by different levels of the taxonomy when using **ggraph**.

## Usage

```
taxonomy_to_tbl_graph(tax_df, columns = NULL, root_name = "AllCells")
```

## Arguments

| | |
|---|---|
| `tax_df` | A data frame with one row per unique path in the hierarchy. For example, if your columns are `c(\"broad\",\"mid\",\"fine\")`, each row is a single path from `broad -> mid -> fine`. |
| `columns` | A character vector of column names in `tax_df` to use. They should be ordered from the broadest level (first) to the finest level (last). If `NULL`, the function will use all columns of `tax_df` in their given order. |

root_name A character string naming the artificial root node, inserted above the first hierarchy column. Default is \"AllCells\".

## Details

1. The function first builds an edge list

    1. Root -> level1 for each row
    2. level1 -> level2
    3. ...
    4. level_{N-1} -> levelN

    and removes duplicates, creating a single connected tree.

    2. It then *annotates each node* with the best-known taxonomy data. For a node named x, we look up all rows of tax_df where x appears in columns, gather the distinct values from each col, and store them joined with \"|\" if more than one distinct value is found.

    This means if a node is shared among multiple broad categories (uncommon, but possible), that node's broad column will contain something like \"Epithelial|Stromal\".

## Value

A tbl_graph object (directed) with a single root node. The node data includes extra columns corresponding to each level in columns. If a node corresponds to multiple categories at a given level, these are combined with \"|\".

## Examples

```
# Minimal example with a 3-level hierarchy (broad -> mid -> fine)
tax_df_example <- data.frame(
    broad = c("Epithelial", "Epithelial", "Stromal"),
    mid   = c("Alveolar", "Alveolar", "Fibroblast"),
    fine  = c("AlveolarType1", "AlveolarType2", "Fibroblast1"),
    stringsAsFactors = FALSE
)
library(tidygraph)
library(ggraph)
library(ggplot2)

# Build a single-root tree and incorporate node metadata
tbl_g <- taxonomy_to_tbl_graph(
    tax_df_example,
    columns   = c("broad", "mid", "fine"),
    root_name = "AllCells"
)

# Inspect node data (metadata for each node)
as.data.frame(tbl_g, "nodes")

# Visualize with ggraph, coloring by 'broad' level
ggraph(tbl_g, layout = "tree") +
```

```
geom_edge_diagonal() +
geom_node_point(aes(color = broad), size = 3) +
geom_node_text(aes(label = name), vjust = 1, hjust = 0.5) +
theme_minimal() +
labs(title = "Single-Root Taxonomy Tree")
```

# Index