

# Package ‘SpaceMarkers’

April 8, 2026

**Type** Package

**Title** Spatial Interaction Markers

**Version** 2.1.0

**BugReports** <https://github.com/DeshpandeLab/SpaceMarkers/issues>

**URL** <https://github.com/DeshpandeLab/SpaceMarkers>

**Description** Spatial transcriptomic technologies have helped to resolve the connection between gene expression and the 2D orientation of tissues relative to each other. However, the limited single-cell resolution makes it difficult to highlight the most important molecular interactions in these tissues. SpaceMarkers, R/Bioconductor software, can help to find molecular interactions, by identifying genes associated with latent space interactions in spatial transcriptomics.

**Depends** R (>= 4.4.0)

**biocViews** SingleCell, GeneExpression, Software, Spatial,  
Transcriptomics

**Imports** matrixStats, matrixTests, rstatix, spatstat.explore,  
spatstat.geom, ape, hdf5r, nanoparquet, jsonlite, Matrix,  
qvalue, stats, utils, methods, ggplot2, reshape2, RColorBrewer,  
circlize, mixtools, dplyr, readbitmap, rlang, effsize, viridis

**Suggests** data.table, devtools, knitr, cowplot, rjson, rmarkdown,  
BiocStyle, testthat (>= 3.0.0), CoGAPS, ComplexHeatmap

**Enhances** BiocParallel

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Encoding** UTF-8

**LazyData** false

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**License** MIT + file LICENSE

**git\_url** <https://git.bioconductor.org/packages/SpaceMarkers>

**git\_branch** devel

**git\_last\_commit** 2b359d8

**git\_last\_commit\_date** 2025-10-29

**Repository** Bioconductor 3.23

**Date/Publication** 2026-04-07

**Author** Atul Deshpande [aut, cre] (ORCID:

<<https://orcid.org/0000-0001-5144-6924>>),

Ludmila Danilova [ctb],

Dmitrijs Lvovs [ctb] (ORCID: <<https://orcid.org/0009-0003-2152-6853>>)

**Maintainer** Atul Deshpande <[adeshpande@jhu.edu](mailto:adeshpande@jhu.edu)>

## Contents

|  |    |
|--|----|
| .calc_IM_scores . . . . .                | 3  |
| .calc_threshold . . . . .                | 4  |
| .find_genes_of_interest . . . . .        | 4  |
| .get_BTME_features . . . . .             | 5  |
| .get_cogaps_features . . . . .           | 5  |
| .get_csv_features . . . . .              | 6  |
| .get_seurat_features . . . . .           | 6  |
| .infer_method . . . . .                  | 6  |
| .pick_image . . . . .                    | 7  |
| .read_format . . . . .                   | 7  |
| .row_t_test . . . . .                    | 8  |
| calculate_gene_scores_directed . . . . . | 8  |
| calculate_gene_set_score . . . . .       | 9  |
| calculate_gene_set_specificity . . . . . | 10 |
| calculate_influence . . . . .            | 11 |
| calculate_lr_scores . . . . .            | 11 |
| calculate_overlap_directed . . . . .     | 12 |
| calculate_overlap_undirected . . . . .   | 13 |
| calculate_thresholds . . . . .           | 14 |
| curated_genes . . . . .                  | 15 |
| find_all_hotspots . . . . .              | 15 |
| find_hotspots_gmm . . . . .              | 16 |
| find_pattern_hotspots . . . . .          | 16 |
| get_im_scores . . . . .                  | 18 |
| get_interacting_genes . . . . .          | 19 |
| get_pairwise_interacting_genes . . . . . | 21 |
| get_spatial_features . . . . .           | 23 |
| get_spatial_parameters . . . . .         | 24 |
| get_spatial_params_morans_i . . . . .    | 26 |
| load10XCoords . . . . .                  | 27 |
| load10XExpr . . . . .                    | 28 |
| lrdf . . . . .                           | 28 |
| optParams . . . . .                      | 29 |
| plot_cell_interaction_circos . . . . .   | 29 |

|  |           |
|--|-----------|
| <code>.calc_IM_scores</code>                 | 3         |
| <code>plot_im_scores</code>                  | 32        |
| <code>plot_overlap_scores</code>             | 33        |
| <code>plot_source_to_target_circos</code>    | 34        |
| <code>plot_spatial_data_over_image</code>    | 37        |
| <code>plot_target_from_sources_circos</code> | 38        |
| <b>Index</b>                                 | <b>41</b> |

---

|                              |   |
|------------------------------|---|
| <code>.calc_IM_scores</code> | <i>Calculate interaction scores for a specific pattern pair</i> |
|------------------------------|---|

---

## Description

This function calculates interaction scores for a specific pattern pair using the `.classify_spots` function to determine the region of each spot.

## Usage

```
.calc_IM_scores(
  data,
  pat_hotspots,
  influence_hotspots,
  patternpair,
  avoid_confounders = FALSE,
  ...
)
```

## Arguments

|                                 |  |
|---------------------------------|--|
| <code>data</code>               | A numeric matrix with genes as rows and barcodes as columns.                                   |
| <code>pat_hotspots</code>       | A data frame with pattern hotspots, containing columns for x, y, and barcode.                  |
| <code>influence_hotspots</code> | A data frame with influence hotspots, containing columns for x, y, and barcode.                |
| <code>patternpair</code>        | A character vector of length 2 specifying the pattern pair to analyze.                         |
| <code>avoid_confounders</code>  | Logical (default=FALSE) indicating whether to avoid confounding effects due to colocalization. |
| <code>...</code>                | Additional parameters to pass to lower level functions.  |

## Value

A data frame with interaction scores for the specified pattern pair.

---

`.calc_threshold`      *Compute the threshold for identifying outlier values or hotspots*

---

### Description

This function computes the threshold for identifying outlier values or hotspots by fitting a normal mixture model to the data.

### Usage

```
.calc_threshold(df, minval = 0.01, maxval = 0.99, method = c("abs", "pct"))
```

### Arguments

|                     |   |
|---------------------|---|
| <code>df</code>     | A vector containing pattern values  |
| <code>minval</code> | Minimum value for quantile threshold  |
| <code>maxval</code> | Maximum value for quantile threshold  |
| <code>method</code> | Method to use for threshold calculation. Options are "abs" for absolute (default) and "pct" for percentile. |

### Value

A list containing the computed thresholds

---

`.find_genes_of_interest`

*.find\_genes\_of\_interest Identify genes associated with pattern interaction. This function identifies genes exhibiting significantly higher values of testMat in the Interaction region of the two patterns compared to regions with exclusive influence from either pattern. It uses Kruskal-Wallis test followed by posthoc analysis using Dunn's Test to identify the genes.*

---

### Description

`.find_genes_of_interest` Identify genes associated with pattern interaction. This function identifies genes exhibiting significantly higher values of testMat in the Interaction region of the two patterns compared to regions with exclusive influence from either pattern. It uses Kruskal-Wallis test followed by posthoc analysis using Dunn's Test to identify the genes.

### Usage

```
.find_genes_of_interest(testMat, goodGenes, region, fdr.level=0.05,
  analysis=c("enrichment", "overlap"),...)
```

**Arguments**

|                        |  |
|------------------------|--|
| <code>testMat</code>   | A matrix of counts with cells as columns and genes as rows   |
| <code>goodGenes</code> | A vector of user specified genes expected to interact a priori. The default for this is NULL as the function can find these genes itself |
| <code>region</code>    | A data frame of the reference pattern regions that overlap with the other patterns   |
| <code>fdr.level</code> | False Discovery Rate. The default value is 0.05.   |
| <code>analysis</code>  | a character string that specifies the type of analysis to carry out, whether overlap or enrichment.                                      |
| <code>...</code>       | Additional arguments to be passed to lower level functions   |

**Value**

a list of genes exhibiting significantly higher values of `testMat` in the Interaction region of the two #' patterns compared to regions with exclusive influence from either pattern.

---

`.get_BTME_features`     *.get\_BTME\_features Load features BayesTME object*

---

**Description**

`.get_BTME_features` Load features BayesTME object

**Usage**

`.get_BTME_features(hf)`

---

`.get_cogaps_features`     *.get\_cogaps\_features Load features CoGAPS object*

---

**Description**

`.get_cogaps_features` Load features CoGAPS object

**Usage**

`.get_cogaps_features(obj)`

---

`.get_csv_features`      *.get\_csv\_features Load features from dataframe*

---

**Description**

`.get_csv_features` Load features from dataframe

**Usage**

`.get_csv_features(obj)`

---

`.get_seurat_features`      *.get\_seurat\_features Load features Seurat object*

---

**Description**

`.get_seurat_features` Load features Seurat object

**Usage**

`.get_seurat_features(obj)`

---

`.infer_method`      *inferMethod Infer the method used to obtain spatial features*

---

**Description**

`inferMethod` Infer the method used to obtain spatial features

**Usage**

`.infer_method(spObject, method)`

---

*.pick\_image*                      *.pick\_image*

---

**Description**

The function picks the appropriate histology image file from the spatial directory based on the specified resolution.

**Usage**

```
.pick_image(sp_dir, res)
```

**Arguments**

sp\_dir                      path to the spatial directory  
res                          a character string specifying the resolution of the image

**Value**

a character string of the image file name

---

*.read\_format*                      *readFormat Reads a format into an R object*

---

**Description**

*readFormat Reads a format into an R object*

**Usage**

```
.read_format(path)
```

---

`.row_t_test` *Perform row-wise t-tests from scratch*

---

### Description

This function iterates over the rows of a matrix and performs a t-test comparing two groups of columns. It calculates the t-statistic, p-value, and sample sizes without relying on `stats::t.test()` for the core logic.

### Usage

```
.row_t_test(in.data, region, min_bins = 50, ...)
```

### Arguments

|                       |   |
|-----------------------|---|
| <code>in.data</code>  | A numeric matrix. Rows represent features, columns represent samples.   |
| <code>region</code>   | A factor or vector indicating the group membership for each column of <code>in.data</code> . Must have exactly two levels/unique values. Its length must equal <code>ncol(in.data)</code> . |
| <code>min_bins</code> | Minimum number of non-missing observations required in each group to perform the t-test.  |
| <code>...</code>      | Additional parameters to pass to the t-test function.   |

### Value

A matrix with rows corresponding to the features and columns: - `statistic`: The calculated t-statistic. - `p.value`: The calculated two-sided p-value. - `n1`: Number of non-missing observations in group 1 for that row. - `n2`: Number of non-missing observations in group 2 for that row.

---

`calculate_gene_scores_directed`  
*Calculate interaction scores for all pattern pairs*

---

### Description

This function calculates interaction scores for all pattern pairs using the `.calc_IM_scores` function. It can run in parallel if `BiocParallel` is available.

### Usage

```
calculate_gene_scores_directed(
  data,
  pat_hotspots,
  influence_hotspots,
  pattern_pairs = NULL,
  ...
)
```

**Arguments**

|                                 |   |
|---------------------------------|---|
| <code>data</code>               | A numeric matrix with genes as rows and barcodes as columns.  |
| <code>pat_hotspots</code>       | A data frame with pattern hotspots, containing columns for x, y, and barcode.   |
| <code>influence_hotspots</code> | A data frame with influence hotspots, containing columns for x, y, and barcode.   |
| <code>pattern_pairs</code>      | A data frame with pattern pairs to calculate interaction scores for. If NULL, all combinations of patterns in <code>pat_hotspots</code> will be used. If provided, it should have two columns with pattern names. Each row should represent a pair of patterns for which interaction scores will be calculated. |
| <code>...</code>                | Additional parameters to pass to lower level functions.   |

**Value**

A data frame with interaction scores for all pattern pairs.

---

```
calculate_gene_set_score
      calculate_gene_set_score
```

---

**Description**

Calculate the mean interaction score for a set of genes

**Usage**

```
calculate_gene_set_score(
  IMscores,
  gene_sets,
  weighted = TRUE,
  method = c("geometric_mean", "arithmetic_mean")
)
```

**Arguments**

|                        |  |
|------------------------|--|
| <code>IMscores</code>  | A matrix of interaction scores   |
| <code>gene_sets</code> | A list of gene sets, where each set is a vector of gene names  |
| <code>weighted</code>  | Logical; if TRUE, gene scores are weighted by their occurrence in multiple gene sets                               |
| <code>method</code>    | Character; specifies the aggregation method for gene set scores. Options are "geometric_mean" or "arithmetic_mean" |

**Details**

This function computes mean interaction scores for given gene sets across cell interactions. It supports both geometric and arithmetic means, and can weight gene contributions based on their presence in multiple gene sets.

**Value**

A matrix of mean interaction scores for genes in each gene set, with attributes for log p-value sums and number of genes for later fisher combination

---

```
calculate_gene_set_specificity  
    calculate_gene_set_specificity
```

---

**Description**

This function computes specificity scores for given gene sets across cell types or spatial patterns. It uses fold-change scores and p-values to weight gene contributions, and supports both geometric and arithmetic means.

**Usage**

```
calculate_gene_set_specificity(  
  data,  
  spPatterns,  
  gene_sets,  
  weighted = TRUE,  
  method = c("geometric_mean", "arithmetic_mean")  
)
```

**Arguments**

|            |  |
|------------|--|
| data       | A numeric matrix or data frame of gene expression values (genes x samples).  |
| spPatterns | A data frame or matrix containing spatial pattern information, with columns for cell types and optionally "x", "y", "barcode". |
| gene_sets  | A named list of character vectors, where each vector contains gene names for a gene set.                                       |
| weighted   | Logical; if TRUE, gene scores are weighted by their occurrence in multiple gene sets.  |
| method     | Character; specifies the aggregation method for gene set scores. Options are "geometric_mean" or "arithmetic_mean".            |

**Details****Calculate Gene Set Specificity Scores**

- Genes not present in the data are excluded.
- Genes with all zero expression are removed.
- Fold-change scores and p-values are calculated using `.calculate_all_fc_scores`.
- Scores are normalized and weighted by p-value significance.
- For each gene set, scores are aggregated using the specified method and gene weights.

**Value**

A numeric matrix of gene set specificity scores (gene sets x cell types).

**Examples**

```
# Example usage:
# gene_set_scores <- calculate_gene_set_specificity(expr_matrix, spPatterns, gene_sets)
```

---

calculate\_influence     *Compute the spatial influence of a spatial feature*

---

**Description**

This function computes the spatial influence of a specified pattern

**Usage**

```
calculate_influence(spPatterns, optParams, ...)
```

**Arguments**

|            |   |
|------------|---|
| spPatterns | A data frame containing x, y coordinates and pattern name |
| optParams  | A data frame with optimal parameters for the pattern      |
| ...        | Additional parameters for the Smooth function             |

**Value**

A data frame with the spatial influence of the specified pattern

---

calculate\_lr\_scores     *calculate\_lr\_scores*

---

**Description**

Calculate L-R pair scores using Fisher's method

**Usage**

```
calculate_lr_scores(
  ligand_scores,
  receptor_scores,
  lr_pairs,
  ligand_test = c("greater", "two.sided"),
  method = c("geometric_mean", "arithmetic_mean"),
  weighted = TRUE
)
```

**Arguments**

|                              |   |
|------------------------------|---|
| <code>ligand_scores</code>   | Output from <code>getGeneSetScore</code> for ligands  |
| <code>receptor_scores</code> | Output from <code>getGeneSetScore</code> for receptors  |
| <code>lr_pairs</code>        | Data frame with columns 'ligand' and 'receptor'   |
| <code>ligand_test</code>     | Character; specifies the type of test for ligand overexpression. Options are "greater" or "two.sided"         |
| <code>method</code>          | Character; specifies the aggregation method for L-R scores. Options are "geometric_mean" or "arithmetic_mean" |
| <code>weighted</code>        | Logical; if TRUE, L-R scores are weighted by their occurrence in multiple L-R pairs                           |

**Details**

This function computes L-R pair scores by combining ligand and receptor overexpression scores using either geometric or arithmetic mean. It can also weight L-R pairs based on their presence in multiple pairs to reduce bias from promiscuous ligands or receptors.

**Value**

Data frame with L-R scores and p-values

---

```
calculate_overlap_directed
      calculate_overlap_directed
```

---

**Description**

Calculate the overlap scores between patterns in hotspots

**Usage**

```
calculate_overlap_directed(
  pat_hotspots,
  influence_hotspots,
  patternList = NULL,
  method = c("relative-abundance", "differential-abundance", "absolute")
)
```

**Arguments**

|                                 |   |
|---------------------------------|---|
| <code>pat_hotspots</code>       | A data frame with columns x, y, barcode and pattern names |
| <code>influence_hotspots</code> | A data frame with columns x, y, barcode and pattern names |

|             |   |
|-------------|---|
| patternList | A character vector of pattern names to calculate overlap scores for. If NULL, all patterns in pat_hotspots and influence_hotspots will be used. |
| method      | The method to calculate overlapping abundance scores. Options are "relative-abundance", "differential-abundance" and "absolute"                 |

### Details

The function calculates the overlap scores between patterns hotspots using the specified method. The default method is "relative-abundance"

### Value

A data frame with columns pattern, influence and overlapping abundance

### Examples

```
hotspots <- data.frame(x = c(1,2,3,4,5),
  y = c(1,2,3,4,5),
  barcode = c("A", "B", "C", "D", "E"),
  pattern1 = c(1,0,1,0,1),
  pattern2 = c(1,1,0,0,1))
influence_hotspots <- data.frame(x = c(1,2,3,4,5),
  y = c(1,2,3,4,5),
  barcode = c("A", "B", "C", "D", "E"),
  pattern1 = c(0,1,1,0,0),
  pattern2 = c(0,1,0,1,1))
calculate_overlap_directed(pat_hotspots = hotspots, influence_hotspots = influence_hotspots)
```

---

```
calculate_overlap_undirected
  calculate_overlap_undirected
```

---

### Description

Calculate the overlap scores between patterns in hotspots

### Usage

```
calculate_overlap_undirected(
  hotspots,
  patternList = NULL,
  method = c("Szymkiewicz-Simpson", "Jaccard", "Sorensen-Dice", "Ochiai", "absolute")
)
```

### Arguments

|             |  |
|-------------|--|
| hotspots    | A data frame with columns x, y, barcode and pattern names  |
| patternList | A character vector of pattern names to calculate overlap scores for  |
| method      | The method to calculate overlap scores. Options are "Szymkiewicz-Simpson", "Jaccard", "Sorensen-Dice", "Ochiai" and "absolute" |

### Details

The function calculates the overlap scores between patterns hotspots using the specified method. The default method is "Szymkiewicz-Simpson" overlap coefficient.

### Value

A data frame with columns pattern1, pattern2 and overlapScore

### Examples

```
hotspots <- data.frame(x = c(1,2,3,4,5),
                      y = c(1,2,3,4,5),
                      barcode = c("A","B","C","D","E"),
                      pattern1 = c("pattern1",NA,"pattern1",NA,"pattern1"),
                      pattern2 = c("pattern2","pattern2",NA,NA,"pattern2"))
calculate_overlap_undirected(hotspots)
calculate_overlap_undirected(hotspots, c("pattern1","pattern2"))
```

---

calculate\_thresholds *Compute the thresholds for all columns in a data frame*

---

### Description

This function computes the thresholds for all columns in a data frame. The data frame could be an spPatterns object or an spInfluence object.

### Usage

```
calculate_thresholds(df, minvals = 0.01, maxvals = 0.99, ...)
```

### Arguments

|         |  |
|---------|--|
| df      | A data frame with pattern values (optionally with x, y, barcode columns) |
| minvals | Minimum value for quantile threshold                                     |
| maxvals | Maximum value for quantile threshold                                     |
| ...     | Additional parameters to pass to lower level functions                   |

### Value

A list containing the computed thresholds for each pattern

---

|               |   |
|---------------|---|
| curated_genes | <i>Curated Genes for example purposes</i> |
|---------------|---|

---

**Description**

A vector with genes selected based on previous runs of SpaceMarkers on the Visium 10x breast ductal carcinoma spatial transcriptomics dataset

**Format**

A vector with 114 pre-selected genes

**Value**

a vector of genes

---

|                   |   |
|-------------------|---|
| find_all_hotspots | <i>Find hotSpots for all spatial patterns</i> |
|-------------------|---|

---

**Description**

Convenience function to find hotspots for all spatial patterns

**Usage**

```
find_all_hotspots(
  spPatterns,
  params = NULL,
  outlier = "positive",
  nullSamples = 1000,
  includeSelf = TRUE,
  ...
)
```

**Arguments**

|            |  |
|------------|--|
| spPatterns | A data frame that contains the spatial coordinates and metrics for spatial features (cell types/cell processes). The column names must include 'x' and 'y' as well as the spatially varying features.          |
| params     | a named vector of the optimal sigma and threshold for a given spatial pattern. The names are should be 'sigmaOpt' and 'threshOpt'. The default value is NULL.  |
| outlier    | a character string specifying whether to apply the outlier threshold to the kernel density distribution in a one-sided manner (specify 'positive' the default) or in a two sided manner (specify 'two.sided'). |

|                          |   |
|--------------------------|---|
| <code>nullSamples</code> | a numeric values specifying the number of spatial patterns to randomly sample for a null distribution.  |
| <code>includeSelf</code> | a logic value specifying whether to consider the spatial influence the pattern has on surrounding regions only (set to FALSE), or whether to also consider the influence of the pattern itself (set to TRUE , the default). |
| <code>...</code>         | Arguments passed to methods   |

---

`find_hotspots_gmm`      *Find hotspots for all patterns or influences based on values*

---

### Description

Convenience function to find hotspots for all spatial patterns or influence dataframes based on provided thresholds

### Usage

```
find_hotspots_gmm(df, threshold = 0.1, ...)
```

### Arguments

|                        |  |
|------------------------|--|
| <code>df</code>        | A data frame with pattern values (optionally with x, y, barcode columns)   |
| <code>threshold</code> | a scalar or vector of thresholds for each column in the data frame. Either user provided or the output of <code>@calculate_thresholds</code> |
| <code>...</code>       | Additional parameters to pass to lower level functions   |

### Value

a data frame with the same dimensions as the input data frame.

---

`find_pattern_hotspots`      *Identify hotspots of spatial pattern influence*

---

### Description

This function calculates 'hotspots' which are regions of high spatial influence based on an outlier threshold from a null distribution.

**Usage**

```
find_pattern_hotspots(
  spPatterns,
  params = NULL,
  patternName = "Pattern_1",
  outlier = "positive",
  nullSamples = 1000,
  includeSelf = TRUE,
  ...
)
```

**Arguments**

|             |   |
|-------------|---|
| spPatterns  | A data frame that contains the spatial coordinates and metrics for spatial features (cell types/cell processes). The column names must include 'x' and 'y' as well as the spatially varying features.                       |
| params      | a named vector of the optimal sigma and threshold for a given spatial pattern. The names are should be 'sigmaOpt' and 'threshOpt'. The default value is NULL.   |
| patternName | a character string that specifies the pattern of interest   |
| outlier     | a character string specifying whether to apply the outlier threshold to the kernel density distribution in a one-sided manner (specify 'positive' the default) or in a two sided manner (specify 'two.sided').              |
| nullSamples | a numeric values specifying the number of spatial patterns to randomly sample for a null distribution.  |
| includeSelf | a logic value specifying whether to consider the spatial influence the pattern has on surrounding regions only (set to FALSE), or whether to also consider the influence of the pattern itself (set to TRUE , the default). |
| ...         | Arguments passed to methods   |

**Value**

a character vector with the spatial feature name if the spatial influence exceeded the threshold for that spot/cell, and NA otherwise

**See Also**

Other getIntGenes: [get\\_interacting\\_genes\(\)](#), [get\\_pairwise\\_interacting\\_genes\(\)](#)

**Examples**

```
library(SpaceMarkers)
#Visium data links
urls <- read.csv(system.file("extdata","visium_data.txt",
                           package="SpaceMarkers",mustWork = TRUE))
sp_url <- urls[["visium_url"]][2]
#Remove present Directories if any
unlink(basename(sp_url))
```

```

unlink("spatial", recursive = TRUE)
#Obtaining CoGAPS Patterns i.e Spatial Features
cogaps_result <- readRDS(system.file("extdata", "CoGAPS_result.rds",
                                   package="SpaceMarkers", mustWork = TRUE))
spFeatures <- slot(cogaps_result, "sampleFactors")
#Obtaining Spatial Coordinates
download.file(sp_url, basename(sp_url), mode = "wb")
untar(basename(sp_url))
spCoords <- load10XCoords(visiumDir = ".", version = "1.0")
rownames(spCoords) <- spCoords$barcode
#Match Dimensions
barcodes <- intersect(rownames(spFeatures), spCoords$barcode)
spCoords <- spCoords[barcodes,]
spFeatures <- spFeatures[barcodes,]
spPatterns <- cbind(spCoords, spFeatures[barcodes,])
spPatterns <- spPatterns[c("barcode", "y", "x", "Pattern_1", "Pattern_5")]
data("optParams")
hotspots <- find_pattern_hotspots(
  spPatterns = spPatterns,
  patternName = "Pattern_1",
  params = optParams["Pattern_1"],
  outlier = "positive", nullSamples = 1000, includeSelf = TRUE)
#Remove present Directories if any
unlink(basename(sp_url))
unlink("spatial", recursive = TRUE)

```

---

get\_im\_scores

*get\_im\_scores*

---

## Description

Get the interaction scores for SpaceMarkers

## Usage

```
get_im_scores(SpaceMarkers)
```

## Arguments

SpaceMarkers    A list of SpaceMarkers objects

## Value

A data frame with columns Gene and SpaceMarkersMetric

## Examples

```

example(get_pairwise_interacting_genes)
get_im_scores(SpaceMarkers)

```

---

get\_interacting\_genes *Calculate Interaction Regions and Associated Genes*

---

### Description

This function calculates statistically significant genes using a non-parametric Kruskal-Wallis test for genes in any one region of influence and a post hoc Dunn's test is used for analysis of genes between regions.

### Usage

```
get_interacting_genes(
  data,
  spPatterns,
  refPattern = "Pattern_1",
  mode = c("DE", "residual"),
  optParams = NULL,
  reconstruction = NULL,
  hotspots = NULL,
  analysis = c("enrichment", "overlap"),
  minOverlap = 50,
  ...
)
```

### Arguments

|                |   |
|----------------|---|
| data           | original spatial data matrix.   |
| spPatterns     | A data frame that contains the spatial coordinates and metrics for spatial features (cell types/cell processes). The column names must include 'x' and 'y' as well as the spatially varying features.   |
| refPattern     | a character string that specifies the pattern whose "interaction" with every other pattern we want to study. The default value is "Pattern_1".  |
| mode           | SpaceMarkers mode of operation. Possible values are "DE" (the default) or "residual".   |
| optParams      | a matrix with dimensions 2 X N, where N is the number of spatial patterns with optimal parameters. The first row contains the kernel width 'sigmaOpt' for each pattern, and the second row is the threshOpt (outlier threshold) for each pattern. Users can also input their preferred param values. The default value is NULL. |
| reconstruction | reconstruction of the data matrix from latent spaces. Required for "residual" mode.   |
| hotspots       | a vector that specifies the patterns to compare to the 'refPattern'. The default is NULL which indicates that all patterns would be compared to the 'refPattern'.   |
| analysis       | a character string that specifies the type of downstream analysis to be performed. Possible values are "enrichment" (default) and "overlap". In enrichment mode, all genes are returned, ranked by the SpaceMarkers metric. In overlap mode,  |

only the genes which are significantly overexpressed in the interaction region are returned.

`minOverlap` a number that specifies the minimum overlap between genes in two patterns to be considered for the statistical tests. The default is 50.

... Arguments passed to methods

### Value

a list of data frames with information about the interacting genes of the `refPattern` and each latent feature pattern matrix (`interacting_genes` object). There is also a data frame with all of the regions of influence for any two of patterns (the `hotspots` object).

### See Also

Other `getIntGenes`: [find\\_pattern\\_hotspots\(\)](#), [get\\_pairwise\\_interacting\\_genes\(\)](#)

### Examples

```
library(SpaceMarkers)
#Visium data links
urls <- read.csv(system.file("extdata","visium_data.txt",
package="SpaceMarkers",mustWork = TRUE))
counts_url <- urls[["visium_url"]][1]
sp_url <- urls[["visium_url"]][2]
#Remove present Directories if any
unlink(basename(sp_url))
unlink("spatial", recursive = TRUE)
files <- list.files(".")[grepl(basename(counts_url),list.files("."))]
unlink(files)
download.file(counts_url,basename(counts_url), mode = "wb")
counts_matrix<-load10XExpr(visiumDir=".",h5filename = basename(counts_url))
#Obtaining CoGAPS Patterns
cogaps_result <- readRDS(system.file("extdata","CoGAPS_result.rds",
package="SpaceMarkers",mustWork = TRUE))
features <- intersect(rownames(counts_matrix),rownames(
  slot(cogaps_result,"featureLoadings")))
barcodes <- intersect(colnames(counts_matrix),rownames(
  slot(cogaps_result,"sampleFactors")))
counts_matrix <- counts_matrix[features,barcodes]
cogaps_matrix <- slot(cogaps_result,"featureLoadings")[features,]%*%
  t(slot(cogaps_result,"sampleFactors")[barcodes,])
#Obtaining Spatial Coordinates
download.file(sp_url, basename(sp_url), mode = "wb")
untar(basename(sp_url))
spCoords <- load10XCoords(visiumDir = ".", version = "1.0")
rownames(spCoords) <- spCoords$barcode
spCoords <- spCoords[barcodes,]
spPatterns <- cbind(spCoords,slot(cogaps_result,
"sampleFactors")[barcodes,])
data("curated_genes")
spPatterns<-spPatterns[c("barcode","y","x","Pattern_1","Pattern_5")]
```

```

counts_matrix <- counts_matrix[curated_genes,]
cogaps_matrix <- cogaps_matrix[curated_genes, ]
data("optParams")
SpaceMarkersMode <- "DE"
ref_Pattern <- "Pattern_1"
SpaceMarkers_test <- get_interacting_genes(
  data=counts_matrix,reconstruction=NULL,
  optParams = optParams,
  spPatterns = spPatterns,
  refPattern = "Pattern_1",
  mode="DE",analysis="overlap")
#Remove present Directories if any
unlink(basename(sp_url))
unlink("spatial", recursive = TRUE)
files <- list.files(".")[grepl(basename(counts_url),list.files("."))]
unlink(files)

```

---

```

get_pairwise_interacting_genes
      get_pairwise_interacting_genes

```

---

### Description

Performs pairwise analysis to find genes associated with spatial interaction between pairs of spatially varying patterns.

### Usage

```

get_pairwise_interacting_genes(
  data,
  spPatterns,
  mode = c("DE", "residual"),
  optParams = NULL,
  reconstruction = NULL,
  hotspots = NULL,
  minOverlap = 50,
  analysis = c("enrichment", "overlap"),
  pattern_pairs = NULL,
  ...,
  workers = NULL
)

```

### Arguments

|            |   |
|------------|---|
| data       | original spatial data matrix.   |
| spPatterns | A data frame that contains the spatial coordinates and metrics for spatial features (cell types/cell processes). The column names must include 'x' and 'y' as well as the spatially varying features. |

|                |   |
|----------------|---|
| mode           | SpaceMarkers mode of operation. Possible values are "DE" (the default) or "residual".   |
| optParams      | a matrix with dimensions 2 X N, where N is the number of spatial patterns with optimal parameters. The first row contains the kernel width 'sigmaOpt' for each pattern, and the second row is the threshOpt (outlier threshold) for each pattern. Users can also input their preferred param values. The default value is NULL.           |
| reconstruction | reconstruction of the data matrix from latent spaces. Required for "residual" mode.   |
| hotspots       | a vector that specifies the patterns to compare to the 'refPattern'. The default is NULL which indicates that all patterns would be compared to the 'refPattern'.   |
| minOverlap     | a number that specifies the minimum overlap between genes in two patterns to be considered for the statistical tests. The default is 50.  |
| analysis       | a character string that specifies the type of downstream analysis to be performed. Possible values are "enrichment" (default) and "overlap". In enrichment mode, all genes are returned, ranked by the SpaceMarkers metric. In overlap mode, only the genes which are significantly overexpressed in the interaction region are returned. |
| pattern_pairs  | A matrix of pattern pairs to be analyzed. Default is  |
| ...            | Arguments passed to methods   |
| workers        | (optional) Number of workers to be used for parallel processing.  |

## Details

=====

## Value

a list of data frames for each pattern with 1) names of the patterns (patterns object) 2) data frame with the hotspots of influence for the two patterns (the hotspots object). 3) data frame with the genes associated with the interaction between the two patterns (interacting genes object, empty if insufficient interaction).

## See Also

Other getIntGenes: [find\\_pattern\\_hotspots\(\)](#), [get\\_interacting\\_genes\(\)](#)

## Examples

```
library(SpaceMarkers)
#Visium data links
urls <- read.csv(system.file("extdata", "visium_data.txt",
package="SpaceMarkers", mustWork = TRUE))
counts_url <- urls[["visium_url"]][1]
sp_url <- urls[["visium_url"]][2]
#Remove present Directories if any
unlink(basename(sp_url))
unlink("spatial", recursive = TRUE)
files <- list.files(".")[grepl(basename(counts_url), list.files("."))]
```

```

unlink(files)
download.file(counts_url,basename(counts_url), mode = "wb")
counts_matrix<-load10XExpr(visiumDir=".",
h5filename = basename(counts_url))
#Obtaining CoGAPS Patterns
cogaps_result <- readRDS(system.file("extdata","CoGAPS_result.rds",
package="SpaceMarkers",mustWork = TRUE))
features <- intersect(rownames(counts_matrix),rownames(
  slot(cogaps_result,"featureLoadings")))
barcodes <- intersect(colnames(counts_matrix),rownames(
  slot(cogaps_result,"sampleFactors")))
counts_matrix <- counts_matrix[features,barcodes]
cogaps_matrix <- slot(cogaps_result,"featureLoadings")[features,]%*%
  t(slot(cogaps_result,"sampleFactors")[barcodes,])
#Obtaining Spatial Coordinates
download.file(sp_url, basename(sp_url), mode = "wb")
untar(basename(sp_url))
spCoords <- load10XCoords(visiumDir = ".", version = "1.0")
rownames(spCoords) <- spCoords$barcode
spCoords <- spCoords[barcodes,]
spPatterns <- cbind(spCoords,
slot(cogaps_result,"sampleFactors")[barcodes,])
data("curated_genes")
spPatterns<-spPatterns[c("barcode","y","x","Pattern_1",
"Pattern_3","Pattern_5")]
counts_matrix <- counts_matrix[curated_genes,]
cogaps_matrix <- cogaps_matrix[curated_genes, ]
optParams <- matrix(c(6, 2, 6, 2, 6, 2), nrow = 2)
rownames(optParams) <- c("sigmaOpt","threshOpt")
colnames(optParams) <- c("Pattern_1","Pattern_3","Pattern_5")
SpaceMarkersMode <- "DE"
pattern_pairs <- matrix(c("Pattern_1", "Pattern_1",
"Pattern_3", "Pattern_5"), nrow=2)
SpaceMarkers <- get_pairwise_interacting_genes(
  data=counts_matrix,reconstruction=NULL,
  optParams = optParams,
  spPatterns = spPatterns,
  mode="DE",analysis="enrichment", pattern_pairs=pattern_pairs)
#Remove present Directories if any
unlink(basename(sp_url))
unlink("spatial", recursive = TRUE)
files <- list.files(".")[grepl(basename(counts_url),list.files("."))]
unlink(files)

```

---

get\_spatial\_features    *Load spatial features*

---

## Description

This function loads spatial features from a file containing spatial features

**Usage**

```
get_spatial_features(filePath, method = NULL, featureNames = ".")
```

**Arguments**

|              |   |
|--------------|---|
| filePath     | A string path to the location of the file containing the spatial features.  |
| method       | A string specifying the type of object to obtain spatial feature from. Default NULL, where the method is inferred based on object type. Other methods are: "CoGAPS", "Seurat", or "BayesTME". |
| featureNames | An array of strings specifying the column names corresponding to the feature names or a regex string. In the case of Seurat, all metadata columns with "_Feature" suffix are selected.        |

**Value**

a matrix of spatial features with barcodes associated with individual coordinates

**Examples**

```
library(SpaceMarkers)
#CoGAPS data filePath
filePath <- system.file("extdata", "CoGAPS_result.rds",
  package = "SpaceMarkers", mustWork = TRUE)
spFeatures <- get_spatial_features(filePath, method = "CoGAPS")
head(spFeatures)
```

---

```
get_spatial_parameters
```

*Read optimal parameters for spatial kernel density from user input or .json file*

---

**Description**

This function obtains the width of a spatial kernel density ( $\sigma$ ) from either the user input or from a scale factors .json file. The outlier threshold around the set of spots (threshold) for each pattern is specified by the user (default is 4).

**Usage**

```
get_spatial_parameters(
  spatialPatterns,
  visiumDir = ".",
  spatialDir = "spatial",
  pattern = "scalefactors_json.json",
  sigma = NULL,
  threshold = 4,
```

```

    resolution = c("fullres", "lowres", "hires"),
    ...
  )

```

## Arguments

|                 |  |
|-----------------|--|
| spatialPatterns | A data frame that contains the spatial coordinates for each cell type. The column names must include 'x' and 'y' as well as a set of numbered columns named 'Pattern_1.....N'. |
| visiumDir       | A string path specifying the location of the 10xVisium directory   |
| spatialDir      | A string path specifying the location of the spatial folder containing the .json file of the spot characteristics  |
| pattern         | A string specifying the name of the .json file   |
| sigma           | A numeric value specifying the width of the kernel density estimate to be used for smoothing   |
| threshold       | A numeric value specifying how many standard deviations above the mean of a null distribution to use an outlier threshold for identifying 'hotspots'                           |
| resolution      | A string specifying image resolution to be used for spot diameter. Can take values of "fullres" (default), "lowres" or "hires".  |
| ...             | Arguments passed to methods  |

## Value

a numeric matrix of sigmaOpts - the optimal width of the gaussian distribution, and the threshOpt - outlier threshold around the set of spots for each pattern

## Examples

```

library(SpaceMarkers)
# Create test data
cells <- c()
test_num <- 500
for(i in 1:test_num){
  cells[length(cells)+1] <- paste0("cell_",i)
}
spPatterns <- data.frame(barcode = cells,
y = runif(test_num, min=0, max=test_num),
x = runif(test_num, min=0, max=test_num),
Pattern_1 = runif(test_num, min=0, max=1),
Pattern_2 = runif(test_num, min=0, max=1) )
# Call the get_spatial_parameters function with the test data
optParams <- get_spatial_parameters(spPatterns, sigma = 10)

```

---

```
get_spatial_params_morans_i
```

*Calculate the optimal parameters from spatial kernel density for cell-cell interactions*

---

## Description

This function uses `Morans.I` to calculate the optimal width of the kernel density (`sigmaOpt`) as well as the outlier threshold around the set of spots (`threshOpt`) for a null distribution.

## Usage

```
get_spatial_params_morans_i(spatialPatterns, ...)
```

## Arguments

`spatialPatterns`

A data frame that contains the spatial coordinates for each cell type. The column names must include 'x' and 'y' as well as a set of numbered columns named 'Pattern\_1.....N'.

...

Arguments passed to methods

## Value

a numeric matrix of `sigmaOpts` - the optimal width of the gaussian distribution, and the `threshOpt` - outlier threshold around the set of spots for each pattern

## Examples

```
library(SpaceMarkers)
# Create test data
cells <- c()
test_num <- 500
for(i in 1:test_num){
  cells[length(cells)+1] <- paste0("cell_",i)
}
spPatterns <- data.frame(barcode = cells,
  y = runif(test_num, min=0, max=test_num),
  x = runif(test_num, min=0, max=test_num),
  Pattern_1 = runif(test_num, min=0, max=1),
  Pattern_2 = runif(test_num, min=0, max=1) )
# Call the get_spatial_params_morans_i function with the test data
optParams <- get_spatial_params_morans_i(spPatterns)
```

---

|               |  |
|---------------|--|
| load10XCoords | <i>Load 10x Visium Spatial Coordinates</i> |
|---------------|--|

---

## Description

This function loads spatial coordinates for each cell from a 10X Visium spatial folder.

## Usage

```
load10XCoords(  
  visiumDir,  
  resolution = c("fullres", "lowres", " hires"),  
  version = NULL  
)
```

## Arguments

|            |   |
|------------|---|
| visiumDir  | A string path to the location of the folder containing the spatial coordinates. The folder in your visiumDir must be named 'spatial' and must contain files 'scalefactors_json.json' and 'tissue_positions_list.csv.' |
| resolution | A string specifying which values to look for in the .json object. Can be either fullres (default), lowres or hires.   |
| version    | A string specifying the version of the spaceranger data.  |

## Value

a data frame of the spatial coordinates ( x and y) for each spot/cell

## Examples

```
library(SpaceMarkers)  
#Visium data links  
urls <- read.csv(system.file("extdata","visium_data.txt",  
  package = "SpaceMarkers",mustWork = TRUE))  
sp_url <- urls[["visium_url"]][2]  
# Spatial Coordinates  
download.file(sp_url, basename(sp_url), mode = "wb")  
untar(basename(sp_url))  
spCoords <- load10XCoords(visiumDir = ".", version = "1.0")  
unlink("spatial", recursive = TRUE)  
unlink("Visium_Human_Breast_Cancer_spatial.tar.gz")
```

---

|             |  |
|-------------|--|
| load10XExpr | <i>Load 10X Visium Expression Data</i> |
|-------------|--|

---

**Description**

This loads log-transformed 10X Visium expression data from standard 10X Visium folder.

**Usage**

```
load10XExpr(visiumDir = NULL, h5filename = "filtered_feature_bc_matrix.h5")
```

**Arguments**

|            |   |
|------------|---|
| visiumDir  | A string path to the h5 file with expression information. |
| h5filename | A string of the name of the h5 file in the directory.     |

**Value**

A matrix of class `dgeMatrix` or `Matrix` that contains the expression info for each sample (cells) across multiple features (genes)

**Examples**

```
library(SpaceMarkers)
#Visium data links
urls <- read.csv(system.file("extdata","visium_data.txt",
package = "SpaceMarkers",mustWork = TRUE))
counts_url <- urls[["visium_url"]][1]
#Remove present Directories if any
files <- list.files(".")[grepl(basename(counts_url),list.files("."))]
unlink(files)
download.file(counts_url,basename(counts_url), mode = "wb")
counts_matrix<-load10XExpr(visiumDir=".",h5filename = basename(counts_url))
files <- list.files(".")[grepl(basename(counts_url),list.files("."))]
unlink(files)
```

---

|      |   |
|------|---|
| Irdf | <i>Curated Ligand-receptor interaction genes A list of vectors with genes associated with ligand-receptor interactions from CellChat database</i> |
|------|---|

---

**Description**

Curated Ligand-receptor interaction genes A list of vectors with genes associated with ligand-receptor interactions from CellChat database

**Format**

A data frame with LR interaction genes

**Value**

A list of vectors of LR genes

---

optParams

*Optimal paramters of 5 patterns from CoGAPS.*

---

**Description**

A dataset with the optimal width of the gaussian distribution (sigmaOpt) and the outlier threshold around the set of spots (thresOpt) for each pattern obtained from CoGAPS. CoGAPS was ran on spatial transcriptomic data from a breast cancer sample.

**Format**

A data frame with 2 rows and 5 columns:

**Pattern\_1** immune cell pattern paramters

**Pattern\_2** Disp.1 parameters

**Pattern\_3** intraductal carcinoma (DCIS) parameters

**Pattern\_4** Disp.2 parameters

**Pattern\_5** invasive carcinoma lesion pattern paramters

**Value**

A matrix of optimal parameters for patterns identified by CoGAPS

---

plot\_cell\_interaction\_circos

*Plot Ligand-Receptor Interactions between Cell Types*

---

**Description**

Visualizes ligand-receptor interactions as a circular plot, allowing for selection of cell types, customization of segment and link appearance, and different modes of representing interactions.

**Usage**

```
plot_cell_interaction_circos(
  lr_interactions_df,
  selected_cell_types = NULL,
  cell_order = NULL,
  gap_degree_after_sector = 5,
  track_height_molecules = 0.1,
  molecule_label_cex = 0.6,
  cell_label_cex = 0.9,
  link_transparency = 0.5,
  link_connection_rou = 0.7,
  link_arrowhead_type = "big.arrow",
  link_arrowhead_width = NULL,
  link_arrowhead_length = NULL,
  score_color_palette_fun = NULL,
  split_segments_for_links = TRUE,
  link_buffer_fraction = 0.1,
  scale_link_width_by_score = FALSE,
  score_transform_for_width = function(s) s,
  use_individual_molecule_colors = TRUE,
  default_ligand_color = "lightgreen",
  default_receptor_color = "lightblue",
  individual_ligand_palette_generator = NULL,
  individual_receptor_palette_generator = NULL,
  molecule_segment_border_col = NA,
  inter_molecule_segment_gap = 0.1
)
```

**Arguments**

`lr_interactions_df`  
A data.frame with columns: ligand, receptor, source\_cell\_type, target\_cell\_type, score.

`selected_cell_types`  
Optional character vector. If provided, only these cell types and interactions *between* them will be shown.

`cell_order`  
Optional character vector for specific cell type ordering. If NULL, alphabetical order is used for the (selected) cell types.

`gap_degree_after_sector`  
Numeric, gap in degrees after each sector. Default is 5.

`track_height_molecules`  
Numeric, height of the track for molecule segments. Default is 0.1.

`molecule_label_cex`  
Numeric, cex for molecule labels. Default is 0.6.

`cell_label_cex`  
Numeric, cex for cell type labels. Default is 0.9.

`link_transparency`  
Numeric, alpha for links (0 to 1). Default is 0.5.

|                                       |   |
|---------------------------------------|---|
| link_connection_rou                   | Numeric (0-1) or vector of two for rou in circos.link. Default is 0.7.  |
| link_arrowhead_type                   | Character. Type of arrowhead (e.g., "triangle", "big.arrow"). Default is "big.arrow".   |
| link_arrowhead_width                  | Numeric. Width of the arrowhead. Default uses circlize default.   |
| link_arrowhead_length                 | Numeric. Length of the arrowhead. Default uses circlize default.  |
| score_color_palette_fun               | A function (e.g., from circlize::colorRamp2) to map scores to colors.   |
| split_segments_for_links              | Logical. If TRUE, molecule segments are sized by interaction count/score and links connect to unique sub-segments. Default is TRUE. |
| link_buffer_fraction                  | Numeric (0 to <0.5). Buffer around each link within its sub-segment. Applies if split_segments_for_links is TRUE. Default is 0.1.   |
| scale_link_width_by_score             | Logical. If TRUE (and split_segments_for_links is TRUE), link/sub-segment width is proportional to score. Default is FALSE.         |
| score_transform_for_width             | Function to transform scores for width scaling. Default is function(s) s.   |
| use_individual_molecule_colors        | Logical. If TRUE, each unique ligand/receptor name gets a distinct color. Default is TRUE.  |
| default_ligand_color                  | Character, color for all ligand segments if use_individual_molecule_colors is FALSE. Default is "lightgreen".                       |
| default_receptor_color                | Character, color for all receptor segments if use_individual_molecule_colors is FALSE. Default is "lightblue".                      |
| individual_ligand_palette_generator   | Function (takes n, returns n colors) for unique ligands. Default generates a green palette.   |
| individual_receptor_palette_generator | Function (takes n, returns n colors) for unique receptors. Default generates a blue palette.  |
| molecule_segment_border_col           | Color for the border of L/R segments. Default NA (no border).   |
| inter_molecule_segment_gap            | Numeric, gap between L/R segments on the same track. Default is 0.1.  |

**Value**

Invisibly returns NULL. The function is called for its side effect of creating a plot.

**Examples**

```
## Not run:
if (requireNamespace("circlize", quietly = TRUE) &&
    requireNamespace("RColorBrewer", quietly = TRUE)) {
  lr_data <- data.frame(
    ligand = c("LGF1", "LGF1", "LGF2", "LGF3", "LGF4", "LGF1", "LGF5", "LGF6"),
    receptor = c("REC1", "REC2A", "REC1", "REC3B", "REC1", "REC4", "REC4", "REC2A"),
    source_cell_type = c("CellA", "CellA", "CellB", "CellC",
                        "CellA", "CellD", "CellD", "CellB"),
    target_cell_type = c("CellB", "CellC", "CellB", "CellA",
                        "CellD", "CellA", "CellD", "CellC"),
    score = c(4, 0.9, 0.7, 0.95, 0.6, 0.1, 0.88, 2.5),
    stringsAsFactors = FALSE
  )
  # Basic plot with defaults
  # plot_cell_interaction_circos(lr_data)

  # Plot with selected cell types and custom order
  # plot_cell_interaction_circos(lr_data,
  #                               selected_cell_types = c("CellA", "CellB", "CellD"),
  #                               cell_order = c("CellD", "CellA", "CellB"))
}

## End(Not run)
```

---

plot\_im\_scores

*plot\_im\_scores*


---

**Description**

Plot the top SpaceMarkers IMScores

**Usage**

```
plot_im_scores(
  df,
  interaction,
  cutOff = 0,
  nGenes = 20,
  geneText = 12,
  metricText = 12,
  increments = 1,
  out = NULL
)
```

**Arguments**

df                    A data frame with columns Gene and SpaceMarkersMetric

|             |                                   |
|-------------|-----------------------------------|
| interaction | The interaction to plot           |
| cutOff      | The cut off value for the plot    |
| nGenes      | The number of genes to plot       |
| geneText    | The font size for the gene text   |
| metricText  | The font size for the metric text |
| increments  | The increments for the y-axis     |
| out         | The output path for the plot      |

### Examples

```
example(get_pairwise_interacting_genes)
plot_im_scores(get_im_scores(SpaceMarkers), "Pattern_1_Pattern_3")
```

---

```
plot_overlap_scores  plot_overlap_scores
```

---

### Description

Plot the overlap scores between patterns in hotspots

### Usage

```
plot_overlap_scores(
  df,
  title = "Spatial Overlap Scores",
  out = NULL,
  fontsize = 15
)
```

### Arguments

|          |   |
|----------|---|
| df       | A data frame with columns pattern1, pattern2 and overlapScore |
| title    | The title of the plot   |
| out      | The output path for the plot                                  |
| fontsize | The font size of the plot                                     |

### Value

A ggplot object

### Examples

```
df <- data.frame(pattern1 = c("pattern1", "pattern1", "pattern2", "pattern2"),
  pattern2 = c("pattern1", "pattern2", "pattern1", "pattern2"),
  overlapScore = c(0.5, 0.7, 0.3, 0.9))
plot_overlap_scores(df)
plot_overlap_scores(df, "Overlap Scores", "overlapScores.png", 15)
```

---

```
plot_source_to_target_circos
```

*Plot Ligand-Receptor Interactions from a Single Source to Target Cell Types*

---

## Description

Visualizes ligand-receptor interactions focusing on a single source cell type and its outgoing interactions to a specified set of target cell types. Only ligands from the source and relevant receptors on the targets are shown.

## Usage

```
plot_source_to_target_circos(
  lr_interactions_df,
  source_cell_name,
  target_cell_names,
  cell_order = NULL,
  gap_degree_after_sector = 5,
  track_height_molecules = 0.1,
  molecule_label_cex = 0.6,
  cell_label_cex = 0.9,
  link_transparency = 0.5,
  link_connection_rou = 0.7,
  link_arrowhead_type = "triangle",
  link_arrowhead_width = NULL,
  link_arrowhead_length = NULL,
  score_color_palette_fun = NULL,
  split_segments_for_links = TRUE,
  link_buffer_fraction = 0.1,
  scale_link_width_by_score = FALSE,
  score_transform_for_width = function(s) s,
  use_individual_molecule_colors = TRUE,
  default_ligand_color = "lightgreen",
  default_receptor_color = "lightblue",
  individual_ligand_palette_generator = NULL,
  individual_receptor_palette_generator = NULL,
  molecule_segment_border_col = NA,
  inter_molecule_segment_gap = 0.1
)
```

## Arguments

```
lr_interactions_df
```

A data.frame with columns: ligand, receptor, source\_cell\_type, target\_cell\_type, score.

|                                |  |
|--------------------------------|--|
| source_cell_name               | Character, name of the source cell type.   |
| target_cell_names              | Character vector, names of target cell types to show.  |
| cell_order                     | Optional character vector for specific cell type ordering. If NULL, source cell is first, then targets alphabetically.                         |
| gap_degree_after_sector        | Numeric, gap in degrees after each sector. Default is 5.   |
| track_height_molecules         | Numeric, height of the track for molecule segments. Default is 0.1.  |
| molecule_label_cex             | Numeric, cex for molecule labels. Default is 0.6.  |
| cell_label_cex                 | Numeric, cex for cell type labels. Default is 0.9.   |
| link_transparency              | Numeric, alpha for links (0 to 1). Default is 0.5.   |
| link_connection_rou            | Numeric (0-1) or vector of two for rou in <code>circos.link</code> . Default is 0.7.   |
| link_arrowhead_type            | Character. Type of arrowhead (e.g., "triangle", "big.arrow"). Default is "big.arrow".  |
| link_arrowhead_width           | Numeric. Width of the arrowhead. Default uses <code>circlize</code> default.   |
| link_arrowhead_length          | Numeric. Length of the arrowhead. Default uses <code>circlize</code> default.  |
| score_color_palette_fun        | A function (e.g., from <code>circlize::colorRamp2</code> ) to map scores to colors.  |
| split_segments_for_links       | Logical. If TRUE, molecule segments are sized by interaction count/score and links connect to unique sub-segments. Default is TRUE.            |
| link_buffer_fraction           | Numeric (0 to <0.5). Buffer around each link within its sub-segment. Applies if <code>split_segments_for_links</code> is TRUE. Default is 0.1. |
| scale_link_width_by_score      | Logical. If TRUE (and <code>split_segments_for_links</code> is TRUE), link/sub-segment width is proportional to score. Default is FALSE.       |
| score_transform_for_width      | Function to transform scores for width scaling. Default is <code>function(s) s</code> .  |
| use_individual_molecule_colors | Logical. If TRUE, each unique ligand/receptor name gets a distinct color. Default is TRUE.   |
| default_ligand_color           | Character, color for all ligand segments if <code>use_individual_molecule_colors</code> is FALSE. Default is "lightgreen".                     |
| default_receptor_color         | Character, color for all receptor segments if <code>use_individual_molecule_colors</code> is FALSE. Default is "lightblue".                    |

`individual_ligand_palette_generator`  
 Function (takes n, returns n colors) for unique ligands. Default generates a green palette.

`individual_receptor_palette_generator`  
 Function (takes n, returns n colors) for unique receptors. Default generates a blue palette.

`molecule_segment_border_col`  
 Color for the border of L/R segments. Default NA (no border).

`inter_molecule_segment_gap`  
 Numeric, gap between L/R segments on the same track. Default is 0.1.

**Value**

Invisibly returns NULL. The function is called for its side effect of creating a plot.

**Examples**

```
## Not run:
if (requireNamespace("circlize", quietly = TRUE) &&
    requireNamespace("RColorBrewer", quietly = TRUE)) {
  lr_data <- data.frame(
    ligand = c("LGF1", "LGF1", "LGF2", "LGF3", "LGF4", "LGF1", "LGF5", "LGF6", "LGF7"),
    receptor = c("REC1", "REC2A", "REC1", "REC3B", "REC1", "REC4", "REC4", "REC2A", "REC5"),
    source_cell_type = c("CellA", "CellA", "CellB", "CellC",
                        "CellA", "CellD", "CellD", "CellB", "CellA"),
    target_cell_type = c("CellB", "CellC", "CellB", "CellA",
                        "CellD", "CellA", "CellD", "CellC", "CellE"),
    score = c(4,0.9,0.7,0.95,0.6,0.1,0.88,2.5,3.0),
    stringsAsFactors = FALSE
  )
  # Plot interactions from CellA to CellB and CellC
  # plot_source_to_target_circos(lr_data,
  #                             source_cell_name = "CellA",
  #                             target_cell_names = c("CellB", "CellC"))

  # Custom order and score-based link widths
  # score_pal <- circlize::colorRamp2(c(0, 4), c("white", "blue"))
  # plot_source_to_target_circos(lr_data,
  #                             source_cell_name = "CellD",
  #                             target_cell_names = c("CellA", "CellD"), # Include autocrine
  #                             cell_order = c("CellD", "CellA"),
  #                             scale_link_width_by_score = TRUE,
  #                             score_color_palette_fun = score_pal)
}

## End(Not run)
```

---

```
plot_spatial_data_over_image
      plotSpatialDataOverImage
```

---

## Description

This function plots spatial data over the complementary histology image of varying resolutions

## Usage

```
plot_spatial_data_over_image(
  visiumDir,
  df,
  feature_col,
  barcode_col = "barcode",
  resolution = c("lowres", "hires", "fullres"),
  version = NULL,
  colors = NULL,
  point_size = 2.5,
  stroke = 0.05,
  alpha = 0.5,
  title = "Spatial Heatmap",
  bg_color = NULL,
  crop = TRUE,
  text_size = 15
)
```

## Arguments

|             |  |
|-------------|--|
| visiumDir   | directory with a spatial folder containing scalefactors_json.json, images (lowres, or hires), and coordinates (tissue_positons_(list).csv or probe.csv)          |
| df          | a dataframe with the features of interest. For example, can behotspots (character), and/or influence (numeric)   |
| feature_col | feature to plot over spots, Default: NULL  |
| barcode_col | barcode column name to match with coordinates, Default: 'barcode'  |
| resolution  | Image resolution to scale coordinates too, Default: c("lowres", "hires", "fullres")  |
| version     | Visium version. Automatically infers from load10XCoords if NULL, Default: NULL   |
| colors      | colors to be displayed over spots. If set to NULL, it automatically colors the spots red for character values and uses viridis for numeric values. Default: NULL |
| point_size  | size of spots displayed on the plot, Default: 2.5  |
| stroke      | thickness of spot outline, Default: 0.05   |
| alpha       | Transparency of the spots, Default: 0.5  |

|           |   |
|-----------|---|
| title     | Title displayed on the plot, Default: 'Spatial Heatmap' |
| bg_color  | background color of ggplot box, Default: NULL           |
| crop      | crop spatial plot to a zoomed in window, Default: TRUE  |
| text_size | size of text on the plot, Default: 15                   |

**Value**

a ggplot object

---

plot\_target\_from\_sources\_circos

*Plot Ligand-Receptor Interactions from Multiple Source to a Single Target Cell Type*

---

**Description**

Visualizes ligand-receptor interactions focusing on a single target cell type and its incoming interactions from a specified set of source cell types. Only relevant ligands from the source cells and receptors on the target cell are shown.

**Usage**

```
plot_target_from_sources_circos(
  lr_interactions_df,
  source_cell_names,
  target_cell_name,
  cell_order = NULL,
  gap_degree_after_sector = 5,
  track_height_molecules = 0.1,
  molecule_label_cex = 0.6,
  cell_label_cex = 0.9,
  link_transparency = 0.5,
  link_connection_rou = 0.7,
  link_arrowhead_type = "triangle",
  link_arrowhead_width = 0.1,
  link_arrowhead_length = 0.1,
  score_color_palette_fun = NULL,
  split_segments_for_links = TRUE,
  link_buffer_fraction = 0.1,
  scale_link_width_by_score = FALSE,
  score_transform_for_width = function(s) s,
  use_individual_molecule_colors = TRUE,
  default_ligand_color = "lightgreen",
  default_receptor_color = "lightblue",
  individual_ligand_palette_generator = NULL,
  individual_receptor_palette_generator = NULL,
```

```

    molecule_segment_border_col = NA,
    inter_molecule_segment_gap = 0.1
)

```

## Arguments

**lr\_interactions\_df**  
A data.frame with columns: ligand, receptor, source\_cell\_type, target\_cell\_type, score.

**source\_cell\_names**  
Character vector, names of source cell types to show.

**target\_cell\_name**  
Character, name of the single target cell type.

**cell\_order**  
Optional character vector for specific cell type ordering. If NULL, target cell is first, then sources alphabetically.

**gap\_degree\_after\_sector**  
Numeric, gap in degrees after each sector. Default is 5.

**track\_height\_molecules**  
Numeric, height of the track for molecule segments. Default is 0.1.

**molecule\_label\_cex**  
Numeric, cex for molecule labels. Default is 0.6.

**cell\_label\_cex**  
Numeric, cex for cell type labels. Default is 0.9.

**link\_transparency**  
Numeric, alpha for links (0 to 1). Default is 0.5.

**link\_connection\_rou**  
Numeric (0-1) or vector of two for rou in circos.link. Default is 0.7.

**link\_arrowhead\_type**  
Character. Type of arrowhead (e.g., "triangle", "big.arrow"). Default is "big.arrow".

**link\_arrowhead\_width**  
Numeric. Width of the arrowhead. Default uses circlize default.

**link\_arrowhead\_length**  
Numeric. Length of the arrowhead. Default uses circlize default.

**score\_color\_palette\_fun**  
A function (e.g., from circlize::colorRamp2) to map scores to colors.

**split\_segments\_for\_links**  
Logical. If TRUE, molecule segments are sized by interaction count/score and links connect to unique sub-segments. Default is TRUE.

**link\_buffer\_fraction**  
Numeric (0 to <0.5). Buffer around each link within its sub-segment. Applies if split\_segments\_for\_links is TRUE. Default is 0.1.

**scale\_link\_width\_by\_score**  
Logical. If TRUE (and split\_segments\_for\_links is TRUE), link/sub-segment width is proportional to score. Default is FALSE.

**score\_transform\_for\_width**  
Function to transform scores for width scaling. Default is function(s) s.

`use_individual_molecule_colors`  
 Logical. If TRUE, each unique ligand/receptor name gets a distinct color. Default is TRUE.

`default_ligand_color`  
 Character, color for all ligand segments if `use_individual_molecule_colors` is FALSE. Default is "lightgreen".

`default_receptor_color`  
 Character, color for all receptor segments if `use_individual_molecule_colors` is FALSE. Default is "lightblue".

`individual_ligand_palette_generator`  
 Function (takes n, returns n colors) for unique ligands. Default generates a green palette.

`individual_receptor_palette_generator`  
 Function (takes n, returns n colors) for unique receptors. Default generates a blue palette.

`molecule_segment_border_col`  
 Color for the border of L/R segments. Default NA (no border).

`inter_molecule_segment_gap`  
 Numeric, gap between L/R segments on the same track. Default is 0.1.

**Value**

Invisibly returns NULL. The function is called for its side effect of creating a plot.

**Examples**

```
## Not run:
if (requireNamespace("circlize", quietly = TRUE) &&
    requireNamespace("RColorBrewer", quietly = TRUE)) {
  # Use the same test data from previous examples
  test_lr_data <- data.frame(
    ligand = c("LGF1", "LGF1", "LGF2", "LGF3", "LGF4", "LGF1", "LGF5", "LGF6", "LGF5"),
    receptor = c("REC1", "REC2", "REC1", "REC1", "REC5", "REC4", "REC4", "REC6", "REC2"),
    source_cell_type = c("CellA", "CellA", "CellB", "CellC", "CellA", "CellD", "CellD", "CellE", "CellD"),
    target_cell_type = c("CellB", "CellC", "CellA", "CellD", "CellD", "CellA", "CellD", "CellF", "CellA"),
    score = c(4.0, 2.5, 3.0, 2.2, 1.5, 1.0, 3.5, 0.5, 2.8),
    stringsAsFactors = FALSE
  )
  # Plot interactions from CellB and CellD converging on CellA
  # plot_target_from_sources_circos(test_lr_data,
  #                               source_cell_names = c("CellB", "CellD"),
  #                               target_cell_name = "CellA")

  # Another example: Interactions from CellA and CellC targeting CellD
  # plot_target_from_sources_circos(test_lr_data,
  #                               source_cell_names = c("CellA", "CellC"),
  #                               target_cell_name = "CellD",
  #                               scale_link_width_by_score = TRUE)
}

## End(Not run)
```

# Index

- \* **getIntGenes**
  - find\_pattern\_hotspots, 16
  - get\_interacting\_genes, 19
  - get\_pairwise\_interacting\_genes, 21
- \* **internal**
  - .get\_BTME\_features, 5
  - .get\_cogaps\_features, 5
  - .get\_csv\_features, 6
  - .get\_seurat\_features, 6
  - .infer\_method, 6
  - .read\_format, 7
  - .calc\_IM\_scores, 3
  - .calc\_threshold, 4
  - .find\_genes\_of\_interest, 4
  - .get\_BTME\_features, 5
  - .get\_cogaps\_features, 5
  - .get\_csv\_features, 6
  - .get\_seurat\_features, 6
  - .infer\_method, 6
  - .pick\_image, 7
  - .read\_format, 7
  - .row\_t\_test, 8
- calculate\_gene\_scores\_directed, 8
- calculate\_gene\_set\_score, 9
- calculate\_gene\_set\_specificity, 10
- calculate\_influence, 11
- calculate\_lr\_scores, 11
- calculate\_overlap\_directed, 12
- calculate\_overlap\_undirected, 13
- calculate\_thresholds, 14
- curated\_genes, 15
- find\_all\_hotspots, 15
- find\_hotspots\_gmm, 16
- find\_pattern\_hotspots, 16, 20, 22
- get\_im\_scores, 18
- get\_interacting\_genes, 17, 19, 22
- get\_pairwise\_interacting\_genes, 17, 20, 21
- get\_spatial\_features, 23
- get\_spatial\_parameters, 24
- get\_spatial\_params\_morans\_i, 26
- load10XCoords, 27
- load10XExpr, 28
- lrdf, 28
- optParams, 29
- plot\_cell\_interaction\_circos, 29
- plot\_im\_scores, 32
- plot\_overlap\_scores, 33
- plot\_source\_to\_target\_circos, 34
- plot\_spatial\_data\_over\_image, 37
- plot\_target\_from\_sources\_circos, 38