Package 'SpNeigh'

April 16, 2026

Title Spatial Neighborhood Modeling and Differential Expression Analysis for Transcriptomics

Version 0.99.43

Description SpNeigh provides methods for neighborhood-aware analysis of spatial transcriptomics data. It supports boundary detection, spatial weighting (centroid- and boundary-based), spatially informed differential expression using spline-based models, and spatial enrichment analysis via the Spatial Enrichment Index (SEI). Designed for compatibility with Seurat objects, SpatialExperiment objects and spatial data frames, SpNeigh enables interpretable, publication-ready analysis of spatial gene expression patterns.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports concaveman, dbscan, dplyr, FNN, ggplot2, limma, magrittr, Matrix, methods, patchwork, purrr, rlang, scales, Seurat, sf, SingleCellExperiment, SpatialExperiment, splines, stringr, SummarizedExperiment, tibble, tidyr

Suggests BiocStyle, knitr, rmarkdown, SeuratObject, testthat (>= 3.0.0)

biocViews Spatial, SingleCell, GeneExpression, DifferentialExpression, Transcriptomics, Software

URL https://github.com/jinming-cheng/SpNeigh

BugReports https://github.com/jinming-cheng/SpNeigh/issues

Depends R (>= 4.4.0)

VignetteBuilder knitr

Config/testthat/edition 3

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

git_branch devel

git_last_commit 510beba

git_last_commit_date 2026-04-09

Repository Bioconductor 3.23

Date/Publication 2026-04-15

Author Jinming Cheng [aut, cre] (ORCID:
<https://orcid.org/0000-0003-3806-4694>)

Maintainer Jinming Cheng <jinming.cheng@outlook.com>

Contents

SpNeigh-package *SpNeigh: Spatial Neighborhood Modeling and Differential Expression Analysis for Transcriptomics*

Description

SpNeigh provides methods for neighborhood-aware analysis of spatial transcriptomics data. It supports boundary detection, spatial weighting (centroid- and boundary-based), spatially informed differential expression using spline-based models, and spatial enrichment analysis via the Spatial Enrichment Index (SEI). Designed for compatibility with Seurat objects, SpatialExperiment objects and spatial data frames, SpNeigh enables interpretable, publication-ready analysis of spatial gene expression patterns.

Author(s)

Maintainer: Jinming Cheng <jinming.cheng@outlook.com> (ORCID)

See Also

Useful links:

- https://github.com/jinming-cheng/SpNeigh
- Report bugs at https://github.com/jinming-cheng/SpNeigh/issues

---

addBoundary *Add boundary outlines to a spatial ggplot*

Description

Overlays spatial boundaries (from points or polygons) onto a ggplot2 object. The input can be either a data frame of boundary points or an sf object of POLYGON geometries. This function is typically used as an additive layer (+ addBoundary(...)) in conjunction with a base plot created using plotBoundary().

Usage

```
addBoundary(
  boundary = NULL,
  color_boundary = "black",
  linewidth_boundary = 1.5
)
```

Arguments

boundary	A data frame with columns x, y, and region_id or an sf object of POLYGON or LINESTRING geometries.
color_boundary	Color for boundary lines. Default is "black".
linewidth_boundary	Numeric. Line width for boundary outlines. Default is 1.5.

Value

A ggplot2::geom_path layer that can be added to an existing plot.

Examples

```
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

# Automatically get boundary for a cluster
boundary_points <- getBoundary(
    data = coords, one_cluster = 2,
    subregion_method = "dbscan",
    eps = 120, minPts = 10
)

# Plot with boundary overlay
plotBoundary(coords) + addBoundary(boundary_points)
```

---

addBoundaryPoly *Add boundary polygons or linestrings to a spatial plot*

Description

Overlays boundary polygons or linestrings on a spatial ggplot2 plot. This function adds an sf geometry layer to display complete boundary shapes for visualizing spatial clusters, rings, or enriched zones.

Usage

```
addBoundaryPoly(
  boundary_poly,
  color_boundary = "black",
  linewidth_boundary = 1.5
)
```

Arguments

boundary_poly	An sf object containing POLYGON or LINESTRING geometries and a region_id column.
color_boundary	Color for boundary lines. Default is "black".
linewidth_boundary	Numeric. Line width for boundary outlines. Default is 1.5.

Value

A ggplot2::geom_sf layer that can be added to an existing plot.

Examples

```
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

boundary_points <- getBoundary(
    data = coords, one_cluster = 2,
    subregion_method = "dbscan",
    eps = 120, minPts = 10
)
boundary_polys <- buildBoundaryPoly(boundary_points)

# Add boundary polygons to the plot
plotBoundary(coords) +
    addBoundaryPoly(boundary_polys, color_boundary = "blue")
```

---

boundaryPolyToPoints *Convert boundary polygons to boundary point coordinates*

Description

Extracts the vertex coordinates (x, y) of boundary geometries from an sf object containing POLYGON or LINESTRING features. This is useful for recovering the original boundary points from smoothed or labeled polygonal regions.

Usage

```
boundaryPolyToPoints(boundary_poly = NULL)
```

Arguments

boundary_poly	An sf object containing only POLYGON or LINESTRING geometries. Must include a region_id column for labeling subregions.

Value

A data frame with columns `x`, `y`, and `region_id`, containing the vertex coordinates of the boundary geometries.

Examples

```
# Load coordinates and generate boundary
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))
boundary_points <- getBoundary(data = coords, one_cluster = 2)
boundary_polys <- buildBoundaryPoly(boundary_points)

# Convert back to boundary points
boundary_pts <- boundaryPolyToPoints(boundary_polys)
head(boundary_pts)
```

---

buildBoundaryPoly *Convert boundary points into valid polygon geometries*

Description

Constructs spatial polygon objects from boundary points generated by the `getBoundary()` function. Each set of boundary points (grouped by `region_id`) is converted into a closed polygon. This function ensures that each polygon is valid and ready for downstream spatial analysis.

Usage

```
buildBoundaryPoly(boundary = NULL)
```

Arguments

`boundary`	A data frame of boundary points, typically returned by `getBoundary()`. Must contain columns `x`, `y`, and `region_id`.

Value

An `sf` object containing one or more valid `POLYGON` geometries with a `region_id` column.

Examples

```
# Load coordinates
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

# Get boundary points for cluster 2
boundary_points <- getBoundary(data = coords, one_cluster = 2)
```

```
# Convert to polygon
boundary_polys <- buildBoundaryPoly(boundary_points)

# Plot the resulting polygons
plot(boundary_polys)
```

`colors15_cheng` *Custom Color Palette*

Description

A character vector containing 15 colors.

Usage

```
colors15_cheng
```

Format

A character vector of length 15.

Examples

```
plot(1:15, 1:15, pch = 16, col = colors15_cheng)
barplot(rep(1, 15), col = colors15_cheng)
```

`computeBoundaryWeights`
Compute spatial weights based on distance to nearest boundary

---

Description

Computes spatial weights for a subset of cells based on their Euclidean distance to the nearest boundary segment. This method supports both closed boundary polygons and open boundary edges (e.g., LINESTRINGs), and is useful for modeling proximity-based enrichment near anatomical or user-defined regions.

Usage

```
computeBoundaryWeights(
  data = NULL,
  cell_ids = NULL,
  boundary = NULL,
  scale = TRUE,
  method = c("inverse", "gaussian", "linear", "quadratic"),
  sigma = 0.5
)
```

Arguments

data	A data frame, Seurat object or SpatialExperiment object containing spatial coordinates. Must include columns: cell, x, and y.
cell_ids	A character vector of cell IDs to use for weight computation.
boundary	Either an sf object containing boundary geometries (POLYGON or LINESTRING), or a data frame of boundary points returned from getBoundary().
scale	Logical. Whether to scale distances to the range [0, 1] before applying decay functions. Default is TRUE.
method	Decay function to convert distances to weights. One of: "inverse", "gaussian", "linear", or "quadratic".
sigma	Standard deviation for the Gaussian decay (used only if method = "gaussian"). Default is 0.5 (recommended for scaled distances).

Details

Supports multiple decay methods for converting distance into weights. Optionally scales distances to the [0, 1] range before computing weights.

Value

A named numeric vector of weights, with names corresponding to cell_ids.

Examples

```
# Load spatial coordinates
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

# Get and build boundary polygons from cluster 2
boundary_points <- getBoundary(data = coords, one_cluster = 2)
boundary_polys <- buildBoundaryPoly(boundary_points)

# Compute weights to polygon boundary
cells_c2 <- subset(coords, cluster == 2)$cell
weights <- computeBoundaryWeights(
    data = coords, cell_ids = cells_c2,
```

```
    boundary = boundary_polys[1, ]
)

# Compute weights to a specific boundary edge
boundary_edges <- splitBoundaryPolyByAnchor(boundary_polys[1, ])
weights_edge <- computeBoundaryWeights(
    data = coords, cell_ids = cells_c2,
    boundary = boundary_edges[2, ]
)
```

`computeCentroidWeights` *Compute spatial weights based on distance to the centroid*

Description

Computes spatial weights for a set of cells based on their Euclidean distance to the centroid of the selected group of cells. This is useful for modeling gradient-like spatial expression centered around a cluster or region. Supports multiple decay methods and optional distance scaling.

Usage

```
computeCentroidWeights(
  data = NULL,
  cell_ids = NULL,
  scale = TRUE,
  method = c("inverse", "gaussian", "linear", "quadratic"),
  sigma = 0.5
)
```

Arguments

`data`	A data frame, Seurat object or SpatialExperiment object containing spatial coordinates. Must include columns: `cell`, `x`, and `y`.
`cell_ids`	A character vector of cell IDs to use for weight computation.
`scale`	Logical. Whether to scale distances to the range [0, 1] before applying decay functions. Default is `TRUE`.
`method`	Decay function to convert distances to weights. One of: "inverse", "gaussian", "linear", or "quadratic".
`sigma`	Standard deviation for the Gaussian decay (used only if `method = "gaussian"`). Default is `0.5` (recommended for scaled distances).

Value

A named numeric vector of weights (length = number of cells in `cell_ids`). Higher weights correspond to cells closer to the centroid.

Examples

```
# Load spatial coordinates
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

# Select cells from cluster 2
cells_c2 <- subset(coords, cluster == 2)$cell

# Compute centroid-based weights using default settings
weights <- computeCentroidWeights(data = coords, cell_ids = cells_c2)
head(weights)
```

`computeSEI` *Compute Spatial Enrichment Index (SEI)*

Description

Alias for `computeSpatialEnrichmentIndex`.

Usage

```
computeSEI(exp_mat = NULL, weights = NULL)
```

Arguments

`exp_mat`	A normalized gene expression matrix with genes as rows and cells as columns. Should be of class `matrix` or `dgCMatrix`.
`weights`	A numeric vector of spatial weights (e.g., from `computeBoundaryWeights` or `computeCentroidWeights`). Must be the same length as the number of columns (cells) in `exp_mat`.

Value

A data frame containing the spatial enrichment index (SEI) results.

Examples

```
# Load spatial coordinates and log-normalized expression
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))
logNorm_expr <- readRDS(system.file("extdata", "LogNormExpr.rds",
    package = "SpNeigh"
))

# Compute spatial weights and SEI
bon_c0 <- getBoundary(data = coords, one_cluster = 0)
```

```
cells_c0 <- subset(coords, cluster == 0)$cell
weights <- computeBoundaryWeights(
    data = coords,
    cell_ids = cells_c0,
    boundary = bon_c0
)
sei_df <- computeSEI(logNorm_expr[, cells_c0], weights)
head(sei_df)
```

`computeSpatialEnrichmentIndex` *Compute Spatial Enrichment Index (SEI) for Each Gene*

Description

Calculates the spatial enrichment index (SEI) for each gene based on a user-supplied set of spatial weights. The SEI reflects the extent to which gene expression is enriched in spatially weighted regions of the tissue (e.g., near boundaries or centroids).

Usage

```
computeSpatialEnrichmentIndex(exp_mat = NULL, weights = NULL)
```

Arguments

`exp_mat`	A normalized gene expression matrix with genes as rows and cells as columns. Should be of class `matrix` or `dgCMatrix`.
`weights`	A numeric vector of spatial weights (e.g., from `computeBoundaryWeights` or `computeCentroidWeights`). Must be the same length as the number of columns (cells) in `exp_mat`.

Details

This method supports both dense (`matrix`) and sparse (`dgCMatrix`) gene expression formats, and can be applied using any distance-based weighting scheme.

Value

A data frame with the following columns:

`gene` Gene name

`SEI` Spatial enrichment index: weighted mean expression across cells

`mean_expr` Mean expression across all cells (unweighted)

`normalized_SEI` Ratio of SEI to mean expression; used to compare genes independent of baseline expression

The result is sorted in descending order by `normalized_SEI`.

Examples

```
# Load spatial coordinates and log-normalized expression
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))
logNorm_expr <- readRDS(system.file("extdata", "LogNormExpr.rds",
    package = "SpNeigh"
))

# Compute spatial weights and SEI
bon_c0 <- getBoundary(data = coords, one_cluster = 0)
cells_c0 <- subset(coords, cluster == 0)$cell
weights <- computeBoundaryWeights(
    data = coords,
    cell_ids = cells_c0,
    boundary = bon_c0
)
sei_df <- computeSpatialEnrichmentIndex(logNorm_expr[, cells_c0], weights)
head(sei_df)
```

---

computeSpatialInteractionMatrix

Compute a spatial neighborhood interaction matrix using K-nearest neighbors (KNN)

Description

Computes a spatial interaction matrix where each entry quantifies the number of neighboring cells from a given cluster (columns) that are among the k-nearest neighbors of cells in another cluster (rows). This provides a summary of spatial proximity and enrichment of neighboring clusters around each focal cluster.

Usage

```
computeSpatialInteractionMatrix(data = NULL, cluster_col = NULL, k = 10)
```

Arguments

`data`	A `Seurat` object, a `SpatialExperiment` object, or a data frame containing spatial coordinates.
`cluster_col`	Character scalar specifying the metadata column name containing cluster assignments. If `NULL`, a default is used depending on the input object type: • `"seurat_clusters"` for `Seurat` objects • `"cluster"` for `SpatialExperiment` objects • `"cluster"` for `data.frame` objects
`k`	Integer. Number of nearest neighbors to use for each cell. Default is 10.

Details

The matrix is built by identifying the k nearest neighbors for each cell based on spatial coordinates, and then tabulating the cluster identities of those neighbors with respect to the cluster identity of the focal cell.

Value

A numeric matrix where rows represent focal clusters and columns represent neighboring clusters. Each cell in the matrix indicates how frequently a neighbor cluster appears among the k-nearest neighbors of cells from the focal cluster.

Examples

```
# Load coordinates
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

# Compute interaction matrix using all cells
interaction_matrix <- computeSpatialInteractionMatrix(coords)
head(interaction_matrix)

# Compute interaction matrix for cells inside boundaries
boundary_points <- getBoundary(
    data = coords, one_cluster = 2,
    eps = 120, minPts = 10
)
boundary_points <- subset(boundary_points, region_id == 2) # (Optional)
cells_inside <- getCellsInside(data = coords, boundary = boundary_points)
coords_sub <- subset(coords, cell %in% cells_inside$cell)
computeSpatialInteractionMatrix(coords_sub)

# Compute interaction matrix for cells inside ring region 2
ring_regions <- getRingRegion(boundary = boundary_points, dist = 100)
cells_ring <- getCellsInside(data = coords, boundary = ring_regions)
coords_sub <- subset(coords, cell %in% cells_ring$cell)
computeSpatialInteractionMatrix(coords_sub)
```

---

extractCoords *Extract spatial coordinates and cluster information*

Description

Extract spatial coordinates and (optionally) cluster assignments from a `Seurat` object, a `SpatialExperiment` object, or a user-supplied data frame.

Usage

```
extractCoords(data, cluster_col = NULL, extract_cluster = TRUE, ...)
```

Arguments

data	A Seurat object, a SpatialExperiment object, or a data frame containing spatial coordinates.
cluster_col	Character scalar specifying the metadata column name containing cluster assignments. If NULL, a default is used depending on the input object type: • "seurat_clusters" for Seurat objects • "cluster" for SpatialExperiment objects • "cluster" for data.frame objects
extract_cluster	Logical indicating whether to extract cluster information. If FALSE, only spatial coordinates and cell IDs are returned. Default is TRUE.
...	Additional arguments (unused).

Details

This is an S3 generic with methods implemented for Seurat, SpatialExperiment, and data.frame.

Value

A data frame containing columns x, y, and cell, and optionally cluster. For Seurat and SpatialExperiment inputs, only these standardized columns are returned. For data.frame input, additional columns present in the input may also be retained.

Examples

```
coords <- readRDS(system.file(
    "extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

head(extractCoords(coords))

head(extractCoords(coords, extract_cluster = FALSE))
```

---

factorNaturalOrder *Create a factor with natural (human-friendly) ordering*

Description

Converts a character or numeric vector into a factor where the levels are ordered naturally (e.g., a1, a2, ..., a10 instead of lexicographically as a1, a10, a2, ...). This is useful for plotting or labeling grouped data where numeric substrings should follow numeric order.

Usage

```
factorNaturalOrder(x)
```

Arguments

x	A character or numeric vector to convert to a factor with natural order.

Value

A factor with levels sorted in natural (human-readable) order.

Examples

```
# Numeric vector
factorNaturalOrder(10:1)

# Character vector with embedded numbers
factorNaturalOrder(c("a11", "a12", "a1", "a2", "a"))
```

---

getBoundary *Extract spatial boundary points for a cluster or cell population*

Description

Identifies and returns smoothed spatial boundary points for a specified cell cluster or population using a concave hull algorithm. The function supports both single-region and multi-region boundaries. In multi-region mode, subregions can be detected automatically via DBSCAN or manually using k-means clustering. Outlier cells are first removed using a k-nearest neighbor distance filter to ensure boundary smoothness.

Usage

```
getBoundary(
  data = NULL,
  cluster_col = NULL,
  one_cluster = NULL,
  k = 5,
  distance_cutoff = 30,
  multi_region = TRUE,
  subregion_method = c("dbscan", "kmeans"),
  eps = 80,
  minPts = 10,
  n_subregions = 3
)
```

Arguments

data	A `Seurat` object, a `SpatialExperiment` object, or a data frame containing spatial coordinates.
cluster_col	Character scalar specifying the metadata column name containing cluster assignments. If `NULL`, a default is used depending on the input object type:

- "seurat_clusters" for Seurat objects
- "cluster" for SpatialExperiment objects
- "cluster" for data.frame objects

one_cluster The cluster ID (numeric or character) to extract the boundary for.

k The number of nearest neighbors to use for computing local distances. Default is 5.

distance_cutoff
A numeric threshold on the average distance to neighbors. Cells with a higher mean distance will be removed. Default is 30.

multi_region Logical. If TRUE, identifies multiple spatial subregions within the cluster. Default is TRUE.

subregion_method
Subregion detection method when multi_region = TRUE. Choose from "dbscan" (automatic) or "kmeans" (manual). Default is "dbscan".

eps Neighborhood radius for DBSCAN subregion detection. Only used if subregion_method = "dbscan". Default is 80.

minPts Minimum number of points for DBSCAN core point. Only used if subregion_method = "dbscan". Default is 10.

n_subregions Number of subregions to use if subregion_method = "kmeans". Default is 3.

Value

A data frame containing boundary points with columns x, y, and region_id. If multi_region = TRUE, multiple boundaries are returned and labeled by region_id. Otherwise, a single boundary is returned with region_id = 1.

Examples

```
# Load coordinates
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))
head(coords)

# Get boundary points of cluster 1
boundary_points <- getBoundary(data = coords, one_cluster = 1)
head(boundary_points)

# Number of subregions
table(boundary_points$region_id)

# Get boundary points of cluster 1 using kmeans method
# and manually specify subregion number
boundary_points <- getBoundary(
    data = coords,
    one_cluster = 1,
    subregion_method = "kmeans",
    n_subregions = 2
```

```
)
table(boundary_points$region_id)

# Get boundary points of cluster 1 without multiple regions
boundary_points <- getBoundary(
    data = coords,
    one_cluster = 1,
    multi_region = FALSE
)
table(boundary_points$region_id)
```

`getCellsInside` *Identify cells located within spatial boundaries or ring regions*

Description

Returns the subset of cells from the input data that fall spatially within a given boundary or ring. The boundary can be provided either as raw boundary points (from `getBoundary()`), as polygons (from `buildBoundaryPoly()`), or as ring regions (from `getRingRegion()`). The function uses spatial point-in-polygon matching via `sf::st_within`.

Usage

```
getCellsInside(data = NULL, cluster_col = NULL, boundary = NULL)
```

Arguments

`data`	A `Seurat` object, a `SpatialExperiment` object, or a data frame containing spatial coordinates.
`cluster_col`	Character scalar specifying the metadata column name containing cluster assignments. If `NULL`, a default is used depending on the input object type: • `"seurat_clusters"` for `Seurat` objects • `"cluster"` for `SpatialExperiment` objects • `"cluster"` for `data.frame` objects
`boundary`	An `sf` object (polygon or ring) or a data frame of boundary points returned by `getBoundary()`, `buildBoundaryPoly()`, or `getRingRegion()`.

Value

A data frame (tibble) of cells located inside the given spatial region(s), with region assignment in a `region_id` column.

Examples

```
# Load coordinates
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))
head(coords)

# Get cells inside boundaries
boundary_points <- getBoundary(
    data = coords, one_cluster = 2,
    eps = 120, minPts = 10
)

# Select regions of interests if needed (Optional)
boundary_points <- subset(boundary_points, region_id == 2)

cells_inside <- getCellsInside(data = coords, boundary = boundary_points)
cells_inside

# Get cells inside rings
ring_regions <- getRingRegion(boundary = boundary_points, dist = 100)
cells_ring <- getCellsInside(data = coords, boundary = ring_regions)
cells_ring
```

---

getInnerBoundary *Generate an inner boundary polygon by shrinking an existing boundary inward*

Description

Computes an inward-shifted (contracted) boundary by applying a negative spatial buffer to an existing boundary polygon. This function is a wrapper around `getOuterBoundary()` and is useful for defining an inner region around a spatial cluster or structure.

Usage

```
getInnerBoundary(boundary = NULL, dist = 50)
```

Arguments

`boundary`	A data frame of boundary points (with columns `x`, `y`, `region_id`) or an `sf` object of `POLYGON` geometries.
`dist`	A positive numeric value specifying how far inward to shrink the boundary. This is automatically converted to a negative buffer distance. Default is 50.

Details

Be careful: if the shrinkage distance is too large, the resulting geometry may become invalid or disappear entirely, especially for narrow or irregular shapes.

Value

An sf object containing the inward-shrunk polygons, one per region_id.

See Also

getOuterBoundary() for the outward (positive) buffer version.

Examples

```
# Load coordinates
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

# Get boundary polygons of cluster 2
boundary_points <- getBoundary(data = coords, one_cluster = 2)
boundary_polys <- buildBoundaryPoly(boundary_points)
plot(boundary_polys)

# Generate inner boundary with 50-unit buffer for boundary region 1
inner_boundary <- getInnerBoundary(boundary_polys)
plot(inner_boundary)
```

---

getOuterBoundary *Generate an outer or inner boundary polygon by buffering an existing boundary*

Description

Computes an expanded or shrunken boundary by applying a spatial buffer to an existing polygon. The input can be either boundary points (as returned by getBoundary()) or polygon geometries (as returned by buildBoundaryPoly()). This is useful for defining outer spatial neighborhoods or for shrinking boundaries inward to define inner regions.

Usage

```
getOuterBoundary(boundary = NULL, dist = 100)
```

Arguments

boundary	A data frame of boundary points (with columns x, y, region_id) or an sf object.
dist	A numeric value specifying the buffer distance in spatial units. Use positive values to expand (outer boundary), and negative values to shrink (inner boundary). Default is 100.

Details

Be careful: when using a negative buffer distance (for inner boundaries), polygons may collapse, become invalid, or disappear entirely if the buffer width exceeds the shape's interior size.

Value

An `sf` object of expanded outer boundary polygons with the same `region_id` values as the original input.

See Also

`getInnerBoundary()` for a simplified wrapper for inward shrinking.

Examples

```
# Load coordinates
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

# Get boundary points of cluster 2
boundary_points <- getBoundary(data = coords, one_cluster = 2)

# Build polygons from boundary points
boundary_polys <- buildBoundaryPoly(boundary_points)

# Generate outer boundaries with 100-unit buffer
outer1 <- getOuterBoundary(boundary_points, dist = 100)
outer2 <- getOuterBoundary(boundary_polys, dist = 100)

# Plot original and expanded boundaries
plot(boundary_polys)
plot(outer2, add = TRUE, border = "red")
```

`getRingRegion` *Generate ring regions between a boundary and its outer buffer*

Description

Computes spatial ring-shaped regions by subtracting the original boundary polygons from their corresponding outer buffered polygons. If the `outer_boundary` is not supplied, it will be automatically computed using `getOuterBoundary()`. This is useful for analyzing periphery-enriched cell types or gradient-based features near a boundary.

Usage

```
getRingRegion(boundary = NULL, outer_boundary = NULL, ...)
```

Arguments

boundary	A data frame of boundary points (with columns x, y, region_id) or an sf object.
outer_boundary	Optional sf object containing buffered (outer) boundary polygons. If not provided, it will be automatically computed using getOuterBoundary().
...	Additional arguments passed to getOuterBoundary() if outer_boundary is not provided.

Value

An sf object containing the ring-shaped spatial regions generated by subtracting each inner boundary polygon from its corresponding outer boundary polygon.

Examples

```
# Load coordinates
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))
head(coords)

# Get boundary points of cluster 2
boundary_points <- getBoundary(data = coords, one_cluster = 2)

# Automatically compute outer boundary and get rings
ring_regions <- getRingRegion(boundary = boundary_points, dist = 100)
plot(ring_regions)

# Or provide both inner and outer boundaries explicitly
outer <- getOuterBoundary(boundary_points, dist = 100)
rings <- getRingRegion(boundary = boundary_points, outer_boundary = outer)
plot(rings)
```

---

plotBoundary *Plot spatial cell coordinates with cluster boundaries*

Description

Plots spatial cell locations and overlays cluster or population boundaries if available. If boundary is not provided and one_cluster is specified, the boundary will be automatically generated using the getBoundary() function. This function supports plotting either all clusters or a specific cluster using sub_plot = TRUE. Boundaries can be overlaid as polygons to visualize spatial subregions.

Usage

```
plotBoundary(
  data = NULL,
  cluster_col = NULL,
  one_cluster = NULL,
  boundary = NULL,
  colors = colors15_cheng,
  point_size = 0.5,
  color_boundary = "black",
  linewidth_boundary = 1.5,
  sub_plot = FALSE,
  split_by = NULL,
  ncol = NULL,
  angle_x_label = 0,
  theme_ggplot = theme_spneigh(),
  legend_size = 2,
  ...
)
```

Arguments

data	A Seurat object, a SpatialExperiment object, or a data frame containing spatial coordinates.
cluster_col	Character scalar specifying the metadata column name containing cluster assignments. If NULL, a default is used depending on the input object type: • "seurat_clusters" for Seurat objects • "cluster" for SpatialExperiment objects • "cluster" for data.frame objects
one_cluster	The cluster ID to plot and optionally compute its boundary. Required if sub_plot = TRUE and boundary is not provided.
boundary	A data frame with columns x, y, and region_id or an sf object of POLYGON or LINESTRING geometries.
colors	A vector of cluster colors. Default uses colors15_cheng.
point_size	Numeric. Size of the points representing cells. Default is 0.5.
color_boundary	Color for boundary lines. Default is "black".
linewidth_boundary	Numeric. Line width for boundary outlines. Default is 1.5.
sub_plot	Logical. If TRUE, only cells from the specified one_cluster are plotted. If FALSE (default), all clusters are plotted.
split_by	Optional column name in data to facet the plot by (e.g., sample, condition).
ncol	Number of columns in the faceted plot when split_by is used. Passed to ggplot2::facet_wrap(). Default is NULL, which lets ggplot2 determine layout automatically.
angle_x_label	Numeric angle (in degrees) to rotate the x-axis labels. Useful for improving label readability in faceted or dense plots. Default is 0 (no rotation).

theme_ggplot	A ggplot2 theme object. Default is theme_spneigh().
legend_size	Numeric. Size of legend keys. Default is 2.
...	Additional arguments passed to getBoundary when auto-generating boundaries.

Value

A ggplot object displaying the spatial layout of cells, with optional boundary overlays.

Examples

```
# Load coordinates
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))
head(coords)

# Plot all cells without boundaries
plotBoundary(coords)

# Plot one cluster and its boundary
plotBoundary(coords, one_cluster = 2)

# Manually compute boundary and plot
boundary_points <- getBoundary(
    data = coords, one_cluster = 2,
    eps = 120, minPts = 10
)
plotBoundary(data = coords, boundary = boundary_points)

# plotBoundary for a SpatialExperiment object
logNorm_expr <- readRDS(system.file("extdata", "LogNormExpr.rds",
    package = "SpNeigh"
))
coords_sub <- subset(coords, cluster %in% c("0", "2"))
coords_sub <- as.matrix(coords_sub[, c("x", "y")])
metadata_sub <- subset(
    coords[, c("cell", "cluster")],
    cluster %in% c("0", "2")
)

spe <- SpatialExperiment::SpatialExperiment(
    assay = list("logcounts" = logNorm_expr),
    colData = metadata_sub,
    spatialCoords = coords_sub
)

plotBoundary(data = spe, one_cluster = 2)

# plotBoundary for a Seurat object
seu_sp <- Seurat::CreateSeuratObject(
    assay = "Spatial",
    counts = logNorm_expr,
```

```
    meta.data = metadata_sub
)
SeuratObject::LayerData(seu_sp,
    assay = "Spatial",
    layer = "data"
) <- logNorm_expr

cents <- SeuratObject::CreateCentroids(coords_sub[, c("x", "y")])
fov <- SeuratObject::CreateFOV(
    coords = list("centroids" = cents),
    type = c("centroids"),
    assay = "Spatial"
)
seu_sp[["fov"]] <- fov

seu_sp$seurat_clusters <- seu_sp$cluster

plotBoundary(data = seu_sp, one_cluster = 2, eps = 120)
```

`plotCellsInside` *Plot cells located within spatial boundaries or ring regions*

Description

Visualizes cells that fall within defined spatial regions (boundaries or rings), typically obtained using the `getCellsInside()` function. The cells are colored by cluster, and the function offers two plotting modes: using `geom_sf()` (with fixed 1:1 aspect ratio) or `geom_point()` (with more flexible layout).

Usage

```
plotCellsInside(
  cells_inside = NULL,
  point_size = 0.5,
  colors = colors15_cheng,
  theme_ggplot = theme_spneigh(),
  legend_size = 2,
  fixed_aspect_ratio = TRUE
)
```

Arguments

`cells_inside`	An `sf` object of cells returned by `getCellsInside()`. Must contain `cluster` and `region_id` columns.
`point_size`	Numeric. Size of the points representing cells. Default is 0.5.
`colors`	A vector of cluster colors. Default uses `colors15_cheng`.
`theme_ggplot`	A ggplot2 theme object. Default is `theme_spneigh()`.

legend_size Numeric. Size of legend keys. Default is 2.

fixed_aspect_ratio
Logical. If TRUE, uses geom_sf() to preserve spatial scale. If FALSE, uses geom_point() with extracted coordinates. Default is TRUE.

Value

A ggplot object showing the cells colored by cluster within spatial regions.

Examples

```
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

# Obtain boundaries
boundary_points <- getBoundary(data = coords, one_cluster = 2)

# Select regions of interests if needed (Optional)
boundary_points <- subset(boundary_points, region_id == 2)

# Plot cells inside boundaries
cells_inside <- getCellsInside(data = coords, boundary = boundary_points)
plotCellsInside(cells_inside)

# Plot cells inside rings
ring_regions <- getRingRegion(boundary = boundary_points, dist = 100)
cells_ring <- getCellsInside(data = coords, boundary = ring_regions)
plotCellsInside(cells_ring, fixed_aspect_ratio = FALSE)
```

---

plotEdge *Plot boundary edges or segmented boundary lines*

Description

Plots boundary edges as colored LINESTRING or POLYGON outlines using ggplot2. This function is especially useful for visualizing specific edges extracted from a polygon using splitBoundaryPolyByAnchor().

Usage

```
plotEdge(
  boundary_poly = NULL,
  linewidth_boundary = 1,
  theme_ggplot = theme_spneigh(),
  ...
)
```

Arguments

boundary_poly	An sf object containing POLYGON or LINESTRING geometries and a region_id column.
linewidth_boundary	Numeric value specifying the line width of the edge. Default is 1.
theme_ggplot	A ggplot2 theme object. Default is theme_spneigh().
...	Additional arguments passed to ggplot2::geom_sf().

Value

A ggplot object displaying the edge outlines colored by region_id.

Examples

```
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

# Build boundary polygon and plot its outline
boundary_points <- getBoundary(
    data = coords, one_cluster = 2,
    eps = 120, minPts = 10
)
boundary_polys <- buildBoundaryPoly(boundary_points)
plotEdge(boundary_poly = boundary_polys)

# Split a polygon into edge segments and plot
boundary_edges <- splitBoundaryPolyByAnchor(boundary_polys[1, ])
plotEdge(boundary_poly = boundary_edges)
```

---

plotExpression *Plot gene expression across spatial coordinates*

Description

Visualizes expression of selected genes on a spatial plot using cell coordinates and expression matrix. Can display expression for all cells or a subset (e.g., by cluster or manually specified cells). Also supports splitting the plot using metadata columns (e.g., cluster) and returning either a combined plot or a list of individual ggplot objects.

Usage

```
plotExpression(
  data = NULL,
  exp_mat = NULL,
  genes = NULL,
```

```
  sub_plot = FALSE,
  cluster_col = NULL,
  one_cluster = NULL,
  sub_cells = NULL,
  split_by = NULL,
  ncol = NULL,
  return_list = FALSE,
  point_size = 0.2,
  angle_x_label = 0,
  shuffle = FALSE,
  theme_ggplot = theme_spneigh()
)
```

Arguments

data	A Seurat object, a SpatialExperiment object, or a data frame containing spatial coordinates.
exp_mat	A numeric gene expression matrix with genes as rows and cells as columns. Must be of class matrix or dgCMatrix. If data is a Seurat object, exp_mat can be omitted and will be automatically extracted using Seurat::GetAssayData() from the active assay. If data is a SpatialExperiment object, exp_mat can be omitted and will be automatically extracted from SingleCellExperiment::logcounts(). The column names of exp_mat must match the cell identifiers used in downstream analyses.
genes	Character vector specifying gene names to be plotted. Must match row names in exp_mat.
sub_plot	Logical. If TRUE, only a subset of cells is plotted (based on one_cluster or sub_cells). Default is FALSE.
cluster_col	Character scalar specifying the metadata column name containing cluster assignments. If NULL, a default is used depending on the input object type: • "seurat_clusters" for Seurat objects • "cluster" for SpatialExperiment objects • "cluster" for data.frame objects
one_cluster	Optional. Cluster ID to subset cells if sub_plot = TRUE.
sub_cells	Optional. Vector of cell IDs to include in the plot if sub_plot = TRUE. If both one_cluster and sub_cells are provided, the intersection is used.
split_by	Optional. Column name in the metadata (from data) to facet the plots (e.g., by cluster).
ncol	Number of columns in the faceted plot when split_by is used. Passed to ggplot2::facet_wrap(). Default is NULL, which lets ggplot2 determine layout automatically.
return_list	Logical. If TRUE, returns a named list of individual ggplot objects per gene. If FALSE (default), plots are wrapped into a single patchwork layout.
point_size	Numeric. Size of plotted points. Default is 0.2.

angle_x_label	Numeric angle (in degrees) to rotate the x-axis labels. Useful for improving label readability in faceted or dense plots. Default is 0 (no rotation).
shuffle	Logical. If TRUE, shuffles cell order before plotting. Otherwise, cells with higher expression are plotted on top. Default is FALSE.
theme_ggplot	A ggplot2 theme object. Default is theme_spneigh().

Value

A patchwork object or a list of ggplot objects if return_list = TRUE.

Examples

```
df <- data.frame(
    x = c(rnorm(100, 1), rnorm(100, 5)),
    y = c(rnorm(100, 1), rnorm(100, 5)),
    cell = 1:200,
    cluster = rep(1:2, each = 100)
)
exp_mat <- data.frame(
    gene1 = c(runif(100, 4, 6), runif(100, 0, 2)),
    gene2 = c(runif(100, 0, 1), runif(100, 5, 8))
)

exp_mat <- t(exp_mat)
colnames(exp_mat) <- df$cell

# set a random seed when shuffle is TRUE to reproduce the plot
set.seed(123)
plotExpression(
    data = df, exp_mat = exp_mat, shuffle = TRUE,
    genes = c("gene1", "gene2"), point_size = 2
)

plotExpression(
    data = df, exp_mat = exp_mat,
    genes = "gene1", sub_plot = TRUE,
    one_cluster = 1, point_size = 2
)
```

---

plotInteractionMatrix *Plot a heatmap of a row-scaled spatial interaction matrix*

Description

Visualizes a spatial interaction matrix using a heatmap, where rows represent focal clusters and columns represent neighbor clusters. Each row is scaled using z-scores to highlight relative enrichment patterns across neighbor types. This is useful for detecting spatial proximity patterns between cell populations.

Usage

```
plotInteractionMatrix(
  interaction_matrix = NULL,
  low_color = "blue",
  mid_color = "white",
  high_color = "red",
  angle_x_label = 45,
  title = "Row-Scaled Interaction Matrix (Z-scores)"
)
```

Arguments

interaction_matrix	A numeric matrix with focal clusters as rows and neighbor clusters as columns. Typically the output from computeSpatialInteractionMatrix().
low_color	Color representing low z-score values. Default is "blue".
mid_color	Color representing the midpoint (z-score = 0). Default is "white".
high_color	Color representing high z-score values. Default is "red".
angle_x_label	Angle (in degrees) to rotate x-axis labels. Default is 45.
title	Title for the heatmap.

Value

A ggplot object representing the row-scaled heatmap of the interaction matrix.

Examples

```
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

# Obtain boundaries
boundary_points <- getBoundary(
    data = coords, one_cluster = 2,
    eps = 120, minPts = 10
)
# Select regions of interests if needed (Optional)
boundary_points <- subset(boundary_points, region_id == 2)

ring_regions <- getRingRegion(boundary = boundary_points, dist = 100)
cells_ring <- getCellsInside(data = coords, boundary = ring_regions)
coords_sub <- subset(coords, cell %in% cells_ring$cell)
interaction_matrix <- computeSpatialInteractionMatrix(coords_sub)

plotInteractionMatrix(interaction_matrix)
```

plotRegion *Plot filled spatial regions inside boundaries or rings*

Description

Creates a ggplot of spatial regions (e.g., subregions or ring areas) using filled polygons. Each region is automatically assigned a fill color based on its `region_id`. This function is commonly used to visualize the area inside spatial boundaries or surrounding rings, such as those created by `buildBoundaryPoly()` or `getRingRegion()`.

Usage

```
plotRegion(
  boundary_poly = NULL,
  alpha = 0.5,
  color_boundary = "black",
  linewidth_boundary = 1,
  theme_ggplot = theme_spneigh(),
  ...
)
```

Arguments

`boundary_poly`	An `sf` object of `POLYGON` geometries containing a `region_id` column. Typically created by `buildBoundaryPoly()` or `getRingRegion()`.
`alpha`	Numeric value controlling the transparency of the filled regions. Default is `0.5`.
`color_boundary`	Color of the region outlines. Default is `"black"`.
`linewidth_boundary`	Numeric line width of the region borders. Default is `1`.
`theme_ggplot`	A ggplot2 theme object. Default is `theme_spneigh()`.
`...`	Additional arguments passed to `ggplot2::geom_sf()`.

Value

A `ggplot` object displaying filled spatial regions by `region_id`.

Examples

```
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

# Plot filled boundary regions
boundary_points <- getBoundary(
    data = coords, one_cluster = 2,
    eps = 120, minPts = 10
)
```

```
boundary_polys <- buildBoundaryPoly(boundary_points)
plotRegion(boundary_poly = boundary_polys)

# Plot filled ring regions
ring_regions <- getRingRegion(boundary = boundary_points, dist = 100)
plotRegion(boundary_poly = ring_regions)
```

`plotSpatialExpression` *Plot average gene expression along spatial distance*

Description

Visualizes how the average expression of specified genes varies along a spatial distance gradient. The spatial distance is binned, and the average expression within each bin is plotted as a heatmap.

Usage

```
plotSpatialExpression(
  exp_mat = NULL,
  spatial_distance = NULL,
  genes = NULL,
  n_bins = 50,
  scale_method = c("none", "zscore", "minmax"),
  n_labels = 6,
  row_gap = 0.1,
  column_gap = 0,
  label_x = "Spatial distance",
  label_y = "Gene",
  theme_ggplot = theme_spneigh()
)
```

Arguments

`exp_mat`	A normalized gene expression matrix (genes x cells), either a `matrix` or `dgCMatrix`. Typically log-normalized counts, e.g., from a Seurat object.
`spatial_distance`	A named numeric vector containing the spatial distance (or weights) for each cell.
`genes`	Character vector specifying gene names to be plotted. Must match row names in `exp_mat`.
`n_bins`	Integer. Number of bins to divide the spatial distance into. Default is `50`.
`scale_method`	A string indicating how to scale the average expression values across bins for each gene. Options are:
	`"none"` No scaling (default). The average expression is plotted as-is.
	`"zscore"` Standardize expression (mean 0, SD 1) per gene using `scale()`.

	`"minmax"` Normalize expression to [0, 1] range per gene using `scales::rescale()`.
`n_labels`	Integer. Number of axis labels to show along the distance axis. Default is `6`.
`row_gap`	Numeric between 0 (inclusive) and 1 (exclusive). Gap between rows (genes) in the plot. Default is `0.1`.
`column_gap`	Numeric between 0 (inclusive) and 1 (exclusive). Gap between columns (distance bins) in the plot. Default is `0`.
`label_x`	Character. Label for the x-axis. Default is "Spatial distance".
`label_y`	Character. Label for the y-axis. Default is "Gene".
`theme_ggplot`	A ggplot2 theme object. Default is `theme_spneigh()`.

Value

A ggplot2 object displaying a heatmap of binned average gene expression across spatial distances.

Examples

```
# Example spatial expression heatmap
set.seed(1)
exp_mat <- matrix(runif(1000), nrow = 10)
rownames(exp_mat) <- paste0("Gene", 1:10)
spatial_distance <- runif(100)
plotSpatialExpression(
    exp_mat = exp_mat,
    spatial_distance = spatial_distance,
    genes = rownames(exp_mat)[1:5]
)
```

---

plotStatsBar *Bar plot of cluster statistics for cells inside boundaries or ring regions*

Description

Creates a bar plot to visualize the distribution of cells inside spatial regions (e.g., boundaries or rings), either as raw counts or proportions per cluster. The plot is faceted by `region_id` to show statistics across multiple spatial subregions.

Usage

```
plotStatsBar(
  cell_stats = NULL,
  stat_column = c("proportion", "count"),
  colors = colors15_cheng,
  angle_x_label = 0,
  theme_ggplot = theme_spneigh()
)
```

Arguments

cell_stats	A data frame containing summarized cell statistics, typically the output from statsCellsInside(). Must include columns region_id, cluster, and the specified stat_column.
stat_column	Character. Column name in cell_stats to use for the y-axis. Options are "count" (number of cells) or "proportion" (relative fraction per region).
colors	A vector of cluster colors. Default uses colors15_cheng.
angle_x_label	Numeric angle (in degrees) to rotate the x-axis labels. Useful for improving label readability in faceted or dense plots. Default is 0 (no rotation).
theme_ggplot	A ggplot2 theme object. Default is theme_spneigh().

Value

A ggplot2 object showing a faceted bar plot of cell statistics per region.

Examples

```
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

boundary_points <- getBoundary(
    data = coords, one_cluster = 2,
    eps = 120, minPts = 10
)
boundary_points <- subset(boundary_points, region_id == 2) # (Optional)
cells_inside <- getCellsInside(data = coords, boundary = boundary_points)
stats_cells <- statsCellsInside(cells_inside)

plotStatsBar(stats_cells, stat_column = "proportion")
plotStatsBar(stats_cells, stat_column = "count")
```

---

plotStatsPie *Pie chart or donut chart of cluster proportions inside spatial regions*

Description

Generates pie or donut charts to visualize the proportion of cells from different clusters within each spatial region (e.g., boundary or ring). The plot is faceted by region_id to show the composition of each spatial subregion. Optionally, percentage labels can be added with filtering based on a minimum proportion threshold.

Usage

```
plotStatsPie(
  cell_stats = NULL,
  plot_donut = FALSE,
  add_labels = TRUE,
  label_cutoff = 0.01,
  label_color = "white",
  label_size = 4,
  label_nudge_x = 0.1,
  colors = colors15_cheng
)
```

Arguments

cell_stats	A data frame containing cluster statistics per region, typically the output from statsCellsInside(). Must include columns region_id, cluster, and proportion.
plot_donut	Logical. If TRUE, a donut chart is generated; otherwise, a pie chart. Default is FALSE.
add_labels	Logical. If TRUE, percentage labels are displayed inside each slice. Default is TRUE.
label_cutoff	Numeric. Proportional threshold below which labels are hidden (e.g., 0.01 = 1%). Default is 0.01.
label_color	Character. Color of the percentage labels. Default is "white".
label_size	Numeric. Text size for percentage labels. Default is 4.
label_nudge_x	Numeric. Horizontal adjustment for label positioning. Default is 0.1.
colors	A vector of cluster colors. Default uses colors15_cheng.

Value

A ggplot2 object representing a faceted pie or donut chart per spatial region.

Examples

```
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

boundary_points <- getBoundary(
    data = coords, one_cluster = 2,
    eps = 120, minPts = 10
)
boundary_points <- subset(boundary_points, region_id == 2) # (Optional)
cells_inside <- getCellsInside(data = coords, boundary = boundary_points)
stats_cells <- statsCellsInside(cells_inside)

plotStatsPie(stats_cells, add_labels = FALSE)
plotStatsPie(stats_cells, label_cutoff = 0)
plotStatsPie(stats_cells, label_cutoff = 0.01)
```

```
plotStatsPie(stats_cells, plot_donut = TRUE)
```

---

plotWeights *Plot spatial weights for cells on a spatial plot*

Description

Visualizes spatial weights by plotting cell coordinates colored by a numeric weight value. Only cells in `data` that match the names in `weights` will be included. This is useful for visualizing spatial trends such as proximity to boundaries or centroids.

Usage

```
plotWeights(
  data = NULL,
  weights = NULL,
  point_size = 0.2,
  theme_ggplot = theme_spneigh()
)
```

Arguments

`data`	A data frame, Seurat object or SpatialExperiment object containing spatial coordinates. Must include columns: `cell`, `x`, and `y`.
`weights`	A named numeric vector of spatial weights, with cell IDs as names.
`point_size`	Numeric. Point size of the cells in the plot. Default is `0.2`.
`theme_ggplot`	A ggplot2 theme object. Default is `theme_spneigh()`.

Value

A `ggplot2` object displaying a scatter plot of cells colored by weights.

Examples

```
# Load coordinate data
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

# Cells in cluster 2
cells_c2 <- subset(coords, cluster == 2)[, "cell"]

# Compute centroid weights and plot
weights_cen <- computeCentroidWeights(data = coords, cell_ids = cells_c2)
plotWeights(data = coords, weights = weights_cen)

# Compute boundary weights and plot
```

```
boundary_points <- getBoundary(data = coords, one_cluster = 2)
weights_bon <- computeBoundaryWeights(
    data = coords, cell_ids = cells_c2,
    boundary = boundary_points
)
plotWeights(data = coords, weights = weights_bon)
```

`removeOutliers` *Removes spatial outlier cells based on local k-nearest neighbor distances*

Description

Identifies and removes spatial outliers based on local density. For each cell, the average distance to its k nearest neighbors is computed. Cells with a mean k-NN distance greater than a specified cutoff are considered outliers and removed. This helps retain densely connected cells while filtering isolated ones.

Usage

```
removeOutliers(coords, k = 5, distance_cutoff = 30)
```

Arguments

`coords`	A data frame or matrix of cell coordinates.
`k`	The number of nearest neighbors to use for computing local distances. Default is 5.
`distance_cutoff`	A numeric threshold on the average distance to neighbors. Cells with a higher mean distance will be removed. Default is 30.

Value

A data frame containing the filtered coordinates with an additional column:

- `mean_knn_dist`: The average distance to the `k` nearest neighbors for each point.

Only points with a `mean_knn_dist` less than the specified `distance_cutoff` are retained.

Examples

```
# Set seed for reproducibility
set.seed(123)

# Generate 20 clustered points
n1 <- 20
x1 <- rnorm(n1, mean = 5, sd = 0.5)
y1 <- rnorm(n1, mean = 5, sd = 0.5)
```

```
# Generate 5 outliers
n2 <- 5
x2 <- runif(n2, min = 10, max = 15)
y2 <- runif(n2, min = 10, max = 15)

# Combine clustered points and outliers
coords <- data.frame(x = c(x1, x2), y = c(y1, y2))
dim(coords)

# Plot clustered points and outlier points
plot(coords$x, coords$y,
    pch = 16,
    col = rep(c("red", "green"), times = c(20, 5))
)

# Remove outlier points with a specified knn distance cutoff
new_coords <- removeOutliers(coords, k = 5, distance_cutoff = 2)
dim(new_coords)

# Returns TRUE meaning outliers have been removed in new_coords
all((new_coords$x == x1) & (new_coords$y == y1))
```

`runLimmaDE` *Differential expression analysis between two groups of cells using limma*

Description

Performs differential expression analysis between two groups of cells using the `limma` linear modeling framework. Supports optional observation-level weights (e.g., spatial weights) and filters genes by minimum expression threshold across groups.

Usage

```
runLimmaDE(
  exp_mat = NULL,
  cells_reference = NULL,
  cells_target = NULL,
  weights = NULL,
  adj_p.value = 0.05,
  min.pct = 0
)
```

Arguments

`exp_mat`	A normalized gene expression matrix (genes x cells), either a `matrix` or `dgCMatrix`. Typically log-normalized counts, e.g., from a Seurat object.

`cells_reference`
: A character vector of cell IDs to use as the reference (baseline) group.

`cells_target`
: A character vector of cell IDs to use as the target (comparison) group.

`weights`
: Optional numeric vector of observation-level weights. Must be named with cell IDs and match the length of `cells_reference` + `cells_target`.

`adj_p.value`
: Adjusted p-value threshold for reporting differentially expressed genes. Default is `0.05`.

`min.pct`
: Minimum proportion of cells expressing the gene in either group (values between 0 and 1). Genes not meeting this threshold are excluded before testing. Default is `0`.

Value

A data frame with differentially expressed genes, sorted by absolute log fold change. Includes columns:

logFC Log2 fold change of expression (target vs. reference)

AveExpr Average expression across both groups

t, P.Value, adj.P.Val, B Statistical results from `limma`

pct.reference Proportion of reference cells expressing the gene

pct.target Proportion of target cells expressing the gene

gene Gene name (rownames from `exp_mat`)

Examples

```
# Load coordinates and log-normalized expression data
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))
logNorm_expr <- readRDS(system.file("extdata", "LogNormExpr.rds",
    package = "SpNeigh"
))

# Subset cells from cluster 0 and 2
cells_ref <- subset(coords, cluster == 0)$cell
cells_tar <- subset(coords, cluster == 2)$cell

# Run differential expression with minimum expression threshold
tab <- runLimmaDE(
    exp_mat = logNorm_expr,
    cells_reference = cells_ref,
    cells_target = cells_tar,
    min.pct = 0.25
)

head(tab[, c("gene", "logFC", "adj.P.Val", "pct.reference", "pct.target")])
```

`runSpatialDE` *Differential expression along spatial distance gradients using splines*

Description

Performs spatially-aware differential expression (DE) analysis by modeling gene expression as a smooth function of a continuous spatial covariate (e.g., distance to a boundary or centroid). Natural spline basis functions are used to capture non-linear trends in expression relative to spatial distance. This method is suitable for identifying genes whose expression varies continuously across spatial structures.

Usage

```
runSpatialDE(
  exp_mat = NULL,
  cell_ids = NULL,
  spatial_distance = NULL,
  adj_p.value = 0.05,
  df = 3
)
```

Arguments

`exp_mat`	A normalized gene expression matrix (genes x cells), either a `matrix` or `dgCMatrix`. Typically log-normalized counts, e.g., from a Seurat object.
`cell_ids`	A character vector of cell IDs (column names of `exp_mat`) used for DE analysis.
`spatial_distance`	A named numeric vector containing the spatial distance (or weights) for each cell. Must be the same length as `cell_ids`. Scaled distances are recommended.
`adj_p.value`	Adjusted p-value threshold for reporting differentially expressed genes. Default is `0.05`.
`df`	Integer. Degrees of freedom for the spline basis. Default is 3.

Value

A data frame of differentially expressed genes, including:

AveExpr, F, P.Value, adj.P.Val limma differential expression outputs

Z1, Z2, Z3 Spline coefficients (Z1 typically corresponds to linear trend)

gene Gene name (from `exp_mat`)

trend "Positive" or "Negative" trend based on the sign of `Z1`

The first spline coefficient (`Z1`) captures the main expression trend along the spatial distance.

Examples

```
# Load example data
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))
logNorm_expr <- readRDS(system.file("extdata", "LogNormExpr.rds",
    package = "SpNeigh"
))

# Identify cluster-specific cells and compute spatial weights
cells_c0 <- subset(coords, cluster == 0)$cell
bon_c0 <- getBoundary(data = coords, one_cluster = 0)
weights <- computeBoundaryWeights(
    data = coords, cell_ids = cells_c0,
    boundary = bon_c0
)

# Run spatial DE
result <- runSpatialDE(
    exp_mat = logNorm_expr, cell_ids = cells_c0,
    spatial_distance = weights
)
head(result)
```

---

safeColorPalette *Generate a safe color palette for discrete clusters*

Description

Returns a vector of visually distinct colors for plotting discrete clusters. Uses `colors15_cheng` as the base palette. If the number of clusters exceeds the base palette, additional colors are generated using `scales::hue_pal()`.

Usage

```
safeColorPalette(
  n_clusters = NULL,
  base_colors = colors15_cheng,
  verbose = TRUE
)
```

Arguments

`n_clusters`	Number of unique clusters.
`base_colors`	A character vector of base colors. Default is `colors15_cheng`.
`verbose`	Logical. If `TRUE`, displays a message when extended colors are used. Default is `TRUE`.

Value

A character vector of colors of length `n_clusters`.

Examples

```
safeColorPalette(5)
safeColorPalette(20)
```

`splineDesign` *Generate an orthonormal spline-based design matrix*

Description

Constructs an orthonormal design matrix from a numeric covariate (e.g., spatial distance) using a natural cubic spline basis. The output matrix can be used in linear modeling to capture smooth, non-linear trends along continuous variables.

Usage

```
splineDesign(x, df = 3)
```

Arguments

`x`	A numeric vector representing a continuous covariate (e.g., distance or pseudotime).
`df`	Integer. Degrees of freedom for the spline basis. Default is 3.

Details

The first column of the resulting matrix is aligned to show a positive correlation with the input vector and typically captures the main linear or monotonic trend.

Value

A numeric matrix with orthonormal columns (same number of rows as `x`). The columns represent smoothed trends extracted from the spline basis. The first column is directionally aligned with the input vector (`x`).

Examples

```
x <- seq(0, 1, length.out = 100)
Z <- splineDesign(x)
cor(Z[, 1], x) # Should be > 0

# Use Z in modeling
y <- sin(2 * pi * x) + rnorm(100, sd = 0.2)
fit <- lm(y ~ Z)
summary(fit)
```

splitBoundaryPolyByAnchor *Split a polygon boundary into two parts using anchor points*

Description

This function takes an sf POLYGON object and two anchor points (or infers them) to split the polygon boundary into two LINESTRING edge segments.

Usage

```
splitBoundaryPolyByAnchor(boundary_poly = NULL, pt1 = NULL, pt2 = NULL)
```

Arguments

boundary_poly	An sf POLYGON object.
pt1	A numeric vector of length 2 (x, y) or NULL. If NULL, the leftmost boundary point is used.
pt2	A numeric vector of length 2 (x, y) or NULL. If NULL, the rightmost boundary point is used.

Value

An sf object with two LINESTRING features and a 'region_id' column indicating edge1 and edge2.

Examples

```
# Load coordinates
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))
head(coords)

# Build boundary polygons from the boundary points
boundary_points <- getBoundary(
    data = coords, one_cluster = 2,
    subregion_method = "dbscan",
    eps = 120, minPts = 10
)
boundary_polys <- buildBoundaryPoly(boundary_points)

# Split boundary polygon 1 into two edges using leftmost
# and rightmost anchor points
boundary_edges <- splitBoundaryPolyByAnchor(boundary_polys[1, ])
plot(boundary_edges, lwd = 2)

# Split boundary polygon 1 into two edges using two anchor points
pt1 <- c(4000, 1500)
```

```
pt2 <- c(2000, 3000)
boundary_edges <- splitBoundaryPolyByAnchor(boundary_polys[1, ],
    pt1 = pt1,
    pt2 = pt2
)
plot(boundary_edges, lwd = 2)
```

`statsCellsInside` *Summarize cell counts and proportions inside spatial regions*

Description

Computes the number and proportion of cells from each cluster inside boundaries or ring regions. This function is useful for downstream visualizations such as bar plots or pie charts showing the spatial composition of cell types per region.

Usage

```
statsCellsInside(cells_inside = NULL)
```

Arguments

`cells_inside`	An `sf` object of cells returned by `getCellsInside()`. Must contain `cluster` and `region_id` columns.

Value

A data frame with one row per cluster per region, containing the following columns:

- `region_id`: Identifier for each spatial region.
- `cluster`: Cluster label of the cells.
- `count`: Number of cells from the given cluster in the region.
- `proportion`: Proportion of cells from the given cluster relative to the total number of cells in the region.

Examples

```
# Load coordinates data
coords <- readRDS(system.file("extdata", "MouseBrainCoords.rds",
    package = "SpNeigh"
))

# Get boundary and cells inside
boundary_points <- getBoundary(
    data = coords, one_cluster = 2,
    eps = 120, minPts = 10
)
```

```
# Select regions of interests if needed (Optional)
boundary_points <- subset(boundary_points, region_id == 2)

cells_inside <- getCellsInside(data = coords, boundary = boundary_points)

# Summarize cluster statistics per region
stats_cells <- statsCellsInside(cells_inside)
head(stats_cells)
```

---

theme_spneigh *Custom ggplot2 theme*

Description

Provides a consistent and clean visual style for plots generated within this package. This theme builds on `theme_classic()` and adjusts text sizes and margins for better readability in figures.

Usage

```
theme_spneigh()
```

Value

A `ggplot2` theme object that can be added to ggplot visualizations.

Examples

```
library(ggplot2)
ggplot(mtcars, aes(mpg, wt)) +
    geom_point() +
    theme_spneigh()
```

Index