Package ‘SVG’

February 1, 2026

Type Package

Title Spatially Variable Genes Detection Methods for Spatial Transcriptomics

Version 1.0.0

Description A unified framework for detecting spatially variable genes (SVGs) in spatial transcriptomics data. This package integrates multiple state-of-the-art SVG detection methods including 'MERINGUE' (Moran's I based spatial autocorrelation), 'Giotto' binSpect (binary spatial enrichment test), 'SPARK-X' (non-parametric kernel-based test), and 'nnSVG' (nearest-neighbor Gaussian processes). Each method is implemented with optimized performance through vectorization, parallelization, and 'C++' acceleration where applicable. Methods are described in Miller et al. (2021) <doi:10.1101/gr.271288.120>, Dries et al. (2021) <doi:10.1186/s13059-021-02286-2>, Zhu et al. (2021) <doi:10.1186/s13059-021-02404-0>, and Weber et al. (2023) <doi:10.1038/s41467-023-39748-z>.

License MIT + file LICENSE

URL `https://github.com/Zaoqu-Liu/SVG`, `https://zaoqu-liu.github.io/SVG/`

BugReports `https://github.com/Zaoqu-Liu/SVG/issues`

Encoding UTF-8

LazyData false

Depends R (>= 4.0.0)

Imports parallel, stats, utils, methods, MASS, Rcpp (>= 1.0.0)

Suggests BRISC, geometry, RANN, CompQuadForm, BiocParallel, SpatialExperiment, SingleCellExperiment, SummarizedExperiment, spatstat.geom, spatstat.explore, testthat (>= 3.0.0), knitr, rmarkdown, covr

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.3.3

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation yes

Author Zaoqu Liu [aut, cre] (ORCID: <https://orcid.org/0000-0002-0452-742X>), SVGbench Contributors [ctb] (Original method implementations)

Maintainer Zaoqu Liu <liuzaoqu@163.com>

Repository CRAN

Date/Publication 2026-02-01 07:20:07 UTC

Contents

`ACAT_combine` *ACAT: Aggregated Cauchy Association Test*

Description

Combines multiple p-values using the Aggregated Cauchy Association Test (ACAT). This method is robust and maintains correct type I error even with correlated p-values.

Usage

```
ACAT_combine(pvals, weights = NULL)
```

Arguments

`pvals`	Numeric vector of p-values to combine.
`weights`	Numeric vector of weights. If NULL (default), equal weights are used.

Details

ACAT transforms p-values using the Cauchy distribution and combines them:

$$T = \sum_i w_i \tan(\pi(0.5 - p_i))$$

The combined p-value is then computed from the Cauchy distribution.

This method has several advantages:

- Valid even when p-values are correlated
- Computationally simple
- Handles edge cases (p = 0 or 1) gracefully

Value

A single combined p-value.

References

Liu, Y. et al. (2019) ACAT: A Fast and Powerful P Value Combination Method for Rare-Variant Analysis in Sequencing Studies. The American Journal of Human Genetics.

Examples

```
# Combine independent p-values
pvals <- c(0.01, 0.05, 0.3)
combined_p <- ACAT_combine(pvals)
print(combined_p)
```

`binarize_expression` *Binarize Gene Expression*

Description

Converts continuous gene expression values to binary (0/1) using various methods. Used by the binSpect method.

Usage

```
binarize_expression(
  expr_matrix,
  method = c("kmeans", "rank", "median", "mean"),
  rank_percent = 30,
  n_threads = 1L,
  verbose = FALSE
)
```

Arguments

`expr_matrix`	Numeric matrix of gene expression. Rows are genes, columns are spots/cells.
`method`	Character string specifying binarization method. • `"kmeans"` (default): Use k-means clustering (k=2) to separate high and low expression • `"rank"`: Binarize based on expression rank percentile • `"median"`: Values above median are set to 1 • `"mean"`: Values above mean are set to 1
`rank_percent`	Numeric. For `method = "rank"`, the percentile threshold (0-100). Values in the top `rank_percent` percent are set to 1. Default is 30.
`n_threads`	Integer. Number of threads for parallel computation. Default is 1.
`verbose`	Logical. Whether to print progress. Default is FALSE.

Details

K-means method: For each gene, k-means clustering with k=2 is applied. The cluster with higher mean expression is labeled as 1, the other as 0.

Rank method: For each gene, spots are ranked by expression. The top `rank_percent` percent are labeled as 1.

Value

Binary matrix with same dimensions as input.

Examples

```
# Create example expression matrix
expr <- matrix(rpois(1000, lambda = 10), nrow = 10, ncol = 100)
rownames(expr) <- paste0("gene_", 1:10)

# Binarize using k-means
bin_kmeans <- binarize_expression(expr, method = "kmeans")

# Binarize using rank (top 20%)
bin_rank <- binarize_expression(expr, method = "rank", rank_percent = 20)
```

`buildSpatialNetwork` *Build Spatial Neighborhood Network*

Description

Constructs a spatial neighborhood network from spatial coordinates using either Delaunay triangulation or K-nearest neighbors (KNN) approach.

Usage

```
buildSpatialNetwork(
  coords,
  method = c("delaunay", "knn"),
  k = 10L,
  filter_dist = NA,
  binary = TRUE,
  verbose = FALSE
)
```

Arguments

`coords`	Numeric matrix of spatial coordinates. Rows are spatial locations, columns are coordinate dimensions (typically x, y).
`method`	Character string specifying the network construction method. • `"delaunay"`: Delaunay triangulation (default). Creates a network where neighbors are determined by triangulation. Works well for relatively uniform spatial distributions. • `"knn"`: K-nearest neighbors. Each spot is connected to its k nearest neighbors based on Euclidean distance.
`k`	Integer. Number of nearest neighbors for KNN method. Default is 10. Ignored when `method` = `"delaunay"`.
`filter_dist`	Numeric or NA. Maximum distance threshold for neighbors. Pairs with distance > filter_dist are not considered neighbors. Default is NA (no filtering).
`binary`	Logical. If TRUE (default), return binary adjacency matrix (0/1). If FALSE, return distance-weighted adjacency matrix.
`verbose`	Logical. Whether to print progress messages. Default is FALSE.

Details

Delaunay Triangulation: Creates a network based on Delaunay triangulation, which maximizes the minimum angle of all triangles. This is a natural way to define neighbors in 2D/3D space. Requires the `geometry` package.

K-Nearest Neighbors: Connects each point to its k nearest neighbors based on Euclidean distance. More robust to irregular spatial distributions but requires choosing k. Requires the `RANN` package.

Value

A square numeric matrix representing the spatial adjacency/weight matrix. Row and column names correspond to the spatial locations (from rownames of coords).

- If `binary` = `TRUE`: Values are 1 (neighbors) or 0 (non-neighbors)
- If `binary` = `FALSE`: Values are Euclidean distances (0 for non-neighbors)

See Also

`getSpatialNeighbors_Delaunay`, `getSpatialNeighbors_KNN`

Examples

```
# Generate example coordinates
set.seed(42)
coords <- cbind(x = runif(100), y = runif(100))
rownames(coords) <- paste0("spot_", 1:100)


# Build network using Delaunay (requires geometry package)
if (requireNamespace("geometry", quietly = TRUE)) {
    W_delaunay <- buildSpatialNetwork(coords, method = "delaunay")
}

# Build network using KNN (requires RANN package)
if (requireNamespace("RANN", quietly = TRUE)) {
    W_knn <- buildSpatialNetwork(coords, method = "knn", k = 6)
}
```

`CalSVG` *Unified Interface for SVG Detection*

Description

A unified interface to run different spatially variable gene (SVG) detection methods. This function provides a consistent API for all supported methods.

Usage

```
CalSVG(
  expr_matrix,
  spatial_coords,
  method = c("meringue", "seurat", "binspect", "sparkx", "nnsvg", "markvario"),
  n_threads = 1L,
  verbose = TRUE,
  ...
)
```

Arguments

`expr_matrix`	Numeric matrix of gene expression values. Rows are genes, columns are spatial locations (spots/cells). Should be normalized (e.g., log-transformed counts).
`spatial_coords`	Numeric matrix of spatial coordinates. Rows are spatial locations, columns are x and y (and optionally z) coordinates. Row names should match column names of `expr_matrix`.
`method`	Character string specifying the SVG detection method. One of: "meringue", "seurat", "binspect", "sparkx", "nnsvg", "markvario".

`n_threads`	Integer. Number of threads for parallel computation. Default is 1. Set to higher values for faster computation on multi-core systems.
`verbose`	Logical. Whether to print progress messages. Default is TRUE.
`...`	Additional arguments passed to the specific method function.

Details

This function serves as a wrapper around the individual method functions:

- `method = "meringue"`: Calls `CalSVG_MERINGUE`
- `method = "seurat"`: Calls `CalSVG_Seurat`
- `method = "binspect"`: Calls `CalSVG_binSpect`
- `method = "sparkx"`: Calls `CalSVG_SPARKX`
- `method = "nnsvg"`: Calls `CalSVG_nnSVG`
- `method = "markvario"`: Calls `CalSVG_MarkVario`

For method-specific parameters, please refer to the documentation of individual method functions.

Value

A data.frame containing SVG detection results. The exact columns depend on the method used, but typically include:

- `gene`: Gene identifiers
- `pval` or `p.value`: Raw p-values
- `padj` or `p.adj`: Adjusted p-values (multiple testing corrected)
- Method-specific statistics (e.g., Moran's I, LR statistic, odds ratio)

See Also

`CalSVG_MERINGUE`, `CalSVG_binSpect`, `CalSVG_SPARKX`, `CalSVG_nnSVG`

Examples

```
# Simulate example data
set.seed(42)
n_genes <- 20
n_spots <- 100
expr_matrix <- matrix(rpois(n_genes * n_spots, lambda = 10),
                      nrow = n_genes, ncol = n_spots)
rownames(expr_matrix) <- paste0("gene_", seq_len(n_genes))
colnames(expr_matrix) <- paste0("spot_", seq_len(n_spots))

spatial_coords <- cbind(x = runif(n_spots, 0, 100),
                        y = runif(n_spots, 0, 100))
rownames(spatial_coords) <- colnames(expr_matrix)

# Run SPARK-X method (no external dependencies)
results <- CalSVG(expr_matrix, spatial_coords, method = "sparkx",
                    kernel_option = "single", verbose = FALSE)
head(results)
```

`CalSVG_binSpect` *binSpect: Binary Spatial Enrichment Test for SVG Detection*

Description

Detect spatially variable genes using the binSpect approach from Giotto. This method binarizes gene expression and tests for spatial enrichment of high-expressing cells using Fisher's exact test.

Identifies spatially variable genes by: 1. Binarizing gene expression (high/low) 2. Building a spatial neighborhood network 3. Testing whether high-expressing cells tend to be neighbors of other high-expressing cells more than expected by chance

Usage

```
CalSVG_binSpect(
  expr_matrix,
  spatial_coords,
  bin_method = c("kmeans", "rank"),
  rank_percent = 30,
  network_method = c("delaunay", "knn"),
  k = 10L,
  do_fisher_test = TRUE,
  adjust_method = "fdr",
  n_threads = 1L,
  verbose = TRUE
)
```

Arguments

`expr_matrix` Numeric matrix of gene expression values.

- Rows: genes
- Columns: spatial locations (spots/cells)
- Values: normalized expression (e.g., log counts or normalized counts)

`spatial_coords` Numeric matrix of spatial coordinates.

- Rows: spatial locations (must match columns of expr_matrix)
- Columns: x, y (and optionally z) coordinates

`bin_method` Character string specifying binarization method.

- `"kmeans"` (default): K-means clustering with k=2. Automatically separates high and low expression groups. Robust to different expression distributions.

	• `"rank"`: Top percentage by expression rank. More consistent across genes with different distributions. Controlled by `rank_percent` parameter.
`rank_percent`	Numeric (0-100). For `bin_method = "rank"`, the percentage of cells to classify as "high expressing". Default is 30 (top 30 • Lower values (10-20 • Higher values (40-50
`network_method`	Character string specifying spatial network construction. • `"delaunay"` (default): Delaunay triangulation • `"knn"`: K-nearest neighbors
`k`	Integer. Number of neighbors for KNN network. Default is 10.
`do_fisher_test`	Logical. Whether to perform Fisher's exact test. Default is TRUE. • TRUE: Returns p-values from Fisher's exact test • FALSE: Returns only odds ratios (faster)
`adjust_method`	Character string for p-value adjustment. Default is "fdr" (Benjamini-Hochberg). See `p.adjust()` for options.
`n_threads`	Integer. Number of parallel threads. Default is 1.
`verbose`	Logical. Print progress messages. Default is TRUE.

Details

Method Overview:

binSpect constructs a 2x2 contingency table for each gene based on:

- Cell A expression: High (1) or Low (0)
- Cell B expression: High (1) or Low (0)

For all pairs of neighboring cells (edges in the spatial network):

	Cell B Low	Cell B High
Cell A Low	n_00	n_01
Cell A High	n_10	n_11

Statistical Test: Fisher's exact test is used to test whether n_11 (both neighbors high) is greater than expected under independence.

Odds Ratio Interpretation:

- OR = 1: No spatial pattern
- OR > 1: High-expressing cells cluster together (positive spatial pattern)
- OR < 1: High-expressing cells avoid each other (negative pattern)

Advantages:

- Fast computation (no covariance matrix inversion)
- Robust to outliers through binarization

- Interpretable odds ratio statistic

Considerations:

- Binarization threshold affects results
- K-means may produce unstable results for bimodal distributions
- Rank method more stable but arbitrary threshold

Value

A data.frame with SVG detection results, sorted by significance/score. Columns:

- `gene`: Gene identifier
- `estimate`: Odds ratio from 2x2 contingency table. OR > 1 indicates spatial clustering of high-expressing cells.
- `p.value`: P-value from Fisher's exact test (if requested)
- `p.adj`: Adjusted p-value
- `score`: Combined score = -log10(p.value) * estimate
- `high_expr_count`: Number of high-expressing cells

References

Dries, R. et al. (2021) Giotto: a toolbox for integrative analysis and visualization of spatial expression data. Genome Biology.

See Also

`CalSVG`, `binarize_expression`, `buildSpatialNetwork`

Examples

```
# Load example data
data(example_svg_data)
expr <- example_svg_data$logcounts[1:20, ]
coords <- example_svg_data$spatial_coords


# Basic usage (requires RANN package)
if (requireNamespace("RANN", quietly = TRUE)) {
    results <- CalSVG_binSpect(expr, coords,
                               network_method = "knn", k = 10,
                               verbose = FALSE)
    head(results)
}
```

`CalSVG_MarkVario` *Detect SVGs using Mark Variogram Method*

Description

Identifies spatially variable genes using the mark variogram approach, as implemented in Seurat's `FindSpatiallyVariableFeatures` function with `selection.method = "markvariogram"`.

Usage

```
CalSVG_MarkVario(
  expr_matrix,
  spatial_coords,
  r_metric = 5,
  normalize = TRUE,
  n_threads = 1L,
  verbose = TRUE
)
```

Arguments

`expr_matrix`	Numeric matrix of gene expression values.
`spatial_coords`	Numeric matrix of spatial coordinates.
`r_metric`	Numeric. Distance at which to evaluate the variogram. Default is 5. Larger values capture broader spatial patterns.
`normalize`	Logical. Whether to normalize the variogram. Default is TRUE.
`n_threads`	Integer. Number of parallel threads. Default is 1.
`verbose`	Logical. Print progress messages. Default is TRUE.

Details

Method Overview:

The mark variogram measures how the correlation between gene expression values changes with distance. It is computed using the spatstat package's `markvario` function.

Interpretation:

- Lower variogram values indicate stronger spatial autocorrelation
- Values near 1 indicate random spatial distribution
- Values < 1 indicate positive spatial autocorrelation (clustering)

Note: Requires the `spatstat` package suite to be installed: `spatstat.geom` and `spatstat.explore`.

Value

A data.frame with SVG detection results. Columns:

- `gene`: Gene identifier
- `r.metric.X`: Variogram value at distance r_metric
- `rank`: Rank by variogram value (ascending, lower = more spatially variable)

References

Baddeley, A. et al. (2015) Spatial Point Patterns: Methodology and Applications with R. Chapman and Hall/CRC.

See Also

`CalSVG_Seurat`

Examples

```
# Load example data
data(example_svg_data)
expr <- example_svg_data$logcounts[1:5, ]
coords <- example_svg_data$spatial_coords

# Requires spatstat packages
if (requireNamespace("spatstat.geom", quietly = TRUE) &&
    requireNamespace("spatstat.explore", quietly = TRUE)) {
    results <- CalSVG_MarkVario(expr, coords, verbose = FALSE)
    head(results)
}
```

`CalSVG_MERINGUE` *MERINGUE: Moran's I based Spatially Variable Gene Detection*

Description

Detect spatially variable genes using the MERINGUE approach based on Moran's I spatial autocorrelation statistic.

Identifies spatially variable genes by computing Moran's I spatial autocorrelation statistic for each gene. Genes with significant positive spatial autocorrelation (similar expression values clustering together) are identified as SVGs.

Usage

```
CalSVG_MERINGUE(
  expr_matrix,
  spatial_coords,
  network_method = c("delaunay", "knn"),
  k = 10L,
  filter_dist = NA,
  alternative = c("greater", "less", "two.sided"),
  adjust_method = "BH",
  min_pct_cells = 0.05,
  n_threads = 1L,
  use_cpp = TRUE,
  verbose = TRUE
)
```

Arguments

`expr_matrix` Numeric matrix of gene expression values.

- Rows: genes
- Columns: spatial locations (spots/cells)
- Values: normalized expression (e.g., log-transformed counts)

Row names should be gene identifiers; column names should match row names of `spatial_coords`.

`spatial_coords` Numeric matrix of spatial coordinates.

- Rows: spatial locations (must match columns of expr_matrix)
- Columns: coordinate dimensions (x, y, and optionally z)

`network_method` Character string specifying how to construct the spatial neighborhood network.

- `"delaunay"` (default): Delaunay triangulation. Creates natural neighbors based on geometric triangulation. Good for relatively uniform spatial distributions.
- `"knn"`: K-nearest neighbors. Each spot connected to its k nearest neighbors. More robust for irregular distributions.

`k` Integer. Number of neighbors for KNN method. Default is 10. Ignored when `network_method = "delaunay"`.

- Smaller k (e.g., 5-6): More local patterns, faster computation
- Larger k (e.g., 15-20): Broader patterns, smoother results

`filter_dist` Numeric or NA. Maximum Euclidean distance for neighbors. Pairs with distance > filter_dist are not considered neighbors. Default is NA (no filtering). Useful for:

- Removing long-range spurious connections
- Focusing on local spatial patterns

`alternative` Character string specifying the alternative hypothesis for the Moran's I test.

- `"greater"` (default): Test for positive autocorrelation (clustering of similar values). Most appropriate for SVG detection.

- `"less"`: Test for negative autocorrelation (dissimilar values as neighbors).
- `"two.sided"`: Test for any autocorrelation.

`adjust_method`	Character string specifying p-value adjustment method for multiple testing correction. Passed to `p.adjust()`. Options include: "BH" (default, Benjamini-Hochberg), "bonferroni", "holm", "hochberg", "hommel", "BY", "fdr", "none".
`min_pct_cells`	Numeric (0-1). Minimum fraction of cells that must contribute to the spatial pattern for a gene to be retained as SVG. Default is 0.05 (5 to filter genes driven by only a few outlier cells. Set to 0 to disable this filter.
`n_threads`	Integer. Number of threads for parallel computation. Default is 1. • For large datasets: Set to number of available cores • Uses R's parallel::mclapply (not available on Windows)
`use_cpp`	Logical. Whether to use C++ implementation for faster computation. Default is TRUE. Falls back to R if C++ fails.
`verbose`	Logical. Whether to print progress messages. Default is TRUE.

Details

Method Overview:

MERINGUE uses Moran's I, a classic measure of spatial autocorrelation:

$$I = \frac{n}{W} \frac{\sum_i \sum_j w_{ij}(x_i - \bar{x})(x_j - \bar{x})}{\sum_i (x_i - \bar{x})^2}$$

where:

- n = number of spatial locations
- W = sum of all spatial weights
- w_ij = spatial weight between locations i and j
- x_i = expression value at location i

Interpretation:

- I > 0: Positive autocorrelation (similar values cluster)
- I = 0: Random spatial distribution
- I < 0: Negative autocorrelation (checkerboard pattern)

Statistical Testing: P-values are computed using normal approximation based on analytical formulas for the expected value and variance of Moran's I under the null hypothesis of complete spatial randomness.

Computational Considerations:

- Time complexity: O(n^2) for network construction, O(n*m) for testing (n = spots, m = genes)
- Memory: O(n^2) for storing spatial weights matrix
- For n > 10,000 spots, consider using KNN with small k

Value

A data.frame with SVG detection results, sorted by significance. Columns:

- `gene`: Gene identifier
- `observed`: Observed Moran's I statistic. Range: [-1, 1]. Positive values indicate clustering, negative indicate dispersion.
- `expected`: Expected Moran's I under null (approximately -1/(n-1))
- `sd`: Standard deviation under null hypothesis
- `z_score`: Standardized test statistic (observed - expected) / sd
- `p.value`: Raw p-value from normal approximation
- `p.adj`: Adjusted p-value (multiple testing corrected)

References

- Miller, B.F. et al. (2021) Characterizing spatial gene expression heterogeneity in spatially resolved single-cell transcriptomic data with nonuniform cellular densities. Genome Research.
- Moran, P.A.P. (1950) Notes on Continuous Stochastic Phenomena. Biometrika.
- Cliff, A.D. and Ord, J.K. (1981) Spatial Processes: Models & Applications. Pion.

See Also

`CalSVG` for unified interface, `buildSpatialNetwork` for network construction, `moranI_test` for individual gene testing

Examples

```
# Load example data
data(example_svg_data)
expr <- example_svg_data$logcounts[1:20, ]  # Use subset for speed
coords <- example_svg_data$spatial_coords


# Basic usage (requires RANN package for KNN)
if (requireNamespace("RANN", quietly = TRUE)) {
    results <- CalSVG_MERINGUE(expr, coords,
                               network_method = "knn", k = 10,
                               verbose = FALSE)
    head(results)

    # Get significant SVGs
    sig_genes <- results$gene[results$p.adj < 0.05]
}
```

`CalSVG_nnSVG` *nnSVG: Nearest-Neighbor Gaussian Process SVG Detection*

Description

Detect spatially variable genes using nnSVG, a method based on nearest-neighbor Gaussian processes for scalable spatial modeling.

nnSVG uses nearest-neighbor Gaussian processes (NNGP) to model spatial correlation structure in gene expression. It performs likelihood ratio tests comparing spatial vs. non-spatial models to identify SVGs.

Usage

```
CalSVG_nnSVG(
  expr_matrix,
  spatial_coords,
  X = NULL,
  n_neighbors = 10L,
  order = c("AMMD", "Sum_coords"),
  cov_model = c("exponential", "gaussian", "spherical", "matern"),
  adjust_method = "BH",
  n_threads = 1L,
  verbose = FALSE
)
```

Arguments

`expr_matrix` Numeric matrix of gene expression values.

- Rows: genes
- Columns: spatial locations (spots/cells)
- Values: log-normalized counts (e.g., from scran::logNormCounts)

`spatial_coords` Numeric matrix of spatial coordinates.

- Rows: spatial locations (must match columns of expr_matrix)
- Columns: x, y coordinates

`X` Optional numeric matrix of covariates to regress out.

- Rows: spatial locations (same order as spatial_coords)
- Columns: covariates (e.g., batch, cell type indicators)

Default is NULL (intercept-only model).

`n_neighbors` Integer. Number of nearest neighbors for NNGP model. Default is 10.

- 5-10: Faster, captures local patterns
- 15-20: Better likelihood estimates, slower

Values > 15 rarely improve results but increase computation time.

`order` Character string specifying coordinate ordering scheme.

- `"AMMD"` (default): Approximate Maximum Minimum Distance. Better for most datasets. Requires >= 65 spots.
- `"Sum_coords"`: Order by sum of coordinates. Use for very small datasets (< 65 spots).

`cov_model`	Character string specifying the covariance function. Default is "exponential". • `"exponential"`: Most commonly used, computationally stable • `"gaussian"`: Smoother patterns, requires stabilization • `"spherical"`: Finite range correlation • `"matern"`: Flexible smoothness (includes additional nu parameter)
`adjust_method`	Character string for p-value adjustment. Default is "BH" (Benjamini-Hochberg).
`n_threads`	Integer. Number of parallel threads. Default is 1. Set to number of available cores for faster computation.
`verbose`	Logical. Print progress messages. Default is FALSE.

Details

Method Overview:

nnSVG models gene expression as a Gaussian process:

$$y = X\beta + \omega + \epsilon$$

where:

- y = expression vector
- X = covariate matrix, beta = coefficients
- omega ~ GP(0, sigma^2 * C(phi)) = spatial random effect
- epsilon ~ N(0, tau^2) = non-spatial noise
- C(phi) = covariance function with range phi

Nearest-Neighbor Approximation: Full GP has O(n^3) complexity. NNGP approximates using only k nearest neighbors, reducing complexity to O(n * k^3) = O(n).

Statistical Test: Likelihood ratio test comparing:

- H0 (null): y = X*beta + epsilon (no spatial effect)
- H1 (alternative): y = X*beta + omega + epsilon (with spatial effect)

LR statistic follows chi-squared with df = 2 (testing sigma.sq and phi).

Effect Size: Proportion of spatial variance (prop_sv) measures effect size:

- prop_sv near 1: Strong spatial pattern
- prop_sv near 0: Little spatial structure

Computational Notes:

- Requires BRISC package for NNGP fitting
- O(n) complexity per gene with NNGP approximation
- Parallelization over genes provides good speedup
- Memory: O(n * k) per gene

Value

A data.frame with SVG detection results. Columns:

- `gene`: Gene identifier
- `sigma.sq`: Spatial variance estimate (sigma^2)
- `tau.sq`: Nonspatial variance estimate (tau^2, nugget)
- `phi`: Range parameter estimate (controls spatial correlation decay)
- `prop_sv`: Proportion of spatial variance = sigma.sq / (sigma.sq + tau.sq)
- `loglik`: Log-likelihood of spatial model
- `loglik_lm`: Log-likelihood of non-spatial model (linear model)
- `LR_stat`: Likelihood ratio test statistic = -2 * (loglik_lm - loglik)
- `rank`: Rank by LR statistic (1 = highest)
- `p.value`: P-value from chi-squared distribution (df = 2)
- `p.adj`: Adjusted p-value
- `runtime`: Computation time per gene (seconds)

References

Weber, L.M. et al. (2023) nnSVG for the scalable identification of spatially variable genes using nearest-neighbor Gaussian processes. Nature Communications.

Datta, A. et al. (2016) Hierarchical Nearest-Neighbor Gaussian Process Models for Large Geostatistical Datasets. JASA.

See Also

`CalSVG`, BRISC package documentation

Examples

```
# Load example data
data(example_svg_data)
expr <- example_svg_data$logcounts[1:10, ]  # Small subset
coords <- example_svg_data$spatial_coords


# Basic usage (requires BRISC package)
if (requireNamespace("BRISC", quietly = TRUE)) {
    results <- CalSVG_nnSVG(expr, coords, verbose = FALSE)
    head(results)
}
```

`CalSVG_Seurat` *Seurat-style SVG Detection Methods*

Description

Detect spatially variable genes using methods implemented in Seurat, including Moran's I with inverse distance weights and Mark Variogram.

Identifies spatially variable genes using Moran's I statistic with inverse distance squared weighting, as implemented in Seurat's `FindSpatiallyVariableFeatures` function.

Usage

```
CalSVG_Seurat(
  expr_matrix,
  spatial_coords,
  weight_scheme = c("inverse_squared", "inverse", "gaussian"),
  bandwidth = NULL,
  adjust_method = "BH",
  n_threads = 1L,
  verbose = TRUE
)
```

Arguments

`expr_matrix`	Numeric matrix of gene expression values. • Rows: genes • Columns: spatial locations (spots/cells) • Values: scaled/normalized expression (Seurat typically uses scale.data)
`spatial_coords`	Numeric matrix of spatial coordinates. • Rows: spatial locations (must match columns of expr_matrix) • Columns: x, y coordinates
`weight_scheme`	Character string specifying the distance-based weighting. • `"inverse_squared"` (default): w_ij = 1 / d_ij^2 (Seurat default, emphasizes local neighbors) • `"inverse"`: w_ij = 1 / d_ij (less emphasis on close neighbors) • `"gaussian"`: w_ij = exp(-d_ij^2 / (2 * bandwidth^2)) (controlled by bandwidth parameter)
`bandwidth`	Numeric. Bandwidth for Gaussian weighting. Default is NULL (auto-computed as median pairwise distance). Only used when `weight_scheme = "gaussian"`.
`adjust_method`	Character string for p-value adjustment. Default is "BH" (Benjamini-Hochberg).
`n_threads`	Integer. Number of parallel threads. Default is 1.
`verbose`	Logical. Print progress messages. Default is TRUE.

Details

Method Overview:

This function replicates Seurat's `FindSpatiallyVariableFeatures` with `selection.method = "moransi"`. The key difference from other Moran's I implementations is the weighting scheme:

$$w_{ij} = \frac{1}{d_{ij}^2}$$

where d_ij is the Euclidean distance between locations i and j.

Interpretation:

- Uses continuous distance-based weights (not binary network)
- Emphasizes local spatial relationships
- Higher weights for closer neighbors

Comparison with MERINGUE:

- MERINGUE: Binary adjacency (neighbors = 1, others = 0)
- Seurat: Continuous weights (1/distance^2)
- Seurat method is more sensitive to local patterns

Value

A data.frame with SVG detection results. Columns:

- `gene`: Gene identifier
- `observed`: Observed Moran's I statistic
- `expected`: Expected Moran's I under null
- `sd`: Standard deviation under null
- `p.value`: Raw p-value
- `p.adj`: Adjusted p-value
- `rank`: Rank by p-value (ascending)

References

Hao, Y. et al. (2021) Integrated analysis of multimodal single-cell data. Cell.

Stuart, T. et al. (2019) Comprehensive Integration of Single-Cell Data. Cell.

See Also

`CalSVG`, `CalSVG_MERINGUE`

Examples

```
# Load example data
data(example_svg_data)
expr <- example_svg_data$logcounts[1:20, ]
coords <- example_svg_data$spatial_coords


# Basic usage
results <- CalSVG_Seurat(expr, coords, verbose = FALSE)
head(results)
```

`CalSVG_SPARKX` *SPARK-X: Non-parametric Kernel-based SVG Detection*

Description

Detect spatially variable genes using SPARK-X, a non-parametric method that tests for spatial expression patterns using multiple kernels.

SPARK-X is a scalable non-parametric method for identifying spatially variable genes. It uses variance component score tests with multiple spatial kernels (projection, Gaussian, and cosine) to detect various types of spatial expression patterns.

Usage

```
CalSVG_SPARKX(
  expr_matrix,
  spatial_coords,
  kernel_option = c("mixture", "single"),
  adjust_method = "BY",
  n_threads = 1L,
  verbose = TRUE
)
```

Arguments

`expr_matrix` Numeric matrix of gene expression values.

- Rows: genes
- Columns: spatial locations (spots/cells)
- Values: raw counts or normalized counts (NOT log-transformed)

Note: SPARK-X works best with count data, not log-transformed data.

`spatial_coords` Numeric matrix of spatial coordinates.

- Rows: spatial locations (must match columns of expr_matrix)
- Columns: x, y coordinates

`kernel_option` Character string specifying which kernels to use.

- `"mixture"` (default): Test with all 11 kernels: 1 projection + 5 Gaussian + 5 cosine. Most comprehensive but slower. Recommended for detecting diverse spatial patterns.
- `"single"`: Test with projection kernel only. Faster but may miss some pattern types.

`adjust_method` Character string for p-value adjustment. Default is "BY" (Benjamini-Yekutieli), which is more conservative and appropriate when tests may be correlated. Other options: "BH", "bonferroni", "holm", "none".

`n_threads` Integer. Number of parallel threads. Default is 1. Higher values significantly speed up computation for large datasets.

`verbose` Logical. Print progress messages. Default is TRUE.

Details

Method Overview:

SPARK-X uses a variance component score test framework:

$$T_g = \frac{n \cdot y_g^T K y_g}{\|y_g\|^2}$$

where:

- y_g = expression vector for gene g
- K = spatial kernel matrix (derived from coordinates)
- n = number of spatial locations

Kernel Types:

- `Projection kernel`: Linear kernel based on scaled coordinates. Detects gradients and linear spatial trends.
- `Gaussian kernels`: Multiple bandwidth Gaussian RBF kernels. Detect localized hotspots of different sizes.
- `Cosine kernels`: Multiple frequency periodic kernels. Detect periodic/oscillating spatial patterns.

P-value Computation:

- Individual kernel p-values: Davies' method for quadratic forms
- Combined p-value: ACAT (Aggregated Cauchy Association Test)

Advantages:

- Non-parametric: No distributional assumptions
- Scalable: O(n) complexity, handles millions of cells
- Multiple kernels: Detects diverse pattern types
- Robust: ACAT combination handles correlated tests

Computational Considerations:

- `mixture` option: ~11x slower than `single`
- Memory: O(n) per gene, efficient for large datasets
- Parallelization provides near-linear speedup

Value

A data.frame with SVG detection results. Columns:

- `gene`: Gene identifier
- `p.value`: Combined p-value across all kernels (ACAT method)
- `p.adj`: Multiple testing adjusted p-value
- If `kernel_option = "mixture"`, additional columns for individual kernel statistics and p-values (stat_*, pval_*)

References

Zhu, J., Sun, S., & Zhou, X. (2021). SPARK-X: non-parametric modeling enables scalable and robust detection of spatial expression patterns for large spatial transcriptomic studies. Genome Biology.

See Also

`CalSVG`, `ACAT_combine`

Examples

```
# Load example data
data(example_svg_data)
expr <- example_svg_data$counts[1:20, ]  # Use counts (not log)
coords <- example_svg_data$spatial_coords

# Fast mode with single kernel (no extra dependencies)
results <- CalSVG_SPARKX(expr, coords,
                         kernel_option = "single",
                         verbose = FALSE)
head(results)
```

`data_simulation` *Simulate Spatial Transcriptomics Data with Known SVGs*

Description

Functions to generate simulated spatial transcriptomics data with known spatially variable genes (ground truth). Useful for benchmarking and testing.

Value

See individual function documentation for return values.

`example_svg_data` *Example Spatial Transcriptomics Data*

Description

A pre-generated example dataset for testing SVG detection methods. Contains 500 spots and 200 genes, with 50 known SVGs.

Format

A list with components:

counts Integer matrix (200 genes × 500 spots) of raw counts

logcounts Numeric matrix of log2(counts + 1)

spatial_coords Numeric matrix (500 spots × 2) of x, y coordinates

gene_info Data.frame with columns: gene, is_svg, pattern_type

Value

A list containing the example dataset (see Format section).

Source

Simulated using `simulate_spatial_data`

Examples

```
data(example_svg_data)
str(example_svg_data)


# Run SVG detection (requires RANN package)
if (requireNamespace("RANN", quietly = TRUE)) {
    results <- CalSVG_MERINGUE(
        example_svg_data$counts,
        example_svg_data$spatial_coords,
        verbose = FALSE
    )

    # Check accuracy
    truth <- example_svg_data$gene_info$is_svg
    detected <- results$p.adj < 0.05
    print(table(truth, detected))
}
```

`getSpatialNeighbors_Delaunay`

Build Spatial Network via Delaunay Triangulation

Description

Constructs a spatial adjacency matrix using Delaunay triangulation. Two points are considered neighbors if they share an edge in the triangulation.

Usage

```
getSpatialNeighbors_Delaunay(
  coords,
  filter_dist = NA,
  binary = TRUE,
  verbose = FALSE
)
```

Arguments

`coords`	Numeric matrix of spatial coordinates. Rows are spatial locations, columns are x, y (and optionally z) coordinates.
`filter_dist`	Numeric or NA. Maximum distance threshold for neighbors. Default is NA (no filtering).
`binary`	Logical. If TRUE (default), return binary adjacency matrix.
`verbose`	Logical. Whether to print progress messages. Default is FALSE.

Details

The function uses Delaunay triangulation from the `geometry` package. For 2D coordinates, this creates triangles. For 3D, it creates tetrahedra.

Duplicate coordinates are slightly jittered to avoid computational issues.

Value

Square numeric matrix of spatial adjacency weights.

Examples

```
set.seed(42)
coords <- cbind(x = runif(50), y = runif(50))
rownames(coords) <- paste0("spot_", 1:50)


if (requireNamespace("geometry", quietly = TRUE)) {
    W <- getSpatialNeighbors_Delaunay(coords)
}
```

---

getSpatialNeighbors_KNN

Build Spatial Network via K-Nearest Neighbors

Description

Constructs a spatial adjacency matrix using K-nearest neighbors. Each point is connected to its k nearest neighbors based on Euclidean distance.

Usage

```
getSpatialNeighbors_KNN(
  coords,
  k = 10L,
  mutual = FALSE,
  binary = TRUE,
  verbose = FALSE
)
```

Arguments

`coords`	Numeric matrix of spatial coordinates.
`k`	Integer. Number of nearest neighbors. Default is 10.
`mutual`	Logical. If TRUE, only mutual nearest neighbors are connected (both A->B and B->A must exist). Default is FALSE.

`binary`	Logical. If TRUE (default), return binary adjacency matrix. If FALSE, return distance-weighted matrix.
`verbose`	Logical. Whether to print progress messages. Default is FALSE.

Details

Uses the RANN package for efficient nearest neighbor search with KD-trees. The resulting network may be asymmetric (A is neighbor of B doesn't mean B is neighbor of A) unless `mutual = TRUE`.

Value

Square numeric matrix of spatial adjacency weights.

Examples

```
set.seed(42)
coords <- cbind(x = runif(50), y = runif(50))
rownames(coords) <- paste0("spot_", 1:50)


if (requireNamespace("RANN", quietly = TRUE)) {
    W <- getSpatialNeighbors_KNN(coords, k = 6)
}
```

`moranI` *Calculate Moran's I Statistic*

Description

Computes Moran's I spatial autocorrelation statistic for a numeric vector given a spatial weights matrix.

Usage

```
moranI(x, W, standardize = TRUE)
```

Arguments

`x`	Numeric vector of values (e.g., gene expression).
`W`	Square numeric matrix of spatial weights. Must have the same dimension as length(x).
`standardize`	Logical. If TRUE (default), row-standardize the weights matrix.

Details

Moran's I is defined as:

$$I = \frac{n}{W} \frac{\sum_i \sum_j w_{ij}(x_i - \bar{x})(x_j - \bar{x})}{\sum_i (x_i - \bar{x})^2}$$

where n is the number of observations, W is the sum of all weights, and w_ij is the weight between locations i and j.

Under the null hypothesis of no spatial autocorrelation:

- Expected value: E[I] = -1/(n-1)
- Variance is computed using the analytical formula from Cliff and Ord (1981)

Value

A list containing:

- `observed`: The observed Moran's I statistic
- `expected`: Expected value under null hypothesis of no spatial autocorrelation (typically -1/(n-1))
- `sd`: Standard deviation under null hypothesis

References

Cliff, A.D. and Ord, J.K. (1981) Spatial Processes: Models & Applications. Pion.

Examples

```
# Create example data
set.seed(42)
x <- rnorm(100)
coords <- cbind(runif(100), runif(100))


# Calculate Moran's I (requires RANN package)
if (requireNamespace("RANN", quietly = TRUE)) {
    W <- buildSpatialNetwork(coords, method = "knn", k = 6)
    result <- moranI(x, W)
    print(result)
}
```

`moranI_test` *Moran's I Test for Spatial Autocorrelation*

Description

Performs a statistical test for spatial autocorrelation using Moran's I. Returns the test statistic, expected value, standard deviation, and p-value.

Usage

```
moranI_test(
  x,
  W,
  alternative = c("greater", "less", "two.sided"),
  standardize = TRUE
)
```

Arguments

`x`	Numeric vector of values.
`W`	Square numeric matrix of spatial weights.
`alternative`	Character string specifying the alternative hypothesis. One of "greater" (default), "less", or "two.sided". • `"greater"`: Test for positive spatial autocorrelation (similar values cluster together) • `"less"`: Test for negative spatial autocorrelation (dissimilar values are neighbors) • `"two.sided"`: Test for any spatial autocorrelation
`standardize`	Logical. If TRUE (default), row-standardize weights.

Value

A named numeric vector with components:

- `observed`: Observed Moran's I
- `expected`: Expected Moran's I under null
- `sd`: Standard deviation under null
- `p.value`: P-value from normal approximation

Examples

```
set.seed(42)
x <- rnorm(100)
coords <- cbind(runif(100), runif(100))
```

```
# Test for spatial autocorrelation (requires RANN package)
if (requireNamespace("RANN", quietly = TRUE)) {
    W <- buildSpatialNetwork(coords, method = "knn", k = 6)
    result <- moranI_test(x, W)
    print(result)
}
```

`simulate_spatial_data` *Simulate Spatial Transcriptomics Data*

Description

Generates a simulated spatial transcriptomics dataset with a mixture of spatially variable genes (SVGs) and non-spatially variable genes. Uses scientifically accurate count distributions (Negative Binomial).

Usage

```
simulate_spatial_data(
  n_spots = 500,
  n_genes = 200,
  n_svg = 50,
  grid_type = c("hexagonal", "square", "random"),
  pattern_types = c("gradient", "hotspot", "periodic", "cluster"),
  mean_counts = 50,
  dispersion = 5
)
```

Arguments

`n_spots`	Integer. Number of spatial locations. Default is 500.
`n_genes`	Integer. Total number of genes. Default is 200.
`n_svg`	Integer. Number of spatially variable genes. Default is 50.
`grid_type`	Character. Type of spatial layout. • `"hexagonal"` (default): Visium-like hexagonal grid • `"square"`: Square grid • `"random"`: Random spatial distribution
`pattern_types`	Character vector. Types of spatial patterns for SVGs. Any combination of: • `"gradient"`: Linear spatial gradient • `"hotspot"`: Localized expression hotspots • `"periodic"`: Periodic/oscillating patterns • `"cluster"`: Clustered expression Default is all four types.

`mean_counts`	Numeric. Mean expression level for baseline. Default is 50.
`dispersion`	Numeric. Dispersion parameter for Negative Binomial. Smaller values = more overdispersion. Default is 5.

Details

Spatial Patterns:

- **Gradient**: Expression increases linearly along x-axis
- **Hotspot**: High expression in circular regions
- **Periodic**: Sine wave pattern along x-axis
- **Cluster**: Expression in spatially defined clusters

Count Distribution: Counts are drawn from Negative Binomial distribution:

$$X \sim NB(\mu, \phi)$$

where mu is the mean (modulated by spatial pattern) and phi is dispersion.

Value

A list containing:

- `counts`: Matrix of gene counts (genes × spots)
- `spatial_coords`: Matrix of spatial coordinates (spots × 2)
- `gene_info`: Data.frame with gene metadata including `is_svg` (TRUE/FALSE) and `pattern_type`
- `logcounts`: Log-normalized counts (log2(counts + 1))

Examples

```
# Set seed for reproducibility before calling
set.seed(42)
sim_data <- simulate_spatial_data(n_spots = 200, n_genes = 50, n_svg = 10)
str(sim_data, max.level = 1)


# Use with SVG detection (requires RANN)
if (requireNamespace("RANN", quietly = TRUE)) {
    results <- CalSVG_MERINGUE(sim_data$counts, sim_data$spatial_coords,
                               network_method = "knn", k = 10, verbose = FALSE)
}
```

`utils_spatial` *Spatial Network Utilities*

Description

Utility functions for building and manipulating spatial neighborhood networks. These functions are used by SVG detection methods to define spatial relationships between spots/cells.

Value

See individual function documentation for return values.

`utils_stats` *Statistical Utilities for SVG Detection*

Description

Statistical utility functions used by SVG detection methods, including Moran's I calculation, p-value computation, and expression binarization.

Value

See individual function documentation for return values.

Index