# Package 'CatsCradle'

November 11, 2026

**Title** This package provides methods for analysing spatial transcriptomics data and for discovering gene clusters

**Version** 1.6.1

**Description** This package addresses two broad areas. It allows for in-depth analysis of spatial transcriptomic data by identifying tissue neighbourhoods. These are contiguous regions of tissue surrounding individual cells. 'CatsCradle' allows for the categorisation of neighbourhoods by the cell types contained in them and the genes expressed in them. In particular, it produces Seurat objects whose individual elements are neighbourhoods rather than cells. In addition, it enables the categorisation and annotation of genes by producing Seurat objects whose elements are genes.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**Imports** Seurat (>= 5.0.1), ggplot2, networkD3, stringr, pracma, reshape2, rdist, igraph, geometry, Rfast, data.table, abind, pheatmap, EBImage, S4Vectors, SeuratObject, SingleCellExperiment, SpatialExperiment, Matrix, methods, SummarizedExperiment, msigdbr

**Suggests** fossil, interp, knitr, BiocStyle, tictoc

**Depends** R (>= 4.4.0)

**LazyData** false

**VignetteBuilder** knitr

**BugReports** https://github.com/AnnaLaddach/CatsCradle/issues

**URL** https://github.com/AnnaLaddach/CatsCradle

**biocViews** BiologicalQuestion, StatisticalMethod, GeneExpression, SingleCell, Transcriptomics, Spatial

**NeedsCompilation** no

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

**git_branch** RELEASE_3_23

**git_last_commit** c07e3c1

**git_last_commit_date** 2026-07-02

**Repository** Bioconductor 3.23

**Date/Publication** 2026-11-11

**Author** Anna Laddach [aut] (ORCID: <https://orcid.org/0000-0001-5552-6534>), Michael Shapiro [aut, cre] (ORCID: <https://orcid.org/0000-0002-2769-9320>)

**Maintainer** Michael Shapiro <michael.shapiro@crick.ac.uk>

# Contents

aggregateFeatureMatrix . . . 4
aggregateGeneExpression . . . 5
annotateGeneAsVector . . . 6
annotateGenesByGeneSet . . . 6
annotateLRInteractionCounts . . . 7
cellTypesPerCellTypeGraphFromCellMatrix . . . 8
cellTypesPerCellTypeGraphFromNbhdMatrix . . . 9
collapseExtendedNBHDs . . . 10
combinatorialSpheres . . . 11
computeCellTypesPerCellTypeMatrix . . . 12
computeEdgeGraph . . . 12
computeEdgeObject . . . 13
computeGraphEmbedding . . . 14
computeMoransI . . . 15
computeNBHDByCTMatrix . . . 15
computeNBHDVsCTObject . . . 16
computeNeighbourEnrichment . . . 17
computeNeighboursDelaunay . . . 18
computeNeighboursEuclidean . . . 19
convertToLong . . . 19
countLRInteractionsPerCell . . . 20
cullEdges . . . 21
desymmetriseNN . . . 21
directedHausdorfDistance . . . 22
edgeCutoffsByClustering . . . 23
edgeCutoffsByPercentile . . . 24
edgeCutoffsByWatershed . . . 24
edgeCutoffsByZScore . . . 25
edgeLengthPlot . . . 26
edgeLengthsAndCellTypePairs . . . 27
exampleObjects . . . 27
exSeuratObj . . . 28
formatData . . . 28
geneSetsVsGeneClustersPValueMatrix . . . 29
getAverageExpressionDF . . . 30

getAverageExpressionMatrix . . . 30
getBinarisedMatrix . . . 31
getClusterOrder . . . 32
getExtendedNBHDs . . . 32
getFeatureZScores . . . 33
getGeneClusterAveragesPerCell . . . 34
getGeneNeighbors . . . 35
getInteractionsOnEdges . . . 35
getLigandReceptorNetwork . . . 36
getLigandReceptorPairsInPanel . . . 37
getNearbyGenes . . . 37
getNearestNeighbourLists . . . 38
getObjectSubsetClusteringPValue . . . 39
getObjectSubsetClusteringStatistics . . . 40
getSubsetComponents . . . 41
humanLRN . . . 42
ligandReceptorResults . . . 42
make.getExample . . . 43
makeLRInteractionHeatmap . . . 44
makeSummedLRInteractionHeatmap . . . 45
meanGeneClusterOnCellUMAP . . . 46
meanZPerCluster . . . 46
meanZPerClusterOnUMAP . . . 47
medianComplementDistance . . . 48
medianComplementPValue . . . 49
moransI . . . 50
moransILigandReceptor . . . 50
mouseLRN . . . 51
nbhdsAsEdgesToNbhdsAsList . . . 51
neighbourhoodDiameter . . . 52
orderGeneSetPValues . . . 53
performLigandReceptorAnalysis . . . 53
performLigandReceptorAnalysisAnalytical . . . 56
performLigandReceptorAnalysisPermutation . . . 57
permuteColumns . . . 59
permuteMatrix . . . 59
plotLRDotplot . . . 60
predictAnnotation . . . 61
predictAnnotationAllGenes . . . 62
predictGeneAnnotationImpl . . . 63
randomiseGraph . . . 64
randomiseNodeIndices . . . 65
readGmt . . . 65
runGeometricClusteringTrials . . . 66
runMoransI . . . 67
sankeyFromMatrix . . . 68
seuratCells . . . 69
seuratGenes . . . 69

smallXenium . . . . . 70
stripGeneSet . . . . . 70
symmetriseNN . . . . . 71
symmetryCheckNN . . . . . 71
tagRowAndColNames . . . . . 72
transposeObject . . . . . 72
xeniumCells . . . . . 73

**Index** **74**

---

`aggregateFeatureMatrix`

*This function takes a matrix where rows are features and columns are cells, and a neighbourhood list, and creates an matrix where columns are the neighbourhoods, the rows are are the features and the values are aggregated expression values for cells in each neighbourhood.*

---

## Description

This function takes a matrix where rows are features and columns are cells, and a neighbourhood list, and creates an matrix where columns are the neighbourhoods, the rows are are the features and the values are aggregated expression values for cells in each neighbourhood.

## Usage

```
aggregateFeatureMatrix(M, nbhdList, aggregateFunction)
```

## Arguments

| | |
|---|---|
| `M` | • a matrix where column names are cells and row names are features. |
| `nbhdList` | • a named list with memberships of the neighbourhoods of cells |
| `aggregateFunction` | • a function to aggregate expression (e.g. rowSums, rowMeans) |

## Value

a matrix giving aggregated gene expression for a cell's neighbourhood.

aggregateGeneExpression

*This function takes a Seurat object and a list of neighbourhoods and creates a Seurat object where the columns are the neighbourhoods, the rows are are the genes and the values are gene expression totals for the cells in each neighbourhood*

## Description

This function takes a Seurat object and a list of neighbourhoods and creates a Seurat object where the columns are the neighbourhoods, the rows are are the genes and the values are gene expression totals for the cells in each neighbourhood

## Usage

```
aggregateGeneExpression(
  f,
  neighbourhoods,
  self = FALSE,
  verbose = TRUE,
  returnType = "Seurat"
)
```

## Arguments

| | |
|---|---|
| f | • a Seurat object with layer counts or a SingleCellExperiment to be turned into a Seurat object |
| neighbourhoods | • Neighbourhoods as given by a collapsed expanded edge graph, as produced by collapseNeighbourhoods. In particular, each cell should appear as nodeA. |
| self | • include cell in its neighbourhood, defaults to FALSE |
| verbose | • used to control trace, defaults to TRUE |
| returnType | • Will return a SingleCellExperiment if this is either of SCE, SingleCellExperiment or their lower-case equivalents. Otherwise, returns a Seurat object or SingleCellExperiment, depending on the parameter returnType. |

## Value

a Seurat object giving total gene expression in each neighbourhood or SingleCellExperiment

## Examples

```
getExample = make.getExample()
smallXenium = getExample('smallXenium',toy=TRUE)
extendedNeighbours = getExample('extendedNeighbours',toy=TRUE)
agg = aggregateGeneExpression(smallXenium,extendedNeighbours,verbose=FALSE)
```

`annotateGeneAsVector` *This function returns a numeric indicating which gene sets it does and does not belong to. This vector can be normalised to account for the sizes of the sets.*

## Description

This function returns a numeric indicating which gene sets it does and does not belong to. This vector can be normalised to account for the sizes of the sets.

## Usage

```
annotateGeneAsVector(gene, geneSets, normalise = FALSE)
```

## Arguments

| | |
|---|---|
| `gene` | • the gene to annotate |
| `geneSets` | • a list of gene sets |
| `normalise` | • whether to normalise by set size |

## Value

a numeric

## Examples

```
hallmark = make.getExample()('hallmark')
Myc = annotateGeneAsVector('Myc',hallmark)
MycNormalised = annotateGeneAsVector('Myc',hallmark,TRUE)
```

`annotateGenesByGeneSet` *This function annotates genes with terms*

## Description

This essentially inverts a list of gene sets. It takes a list (e.g., Hallmark or GO) where each list item is a name of a gene set and gives the genes in that set and returns a list where each item is a gene and gives the gene sets that gene is in.

## Usage

```
annotateGenesByGeneSet(geneSets)
```

### Arguments

| | |
|---|---|
| geneSets | • a list of gene sets, e.g., as produced by readGmt |

### Value

- A list where names are genes and values are lists of terms

### Examples

```
hallmark = make.getExample()('hallmark')
annotatedGenes = annotateGenesByGeneSet(hallmark)
```

---

## annotateLRInteractionCounts

*This takes a data frame of interaction counts as found by countLRInteractionsPerCell(), the underlying Seurat object and the neighbourhood Seurat object and annotates the counts with the cell type and the neighbourhood type corresponding to the cells of the interaction counts.*

---

### Description

This takes a data frame of interaction counts as found by countLRInteractionsPerCell(), the underlying Seurat object and the neighbourhood Seurat object and annotates the counts with the cell type and the neighbourhood type corresponding to the cells of the interaction counts.

### Usage

```
annotateLRInteractionCounts(interactionCounts, obj, nbhdObj)
```

### Arguments

| | |
|---|---|
| interactionCounts | • as found by countLRInteractionsPerCell() |
| obj | • a Seurat object, or SingleCellExperiment to be turned into a Seurat object |
| nbhdObj | • a neighbourhood x cell type Seurat object or a SingleCellExperiment to be turned into a Seurat object |

### Value

This returns the interaction counts annotated with the cell type and neighbourhood type of each cell.

cellTypesPerCellTypeGraphFromCellMatrix

*This function converts a matrix as found by cellTypesPerCellTypeMatrix into a directed igraph whose vertices correspond to seurat_clusters and whose edge correspond to occupancy fraction.*

## Description

This function converts a matrix as found by cellTypesPerCellTypeMatrix into a directed igraph whose vertices correspond to seurat_clusters and whose edge correspond to occupancy fraction.

## Usage

```
cellTypesPerCellTypeGraphFromCellMatrix(
  M,
  colours = NULL,
  selfEdges = FALSE,
  minWeight = 0,
  edgeWeighting = 20,
  edgeCurved = 0.2,
  arrowSize = 4,
  arrowWidth = 4,
  plotGraph = TRUE
)
```

## Arguments

| | |
|---|---|
| M | • a matrix as found by cellTypesPerCellTypeMatrix. Note, however, that this matrix may need to be reduced to a square matrix as the matrix produced from a subset object may be missing certain cell types as rows. |
| colours | • a named vector of colours used to colour the vertices of the graph. The names are the seurat_clusters as character strings. |
| selfEdges | • a logical which determines whether to include self edges. Defaults to FALSE |
| minWeight | • Allows one to exclude edges of low weight. Defaults to 0, thus including all edges. |
| edgeWeighting | • a parameter used to thicken the edges in the display. Defaults to 20. |
| edgeCurved | • a parameter to set curvature of the edges. Defaults to 0.2 |
| arrowSize | • a parameter to set arrow size. Defaults to 4. |
| arrowWidth | • a parameter to set arrow width. Defaults to 4. |
| plotGraph | • a logical which determines whether to plot the graph. Defaults to TRUE. |

## Value

This returns a directed igraph whose vertices are the cell types and whose arrows indicate "ownership" of cells of the target type by neighbourhoods of cells of the source type. Layout is done witht the FR algorithm and coordinates are found in the coords attribute of G. If colours were supplied these are found in color attribute of V(G). Edge weights and widths are found in the weight and width attributes of E(G).

## Examples

```
getExample = make.getExample()
cellTypesPerCellTypeMatrix = getExample('cellTypesPerCellTypeMatrix')
colours = getExample('colours')
G = cellTypesPerCellTypeGraphFromCellMatrix(cellTypesPerCellTypeMatrix,
                                    minWeight = 0.05, colours = colours)
```

---

`cellTypesPerCellTypeGraphFromNbhdMatrix`

*This function takes a neighbourhood-by-cell type matrix and produces a directed igraph showing the fractions of cells of each type in the neighbourhoods around cells of each type.*

---

## Description

This function takes a neighbourhood-by-cell type matrix and produces a directed igraph showing the fractions of cells of each type in the neighbourhoods around cells of each type.

## Usage

```
cellTypesPerCellTypeGraphFromNbhdMatrix(
  nbhdByCellType,
  clusters,
  colours = NULL,
  selfEdges = FALSE,
  minWeight = 0,
  edgeWeighting = 20,
  edgeCurved = 0.2,
  arrowSize = 4,
  arrowWidth = 4,
  plotGraph = TRUE
)
```

## Arguments

| | |
|---|---|
| `nbhdByCellType` | • A matrix whose rows are neighbourhoods each denoted by the cell at their center, whose columns are cell types, and whose entries are counts. |
| `clusters` | • a named vector whose names are the cells and whose entries are their seurat_clusters. |

| | |
|---|---|
| colours | • a named vector of colours used to colour the vertices of the graph. The names are the seurat_clusters as character strings. |
| selfEdges | • a logical which determines whether to include self edges. Defaults to FALSE |
| minWeight | • Allows one to exclude edges of low weight. Defaults to 0, thus including all edges. |
| edgeWeighting | • a parameter used to thicken the edges in the display. Defaults to 20. |
| edgeCurved | • a parameter to set curvature of the edges. Defaults to 0.2 |
| arrowSize | • a parameter to set arrow size. Defaults to 4. |
| arrowWidth | • a parameter to set arrow width. Defaults to 4. |
| plotGraph | • a logical which determines whether to plot the graph. Defaults to TRUE. |

## Value

This returns a directed igraph whose vertices are the cell types and whose arrows indicate "ownership" of cells of the target type by neighbourhoods of cells of the source type. Layout is done witht the FR algorithm and coordinates are found in the coords attribute of G. If colours were supplied these are found in the color attribute of V(G). Edge weights and widths are found in the weight and width attributes of E(G).

---

`collapseExtendedNBHDs` *This function takes an expanded neighbourhood list and collapses it to a nearest neighbourhood graph where all neighbours of degree <= n in the original graph are considered first neighbours.*

---

## Description

This function takes an expanded neighbourhood list and collapses it to a nearest neighbourhood graph where all neighbours of degree <= n in the original graph are considered first neighbours.

## Usage

```
collapseExtendedNBHDs(
  extendedNeighboursList,
  n = length(extendedNeighboursList)
)
```

## Arguments

| | |
|---|---|
| extendedNeighboursList | • the results of getExtendedNBHDs() |
| n | • the maximum degree to connect neighbours. Defaults to the maximum degree neighbourhoods were expanded to in the results of getExtendedNBHDs(). |

### Value

a graph in neighbour format, i.e., a data frame with columns nodeA and nodeB, where nodes that were originally of degree <= n are connected.

### Examples

```
extendedNeighboursList = make.getExample()('extendedNeighboursList',toy=TRUE)
extendedNeighbours = collapseExtendedNBHDs(extendedNeighboursList, 4)
```

---

## combinatorialSpheres *Discovers the combinatorial ball of a given radius around a fixed set of genes in the nearest neighbor graph of a Seurat object.*

---

### Description

Discovers the combinatorial ball of a given radius around a fixed set of genes in the nearest neighbor graph of a Seurat object.

### Usage

```
combinatorialSpheres(NN, origin, radius)
```

### Arguments

| | |
|---|---|
| NN | • a nearest neighbors graph |
| origin | • a gene or list of genes |
| radius | • the radius of the combinatorial ball to be found. |

### Value

This returns a data frame whose columns are the gene name, the radius from the origin at which it is found

### Examples

```
getExample = make.getExample()
NN = getExample('NN',toy=TRUE)
STranspose = getExample('STranspose',toy=TRUE)
spheres = combinatorialSpheres(NN,'Ccl6',3)
hallmark = getExample('hallmark')
geneSet = intersect(hallmark[["HALLMARK_TNFA_SIGNALING_VIA_NFKB"]],colnames(STranspose))
sphereAroundSet = combinatorialSpheres(NN,geneSet,1)
```

computeCellTypesPerCellTypeMatrix *For each cell type, this function looks at the neighbourhoods around cells of that type and discovers the fractions (or numbers if normalise = F) of those cells of each type.*

## Description

For each cell type, this function looks at the neighbourhoods around cells of that type and discovers the fractions (or numbers if normalise = F) of those cells of each type.

## Usage

```
computeCellTypesPerCellTypeMatrix(nbhdByCellType, cellTypes, normalise = TRUE)
```

## Arguments

| | |
|---|---|
| nbhdByCellType | • A matrix whose rows are neighbourhoods each denoted by the cell at their center, whose columns are cell types, and whose entries are counts. |
| cellTypes | • named vector of cell types where names are each cell and cell types are a factor |
| normalise | • boolean, defaults to TRUE |

## Value

A square matrix whose rownames and colnames are the seurat_clusters as character strings. Each row corresponds to neighbourhoods around all cells of that type and the entries give the fractions of those neighbourhoods occupied by cells of each type.

## Examples

```
getExample = make.getExample()
NBHDByCTMatrix = getExample('NBHDByCTMatrix')
clusters = getExample('clusters')
cellTypesPerCellType = computeCellTypesPerCellTypeMatrix(NBHDByCTMatrix,clusters)
```

computeEdgeGraph *This function takes a spatial graph and computes a new spatial graph where edges become nodes and A-B edges (in the original graph) become connected to all A- edges and all B- edges.*

## Description

This function takes a spatial graph and computes a new spatial graph where edges become nodes and A-B edges (in the original graph) become connected to all A- edges and all B- edges.

### Usage

```
computeEdgeGraph(spatialGraph, selfEdges = FALSE)
```

### Arguments

| | |
|---|---|
| spatialGraph | • a data frame of neighbouring edge pairs. |
| selfEdges | • a logical determining whether to include self edges. Defaults to False. |

### Value

a graph in neighbour format where edges in the original graph become nodes and A-B edges (in the original graph) become connected to all A- edges and all B- edges.

### Examples

```
delaunayNeighbours = make.getExample()('delaunayNeighbours')
edgeNeighbours = computeEdgeGraph(delaunayNeighbours)
```

---

## computeEdgeObject

*This function takes interactionResults and creates a seurat object where each point represents an edge between cells, and spatial coordinates are the centroids of edges between cells. The "expression matrix" is the binarised presence/absence of an interaction (ligand receptor pair) on an edge.*

---

### Description

This function takes interactionResults and creates a seurat object where each point represents an edge between cells, and spatial coordinates are the centroids of edges between cells. The "expression matrix" is the binarised presence/absence of an interaction (ligand receptor pair) on an edge.

### Usage

```
computeEdgeObject(
  ligandReceptorResults,
  centroids,
  npcs = 10,
  returnType = "Seurat"
)
```

### Arguments

| | |
|---|---|
| ligandReceptorResults | • as returned by performLigandReceptorResultsAnalysis() |
| centroids | • a dataframe containing centroids where rownames are cellnames and the first two columns contain x and y coordinates respectively. |
| npcs | • number of pcs used for PCA, defaults to 10 |

returnType — Determines whether to return a Seurat object or a SpatialExperiment. Will do the later if this is set to either SCE, SingleCellExperiment or lower case versions of either.

## Value

This returns a seurat object where each point represents an edge between cells, and spatial coordinates are the centroids of edges between cells. The "expression matrix" is the binarised presence/absence of an interaction (ligand receptor pair) on an edge. Depending on the parameter returnType, this can alternatively be returned as a SpatialExperiment.

## Examples

```
getExample = make.getExample()
centroids = getExample('centroids')
ligandReceptorResults = getExample('ligandReceptorResults')
edgeSeurat = computeEdgeObject(ligandReceptorResults, centroids)
```

---

# computeGraphEmbedding *This function adds a force directed graph embedding to a seurat object*

---

## Description

This function adds a force directed graph embedding to a seurat object

## Usage

```
computeGraphEmbedding(
  seuratObj,
  graph = defaultGraph(seuratObj),
  returnType = "Seurat"
)
```

## Arguments

| | |
|---|---|
| seuratObj | • a seurat object of SingleCellExperiment to be turned into a Seurat object |
| graph | • which graph to extract. Defaults to paste0(f@active.assay,'_snn') |
| returnType | • Will return a SingleCellExperiment if this is either of SCE, SingleCellExperiment or their lower-case equivalents. Otherwise, returns a Seurat object |

## Value

a seurat object with a "graph" dimensionality reduction. Can also be a SingleCellExperiment depending on parameter returnType.

## Examples

```
NBHDByCTSeurat = make.getExample()('NBHDByCTSeurat',toy=TRUE)
objWithEmbedding = computeGraphEmbedding(NBHDByCTSeurat)
```

`computeMoransI` *This function takes a matrix where rows are features and columns are cells, and a neighbourhood list, and computes Moran's I.*

## Description

This function takes a matrix where rows are features and columns are cells, and a neighbourhood list, and computes Moran's I.

## Usage

```
computeMoransI(M, nbhdList)
```

## Arguments

| | |
|---|---|
| `M` | • a matrix where column names are cells and row names are features. |
| `nbhdList` | • a named list with memberships of the neighbourhoods of cells |

## Value

a matrix giving aggregated gene expression for a cell's neighbourhood.

`computeNBHDByCTMatrix` *This function computes a matrix where neighbourhoods are rows and cell types are columns. The values in the matrix indicate the number of cells of a given type within a neighbourhood.*

## Description

This function computes a matrix where neighbourhoods are rows and cell types are columns. The values in the matrix indicate the number of cells of a given type within a neighbourhood.

## Usage

```
computeNBHDByCTMatrix(spatialGraph, cellTypes)
```

## Arguments

| | |
|---|---|
| `spatialGraph` | • a spatial graph in neighbour list format. |
| `cellTypes` | • named vector of cell types where names are each cell and cell types are a factor |

## Value

a matrix of neighbourhoods by cell types

## Examples

```
getExample = make.getExample()
clusters = getExample('clusters')
delaunayNeighbours = getExample('delaunayNeighbours')
NBHDByCTMatrix = computeNBHDByCTMatrix(delaunayNeighbours,clusters)
```

---

`computeNBHDVsCTObject` *This function creates a seurat object using a neighbourhood by cell type matrix*

---

## Description

This function creates a seurat object using a neighbourhood by cell type matrix

## Usage

```
computeNBHDVsCTObject(
  dataMatrix,
  resolution = 0.1,
  npcs = 10,
  n.neighbors = 30L,
  transpose = FALSE,
  verbose = TRUE,
  returnType = "Seurat"
)
```

## Arguments

| | |
|---|---|
| `dataMatrix` | • a matrix of neighbourhoods by cell types or its transpose. |
| `resolution` | • resolution for clustering (default 0.1). |
| `npcs` | • number of pcs used for PCA, defaults to 10. |
| `n.neighbors` | • number of neighbors used by UMAP, defaults to 30. |
| `transpose` | • defaults to FALSE. |
| `verbose` | • defaults to TRUE, used to limit trace if FALSE |
| `returnType` | • Will return a SingleCellExperiment if this is either of SCE, SingleCellExperiment or their lower-case equivalents. Otherwise, returns a Seurat object |

## Value

a seurat object based on a neighbourhood by cell type matrix or its transpose, containing clusters and UMAP. This can also be a SingleCellExperiment depending on the parameter returnType.

## Examples

```
NBHDByCTMatrix = make.getExample()('NBHDByCTMatrix',toy=TRUE)
NBHDByCTSeurat = computeNBHDVsCTObject(NBHDByCTMatrix)
NBHDByCTSingleCell_sce = computeNBHDVsCTObject(NBHDByCTMatrix,returnType='SCE')
```

computeNeighbourEnrichment

*This function calculates P values for whether cell types are more frequently neighbours than expected by chance. By default it calculates P values analytically using a hypergeometric test on the edges, where the arguments to the R phyper function are as follows: q = number of edges between cell type A and B m = number of edges between cell type B and any other cell type n = the number of edges between any cell type apart from cell type B k = number of edges between cell type B and any other cell type The purist may object to the use of the hypergeometric test here. We may think of "edges out of a cell of type A" as being the random draw balls (here, edges) from the urn and "edges out of cells of type B" as being success. However, all edges out of a given cell of type A are in this "random draw". Clearly the edges in this draw are not independent. However, empirically we find that p-values computed using this method correspond very closely to those computed using permutation while the computation time is orders of magnitude faster.*

## Description

For legacy purposes, and for user flexibility, it allows for the calculation of P values by comparison to randomised graphs. It offers two distinct randomisations. One is by permuting the cell types on the neighbour (e.g., delaunay) graph. The other is by comparison to randomised neighbour graphs where edges are randomised but the degree of each node is preserved.

## Usage

```
computeNeighbourEnrichment(
  spatialGraph,
  cellTypes,
  method = "analytical",
  nSim = 1000,
  maxTries = 1000,
  randomiseBy = "cells",
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| spatialGraph | • a spatial graph in neighbour list format. |
| cellTypes | • named vector of cell types where names are each cell and cell types are a factor. |
| method | • method for computing p-values. Defaults to "analytical", in which case an edge-based hypergeometric test is performed. If "permutation" is selected p-values are calculated by comparison to randomised graphs (note this is slower than the analytical approach). |

| | |
|---|---|
| nSim | • the number of randomised graphs to create for pvalue calculation, if the method is set to permutation. |
| maxTries | • the maximum number of tries to remove self edges during graph randomisation. If self edges are remeining this will be reported. |
| randomiseBy | • This takes either the value 'cells' (the default) or 'graph'. In the former case randomisation is carried out by permuting the cell types on the existing graph. In the latter case, the graph is permuted using the function randomiseGraph() which is a heuristic algorithm to preserve the distribution of vertex degrees. |
| verbose | • whether to print trace. Defaults to TRUE |

## Value

A square matrix containing upper tail p values describing whether two cell types are more frequently found together than expected by chance.

## Examples

```
getExample = make.getExample()
delaunayNeighbours = getExample('delaunayNeighbours')
clusters = getExample('clusters')
cellTypesPerCellTypePValues = computeNeighbourEnrichment(delaunayNeighbours,
                                        clusters, verbose = FALSE)
```

---

# computeNeighboursDelaunay

*This function computes a spatial graph where neighbors are identified based on Delaunay triangulation.*

---

## Description

This function computes a spatial graph where neighbors are identified based on Delaunay triangulation.

## Usage

```
computeNeighboursDelaunay(centroids)
```

## Arguments

| | |
|---|---|
| centroids | • a dataframe containing centroids where rownames are cellnames and the first two columns contain x and y coordinates respectively. |

## Value

a graph in neighbour format, i.e., a data frame with columns nodeA and nodeB.

### Examples

```
centroids = make.getExample()('centroids')
delaunayNeighbours = computeNeighboursDelaunay(centroids)
```

---

`computeNeighboursEuclidean` *This function computes a spatial graph where neighbors are identified based on euclidean distance and a user defined threshold.*

---

### Description

This function computes a spatial graph where neighbors are identified based on euclidean distance and a user defined threshold.

### Usage

```
computeNeighboursEuclidean(centroids, threshold)
```

### Arguments

| | |
|---|---|
| `centroids` | • a dataframe containing centroids where rownames are cellnames and columns contain x and y coordinates respectively. |
| `threshold` | • a distance cut off to compute neighbours. |

### Value

a graph in neighbour format, i.e., a data frame with columns nodeA and nodeB.

### Examples

```
centroids = make.getExample()('centroids')
euclideanNeighbours = computeNeighboursEuclidean(centroids,20)
```

---

`convertToLong` *This is a utility function for converting ligandReceptor cluster-level results to long format and calculates adjusted p-values.*

---

### Description

This is a utility function for converting ligandReceptor cluster-level results to long format and calculates adjusted p-values.

### Usage

```
convertToLong(ligandReceptorResults)
```

### Arguments

`ligandReceptorResults`
- ligandReceptorReults calculated using performLigandReceptorAnalysis()

### Value

ligand receptor results in long format

---

`countLRInteractionsPerCell`

*This function takes a listing of the neighbouring cells together with the presence or absence of each ligand-receptor pair on each edge and produces a count showing for each cell, how many neighbours it has with that interaction either as source or as target*

---

### Description

This function takes a listing of the neighbouring cells together with the presence or absence of each ligand-receptor pair on each edge and produces a count showing for each cell, how many neighbours it has with that interaction either as source or as target

### Usage

```
countLRInteractionsPerCell(edges, sourceOrTarget)
```

### Arguments

| | |
|---|---|
| `edges` | • A data frame of neighbouring cells together with their interactions as produced by getInteractionsOnEdges() |
| `sourceOrTarget` | • a character, either 'source' or 'target' telling which direction of interaction to count |

### Value

This returns a data frame with one row for each cell and a column giving the name of that cell and the other columns giving the counts of interactions that it has with its neighbours.

`cullEdges` *This subsets edges by our chosen critera*

## Description

This subsets edges by our chosen critera

## Usage

```
cullEdges(annEdges, cutoffSpec)
```

## Arguments

| | |
|---|---|
| `annEdges` | • a data frame with columns nodeA, nodeB, length and cellTypePair as produced by edgeLengthsAndCellTypePairs. |
| `cutoffSpec` | • This can be either a numeric value which will be applied across all edges as an upper limit or a data frame with columns cellTypePair and cutoff as produced by any of the edgeCutoffsBy functions |

## Value

This returns a subset of the annotated edges

## Examples

```
getExample = make.getExample()
centroids = getExample('centroids')
clusters = getExample('clusters')
delaunayNeighbours = getExample('delaunayNeighbours')
annEdges =
    edgeLengthsAndCellTypePairs(delaunayNeighbours,clusters,centroids)
tolerance = 5
nbins = 15
cutoffDFWater = edgeCutoffsByWatershed(annEdges,
                                       tolerance=tolerance,
                                       nbins=nbins)
culledEdges = cullEdges(annEdges,cutoffDFWater)
```

`desymmetriseNN` *This function takes the data frame of neighbor genes and reduces it so that each undirected edge is represented by only one directed edge. This ensures that randomisation does not magically split undirected edges into two edges.*

### Description

This function takes the data frame of neighbor genes and reduces it so that each undirected edge is represented by only one directed edge. This ensures that randomisation does not magically split undirected edges into two edges.

### Usage

```
desymmetriseNN(NN)
```

### Arguments

| | |
|---|---|
| NN | • a dataframe containing the neighborlist |

### Value

- a neighborListDF with only one directed edge per undirected edge.

### Examples

```
NN = make.getExample()('NN',toy=TRUE)
print(dim(NN))
NNN = desymmetriseNN(NN)
print(dim(NNN))
```

---

## directedHausdorfDistance

*This finds the directed Hausdorf distance from A to B*

---

### Description

This finds the directed Hausdorf distance from A to B

### Usage

```
directedHausdorfDistance(A, B)
```

### Arguments

| | |
|---|---|
| A | • an m x d matrix representing m points in dimension d |
| B | • an n x d matrix representing n points in dimension d |

### Value

This returns the distance of the furthest point in A from its nearest point in B.

## Examples

```
A = matrix(seq_len(8),ncol=2)
B = matrix(seq(from=3,to=16),ncol=2)
d_hausdorf = directedHausdorfDistance(A,B)
```

---

edgeCutoffsByClustering

*This finds proposed cutoffs for edge lengths by clustering the lengths of the edges for each cell type pair using k-means clustering with k = 2*

---

## Description

This finds proposed cutoffs for edge lengths by clustering the lengths of the edges for each cell type pair using k-means clustering with k = 2

## Usage

```
edgeCutoffsByClustering(annEdges)
```

## Arguments

| | |
|---|---|
| annEdges | • a data frame with columns nodeA, nodeB, length and cellTypePair as produced by edgeLengthsAndCellTypePairs. |

## Value

This returns a data frame with columns cellTypePair and cutoff.

## Examples

```
getExample = make.getExample()
centroids = getExample('centroids')
clusters = getExample('clusters')
delaunayNeighbours = getExample('delaunayNeighbours')
annEdges =
    edgeLengthsAndCellTypePairs(delaunayNeighbours,clusters,centroids)
cutoffDF = edgeCutoffsByClustering(annEdges)
```

edgeCutoffsByPercentile

*This finds edge cutoffs by percentile*

## Description

This finds edge cutoffs by percentile

## Usage

```
edgeCutoffsByPercentile(annEdges, percentileCutoff)
```

## Arguments

| | |
|---|---|
| annEdges | • a data frame with columns nodeA, nodeB, length and cellTypePair as produced by edgeLengthsAndCellTypePairs. |
| percentileCutoff | • a numeric |

## Value

This returns a data frame with columns cellTypePair and cutoff.

## Examples

```
getExample = make.getExample()
centroids = getExample('centroids')
clusters = getExample('clusters')
delaunayNeighbours = getExample('delaunayNeighbours')
annEdges =
    edgeLengthsAndCellTypePairs(delaunayNeighbours,clusters,centroids)
cutoffDF = edgeCutoffsByPercentile(annEdges,percentileCutoff=95)
```

edgeCutoffsByWatershed

*This finds proposed cutoffs for edge lengths by computing the histogram of edge lengths for each cell type pair and then using the watershed algorithm to find the hump of the histogram containing the median.*

## Description

This finds proposed cutoffs for edge lengths by computing the histogram of edge lengths for each cell type pair and then using the watershed algorithm to find the hump of the histogram containing the median.

### Usage

```
edgeCutoffsByWatershed(annEdges, nbins = 15, tolerance = 10)
```

### Arguments

| | |
|---|---|
| annEdges | • a data frame with columns nodeA, nodeB, length and cellTypePair as produced by edgeLengthsAndCellTypePairs. |
| nbins | • the number of bins for the histogram |
| tolerance | • the tolerance parameter for the watershed algorithm. |

### Value

This returns a data frame with columns cellTypePair and cutoff.

### Examples

```
getExample = make.getExample()
centroids = getExample('centroids')
clusters = getExample('clusters')
delaunayNeighbours = getExample('delaunayNeighbours')
annEdges =
    edgeLengthsAndCellTypePairs(delaunayNeighbours,clusters,centroids)
cutoffDF = edgeCutoffsByWatershed(annEdges)
```

---

## edgeCutoffsByZScore *This finds edge cutoffs by z-score*

---

### Description

This finds edge cutoffs by z-score

### Usage

```
edgeCutoffsByZScore(annEdges, zCutoff)
```

### Arguments

| | |
|---|---|
| annEdges | • a data frame with columns nodeA, nodeB, length and cellTypePair as produced by edgeLengthsAndCellTypePairs. |
| zCutoff | • a numeric |

### Value

This returns a data frame with columns cellTypePair and cutoff.

### Examples

```
getExample = make.getExample()
centroids = getExample('centroids')
clusters = getExample('clusters')
delaunayNeighbours = getExample('delaunayNeighbours')
annEdges =
    edgeLengthsAndCellTypePairs(delaunayNeighbours,clusters,centroids)
cutoffDF = edgeCutoffsByZScore(annEdges,zCutoff=1.5)
```

---

## edgeLengthPlot *edgeLengthPlot*

---

### Description

This plots histograms of the edge lengths broken out by the cell types of the cells they connect. It optionally plots a cutoff for each pair of types.

### Usage

```
edgeLengthPlot(annEdges, cutoffDF, whichPairs, xLim = 100, legend = FALSE)
```

### Arguments

| | |
|---|---|
| annEdges | • A data frame as produced by edgeLengthsAndCellTypePairs |
| cutoffDF | • A data frame with columns cellTypePair and cutoff. This defaults to NULL in which case no cutoffs will be plotted. |
| whichPairs | • Which cellTypePairs to plot. If this is NULL, we plot all pairs. If this is a numeric, we plot only pairs that have at least this many edges. If this is a character vector, we plot the pairs in this list. |
| xLim | • limits the extent of the plots. Defaults to 100. Can be set to NULL. |
| legend | • Show legend, defaults to FALSE |

### Value

This returns a ggplot object

### Examples

```
getExample = make.getExample()
centroids = getExample('centroids')
clusters = getExample('clusters')
delaunayNeighbours = getExample('delaunayNeighbours')
annEdges =
   edgeLengthsAndCellTypePairs(delaunayNeighbours,clusters,centroids)
cutoffDF = edgeCutoffsByPercentile(annEdges,95)
g = edgeLengthPlot(annEdges,cutoffDF,whichPairs=60)
```

## edgeLengthsAndCellTypePairs

*This function annotates edges with their distance and the types of cells they connect*

### Description

This function annotates edges with their distance and the types of cells they connect

### Usage

```
edgeLengthsAndCellTypePairs(edges, clusters, centroids)
```

### Arguments

| | |
|---|---|
| edges | • A data frame with columns nodeA and nodeB giving the cells of each edge |
| clusters | • the clusters of each cell |
| centroids | • the centroids of each cell |

### Value

a data frame giving the edges (as nodeA and nodeB), their lengths and the cell type pair.

### Examples

```
getExample = make.getExample()
centroids = getExample('centroids')
clusters = getExample('clusters')
delaunayNeighbours = getExample('delaunayNeighbours')
annEdges = edgeLengthsAndCellTypePairs(delaunayNeighbours,clusters,centroids)
```

## exampleObjects

*This returns the names of available example objects.*

### Description

This returns the names of available example objects.

### Usage

```
exampleObjects()
```

### Value

A character vector of the names of available example data objects

### Examples

```
availableObjects = exampleObjects()
```

---

`exSeuratObj` *exSeuratObj*

---

### Description

A Seurat object of 2000 genes by 540 cells.

### Usage

```
exSeuratObj
```

### Format

A Seurat object

A Seurat object of cells. It includes a UMAP of the cells and annotated clustering into cell types. It has been severely reduced in size to accommodate Bioconductor size restrictions.

### Source

This is subset from the data associated with https://www.nature.com/articles/s41586-021-04006-z

---

`formatData` *This is a utility function for converting entries in ligandReceptorResults to long format.*

---

### Description

This is a utility function for converting entries in ligandReceptorResults to long format.

### Usage

```
formatData(data, name)
```

### Arguments

| | |
|---|---|
| `data` | • item from ligandReceptorResults |
| `name` | • name to give column of returned data |

### Value

dataframe with item from ligandReceptorResults in long format

geneSetsVsGeneClustersPValueMatrix

*This compares the gene clusters to other gene sets e.g., GO, Hallmark, and determines the p-value for their overlaps when compared to a set of background genes.*

## Description

This compares the gene clusters to other gene sets e.g., GO, Hallmark, and determines the p-value for their overlaps when compared to a set of background genes.

## Usage

```
geneSetsVsGeneClustersPValueMatrix(
  geneSets,
  clusterDF,
  backgroundGenes,
  adjust = FALSE
)
```

## Arguments

| | |
|---|---|
| geneSets | • a named list of gene sets |
| clusterDF | • a data frame giving the cluster membership of each gene with columns gene and geneCluster |
| backgroundGenes | • a character vector of genes |
| adjust | • a logical deciding whether to adjust p values. Defaults to FALSE. |

## Value

a matrix of p-values rows correspond to the gene sets and the columns correspond the the CatsCradle gene clusters

## Examples

```
getExample = make.getExample()
STranspose = getExample('STranspose',toy=TRUE)
clusterDF = data.frame(gene=colnames(STranspose),
                       geneCluster=STranspose$seurat_clusters)
hallmark = getExample('hallmark')
geneSet = intersect(hallmark[["HALLMARK_TNFA_SIGNALING_VIA_NFKB"]],colnames(STranspose))
pvalueMatrix = geneSetsVsGeneClustersPValueMatrix(geneSet,
                                                  clusterDF,
                                                  colnames(STranspose))
```

## getAverageExpressionDF

*This converts an average gene expression matrix to a data frame.*

### Description

This converts an average gene expression matrix to a data frame.

### Usage

```
getAverageExpressionDF(M)
```

### Arguments

| | |
|---|---|
| M | • An average gene expression matrix. |

### Value

A data frame with columns cellCluster, geneCluster and average expression

### Examples

```
getExample = make.getExample()
averageExpMatrix = getExample('averageExpMatrix',toy=TRUE)
averageExpDF = getAverageExpressionDF(averageExpMatrix)
```

## getAverageExpressionMatrix

*This computes average expression of each gene cluster in each cell cluster and returns the result as a matrix*

### Description

This computes average expression of each gene cluster in each cell cluster and returns the result as a matrix

### Usage

```
getAverageExpressionMatrix(
  f,
  fPrime,
  clusteringName = "seurat_clusters",
  layer = "scale.data"
)
```

### Arguments

| | |
|---|---|
| f | • The Seurat object of cells, or SingleCellExperiment to be turned into a Seurat object |
| fPrime | • The Seurat object of genes, or SingleCellExperiment to be turned into a Seurat object |
| clusteringName | In many cases, this will be the cell clustering, i.e., seurat_clusters, which is the default, but for neighbourhood Seurat objects, this can be neighbourhood_clusters. |
| layer | • layer to use for expression values |

### Value

A matrix of the average expression where the rows correspond to cell clusters and the columns correspond to gene clusters.

### Examples

```
getExample = make.getExample()
STranspose = getExample('STranspose',toy=TRUE)
exSeuratObj = getExample('exSeuratObj',toy=TRUE)
M = getAverageExpressionMatrix(exSeuratObj,STranspose,layer='data')
```

---

## getBinarisedMatrix *This functions retrieves an expression matrix from a seurat object or SingleCellExperiment and binarises it.*

---

### Description

This functions retrieves an expression matrix from a seurat object or SingleCellExperiment and binarises it.

### Usage

```
getBinarisedMatrix(obj, cutoff = 0, layer = "counts")
```

### Arguments

| | |
|---|---|
| obj | • a Seurat object or SingleCellExperiment to be turned into a Seurat object |
| cutoff | • a cutoff for binarisation. Defaults to 0. |
| layer | • layer to fetch data from. Defaults to count. |

### Value

A binarised sparse expression matrix where rows are genes and columns are cells.

`getClusterOrder` *This gets the clusters in their cannonical order*

## Description

This deals with skullduggery in which seurat_clusters has been converted from a factor to a character or a numeric.

## Usage

```
getClusterOrder(f)
```

## Arguments

| | |
|---|---|
| `f` | • a Seurat object with meta.data column seurat_clusters or SingleCellExperiment to be turned into a Seurat object |

## Value

A vector of these unique values in order

## Examples

```
STranspose = make.getExample()('STranspose',toy=TRUE)
geneClusters = getClusterOrder(STranspose)
```

`getExtendedNBHDs` *This function takes a nearest neighbour graph and a radius and calculates nth degree neighbour graphs where max(n) == radius*

## Description

This function takes a nearest neighbour graph and a radius and calculates nth degree neighbour graphs where max(n) == radius

## Usage

```
getExtendedNBHDs(spatialGraph, n)
```

## Arguments

| | |
|---|---|
| `spatialGraph` | • a nearest neighbour graph |
| `n` | • the maximum degree to calculate a neighbour graph with edges connecting vertices of degree n for. |

## Value

A named list of neighbour graphs, where each graph contains edges connecting vertices of degree n. Each graph is named according to degree n.

## Examples

```
delaunayNeighbours = make.getExample()('delaunayNeighbours')
extendedNeighboursList = getExtendedNBHDs(delaunayNeighbours, 4)
```

---

# getFeatureZScores *This gets z-scores for the values of features*

---

## Description

This gets z-scores for the values of features

## Usage

```
getFeatureZScores(f, features = rownames(f), layer = "data")
```

## Arguments

| | |
|---|---|
| f | • a Seurat object of cells or SingleCellExperiment to be converted to a Seurat object |
| features | • a set of features to retrieve z-scores for, defaults to rownames(f) |
| layer | • the data layer to retrieve |

## Value

This returns a data frame with a column for each feature and a row for each cell

## Examples

```
getExample = make.getExample()
exSeuratObj = getExample('exSeuratObj',toy=TRUE)
df = getFeatureZScores(exSeuratObj)
```

`getGeneClusterAveragesPerCell` *This produces a matrix giving the average expression of gene clusters in cells. By default, it uses all cells and all gene clusters.*

## Description

This produces a matrix giving the average expression of gene clusters in cells. By default, it uses all cells and all gene clusters.

## Usage

```
getGeneClusterAveragesPerCell(
  f,
  fPrime,
  cells = colnames(f),
  geneClusters = getClusterOrder(fPrime),
  layer = "data"
)
```

## Arguments

| | |
|---|---|
| `f` | • the cell Seurat object or SingleCellExperiment to be turned into a Seurat object |
| `fPrime` | • the genes Seurat object or SingleCellExperiment to be turned into a Seurat object |
| `cells` | • the cells to compute this for |
| `geneClusters` | • the geneClusters to compute average expression for |
| `layer` | • the data layer to use, defaults to 'data' |

## Value

A matrix where the rows correspond to cells, the columns correspond to geneClusters and the entries give average expression for each cluster in each cell

## Examples

```
getExample = make.getExample()
exSeuratObj = getExample('exSeuratObj',toy=TRUE)
STranspose = getExample('STranspose',toy=TRUE)
clusterExpression = getGeneClusterAveragesPerCell(exSeuratObj,STranspose)
```

getGeneNeighbors *This function gets the neighbors of a given gene using either the gene Seurat object or its nearest neighbor graph returned from getNearestNeighbourLists*

## Description

This function gets the neighbors of a given gene using either the gene Seurat object or its nearest neighbor graph returned from getNearestNeighbourLists

## Usage

```
getGeneNeighbors(gene, NN)
```

## Arguments

| | |
|---|---|
| gene | • the gene in question |
| NN | • either the gene Seurat object or its nearest neighbor graph as found by getNearestNeighbourLists. This can also be a SingleCellExperiment which will be converted to a Seurat object |

## Value

the neighboring genes

## Examples

```
library(Seurat)
getExample = make.getExample()
STranspose = getExample('STranspose',toy=TRUE)
NN = getExample('NN',toy=TRUE)
neighbors = getGeneNeighbors("Ccl6",STranspose)
neighborsAgain = getGeneNeighbors("Ccl6",NN)
```

getInteractionsOnEdges *This function takes a binarised expression matrix, a set of ligand receptor pairs and a set of edges denoting neighbouring cells and annotates these with the ligand receptor interactions taking place on those edges in each direction.*

## Description

This function takes a binarised expression matrix, a set of ligand receptor pairs and a set of edges denoting neighbouring cells and annotates these with the ligand receptor interactions taking place on those edges in each direction.

### Usage

```
getInteractionsOnEdges(M, pairDF, spatialGraph)
```

### Arguments

| | |
|---|---|
| M | • a binarised expression matrix where rows are genes and columns are cells. |
| pairDF | • a data frame giving the ligand-receptor pairs |
| spatialGraph | • a data frame of neighbouring cell pairs. Note that each row is a directed edge (A,B) so that this data frame should have both the edge (A,B) and the edge (B,A) |

### Value

This returns a data frame whose first two columns give the neighbouring cells. Each of the remaining columns is a logical corresponding to a ligand-receptor pair telling whether the ligand is expressed in the first cell and the receptor is expressed in the second cell.

---

## getLigandReceptorNetwork

*This function retrieves the Nichenetr ligand- receptor network for mouse or human.*

---

### Description

This function retrieves the Nichenetr ligand- receptor network for mouse or human.

### Usage

```
getLigandReceptorNetwork(species)
```

### Arguments

| | |
|---|---|
| species | • either 'human' or 'mouse' |

### Value

This returns a data frame whose first two columns are from and to, i.e., ligand and receptor. These are derived from the nichenetr ligand receptor networks.

### Examples

```
lrn = getLigandReceptorNetwork('human')
```

`getLigandReceptorPairsInPanel` *This functions takes an Seurat object, its species and a ligand receptor network and subsets the ligand receptor network to those pairs that occur in the panel*

## Description

This functions takes an Seurat object, its species and a ligand receptor network and subsets the ligand receptor network to those pairs that occur in the panel

## Usage

```
getLigandReceptorPairsInPanel(
  obj,
  species,
  lrn = getLigandReceptorNetwork(species)
)
```

## Arguments

| | |
|---|---|
| `obj` | • a Seurat object or SingleCellExperiment to be converted to a Seurat object |
| `species` | • either 'human' or 'mouse' |
| `lrn` | • a ligand-receptor network, i.e., a data frame with columns from and to. By default, it retrieves the nichenetr ligand receptor network |

## Value

This returns a data frame with columns ligand and receptor

## Examples

```
smallXenium = make.getExample()('smallXenium')
lrPairs = getLigandReceptorPairsInPanel(smallXenium, "mouse")
```

`getNearbyGenes` *Nearby genes*

## Description

This finds the genes near a give subset using either a dimensional reduction or the nearest neighbor graph

### Usage

```
getNearbyGenes(
  fPrime,
  geneSet,
  radius,
  metric = "umap",
  numPCs = NULL,
  weights = FALSE
)
```

### Arguments

| | |
|---|---|
| fPrime | • a Seurat object of genes or SingleCellExperiment to be converted to a Seurat object |
| geneSet | • set of genes |
| radius | • the distance around the given set |
| metric | • the metric to use, one of umap, tsne, pca or nearest neighbor |
| numPCs | • used only if the metric is pca |
| weights | • whether to use edge weights in the NN case |

### Value

This returns a named vector whose values are distance from geneSet and whose names are the nearby genes.

### Examples

```
getExample = make.getExample()
STranspose = getExample('STranspose',toy=TRUE)
hallmark = getExample('hallmark')
geneSet = intersect(colnames(STranspose),hallmark[["HALLMARK_TNFA_SIGNALING_VIA_NFKB"]])
geometricallyNearby = getNearbyGenes(STranspose,geneSet,radius=0.2,metric='umap')
combinatoriallyNearby = getNearbyGenes(STranspose,geneSet,radius=1,metric='NN')
weightedNearby = getNearbyGenes(STranspose,'Myc',radius=1,metric='NN',weights=TRUE)
```

---

## getNearestNeighbourLists

*This function extracts a shared nearest neighbor network from a Seurat object*

---

### Description

This function extracts a shared nearest neighbor network from a Seurat object

### Usage

```
getNearestNeighbourLists(f, graph = defaultGraph(f))
```

### Arguments

| | |
|---|---|
| f | • a Seurat object or SingleCellExperiment to be converted to a Seurat object |
| graph | • which graph to extract. Defaults to paste0(f@active.assay,'_snn') |

### Value

- This returns dataframe of neighbors: nodeA - node names for node A nodeB - node names for node B weight - edge weight

### Examples

```
STranspose = make.getExample()('STranspose',toy=TRUE)
NN = getNearestNeighbourLists(STranspose)
```

---

## getObjectSubsetClusteringPValue

*This function computes a p-value for the geometric clustering of a gene set (in UMAP or PCA reduction) based on the median distance from its complement to the set.*

---

### Description

This function computes a p-value for the geometric clustering of a gene set (in UMAP or PCA reduction) based on the median distance from its complement to the set.

### Usage

```
getObjectSubsetClusteringPValue(
  fPrime,
  geneSubset,
  numTrials = 1000,
  reduction = "UMAP",
  numPCs = 10
)
```

### Arguments

| | |
|---|---|
| fPrime | • a transposed Seurat object, i.e. a Seurat object of genes or SingleCellExperiment to be converted to a Seurat object |
| geneSubset | • a subset of the genes which can be given as a character vector as a logical vector |
| numTrials | • the number of random trials to be carried out for randomised testing. Defaults to 1000. |
| reduction | • can be 'UMAP' or 'PCA', defaults to 'UMAP' |
| numPCs | • number of PCs to use if reduction is 'PCA' |

## Value

A p-value reporting how often a random subset of the same size is sufficiently clustered to produce an equally large distance from its complement.

## Examples

```
getExample = make.getExample()
STranspose = getExample('STranspose')
hallmark = getExample('hallmark',toy=TRUE)
geneSubset = intersect(colnames(STranspose),hallmark[["HALLMARK_TNFA_SIGNALING_VIA_NFKB"]])
p = getObjectSubsetClusteringPValue(STranspose,geneSubset,100)
```

---

# getObjectSubsetClusteringStatistics

*This function computes statistics for the geometric clustering of a gene set (in UMAP or PCA reduction) based on the median distance from its complement to the set.*

---

## Description

This function computes statistics for the geometric clustering of a gene set (in UMAP or PCA reduction) based on the median distance from its complement to the set.

## Usage

```
getObjectSubsetClusteringStatistics(
  fPrime,
  geneSubset,
  numTrials = 1000,
  reduction = "UMAP",
  numPCs = 10
)
```

## Arguments

| | |
|---|---|
| fPrime | • a transposed Seurat object, i.e. a Seurat object of genes or SingleCellExperiment to be converted to a Seurat object |
| geneSubset | • a subset of the genes which can be given as a character vector or as a logical vector |
| numTrials | • the number of random trials to be carried out for randomised testing. Defaults to 1000. |
| reduction | • can be 'UMAP' or 'PCA', defaults to 'UMAP' |
| numPCs | • number of PCs to use if reduction is 'PCA' |

### Value

A list of statistics resulting from the testing of randomised subsets of the same size as the given gene subset. These include subsetDistance, the actual median complement distance; randomSubsetDistance, the median complement distances for randomised subsets; pValue, computed by comparing the real and randomised distances; and zScore, the z-distance of the actual median distance from the mean of the randomised distances.

### Examples

```
getExample = make.getExample()
STranspose = getExample('STranspose',toy=TRUE)
hallmark = getExample('hallmark')
geneSubset = intersect(colnames(STranspose),hallmark[["HALLMARK_TNFA_SIGNALING_VIA_NFKB"]])
stats = getObjectSubsetClusteringStatistics(STranspose,geneSubset,100)
```

---

## getSubsetComponents *This is designed to dectect the components of a gene subset in the case where median complement distance detects clustering.*

---

### Description

This is designed to dectect the components of a gene subset in the case where median complement distance detects clustering.

### Usage

```
getSubsetComponents(fPrime, theSubset, alpha = 0.5, edgeCut = NA)
```

### Arguments

| | |
|---|---|
| fPrime | • a gene Seurat object or SingleCellExperiment |
| theSubset | • a subset of the genes |
| alpha | • a parameter typically less than one controling the granularity of the components. Defaults to .5 |
| edgeCut | • the maximum length of edges included in the subgraph whose components are returned. If it is NA (the default) it is computed using alpha. Otherwise, it can be supplied directly. |

### Value

A list of the components of the subset treated as a graph whose edges are determined by their distance in UMAP coordinates.

humanLRN *humanLRN*

## Description

A data frame giving 12019 human ligand receptor pairs

## Usage

```
humanLRN
```

## Format

a data frame with two columns, 'from' and 'to'

A data frame with two columns, 'from' and 'to'. Each row represents a human ligand - receptor pair.

## Source

This is taken from the nichenetr package, url = https://www.nature.com/articles/s41592-019-0667-5. Specifically we use the human ligand - receptor network.

ligandReceptorResults *ligandReceptorResults*

## Description

The result of performLigandReceptorAnalysis(smallXenium, delaunayNeighbours, "mouse", clusters,verbose=FALSE)

## Usage

```
ligandReceptorResults
```

## Format

A list of data frames.

A list containing: interactionsOnEdges - a data frame whose first two columns give the neighbouring cells and next two columns give their corresponding clusters. Each of the remaining columns is a logical corresponding to a ligand-receptor pair telling whether the ligand is expressed in the first cell and the receptor is expressed in the second cell. totalInteractionsByCluster - a dataframe where the first column gives a directed (sender-receiver) pair of clusters. The second column gives the total number of edges between those clusters. The remaining columns give the total numbers of edges on which particular ligand receptor interactions are present. meanInteractionsByCluster - a dataframe where the first column gives a directed (sender-receiver) pair of clusters. The second column gives the total number of edges

between those clusters. The remaining columns give the total numbers of edges on which particular ligand receptor interactions are present (for that cluster pair) divided by the total number of edges between those clusters. simResults - a dataframe where the rownames are sender-receiver cluster pairs and column names are ligand receptor pairs. Values give the number of simulations for which observed values are greater than simulated values. pValues - a dataframe where the rownames are sender-receiver cluster pairs and column names are ligand receptor pairs. Entries are uppertail pvalues describing whether a particular ligand receptor interaction is observed more frequently between 2 clusters than expected.

## Source

Created from smallXenium and delaunayNeighbours by using performLigandReceptorAnalysis(()

---

`make.getExample` *This function makes the function whichretrieves and makes example data objects.*

---

## Description

This function makes the function whichretrieves and makes example data objects.

## Usage

```
make.getExample()
```

## Value

This returns the function which retrieves and makes example data objects. The latter saves any object it has found for quicker return. Using the value 'list' causes it to return the list of all objects found so far.

## Examples

```
getExample = make.getExample()
## Provided:
smallXenium = getExample('smallXenium')
## Computed:
delaunayNeighbours = getExample('delaunayNeighbours')
```

makeLRInteractionHeatmap

*This function takes ligandReceptorResults and plots a heatmap of -log10(pvalues). If the minimum p-value is 0 a pseudocount of 0.001 will be added before log transformation.*

## Description

This function takes ligandReceptorResults and plots a heatmap of -log10(pvalues). If the minimum p-value is 0 a pseudocount of 0.001 will be added before log transformation.

## Usage

```
makeLRInteractionHeatmap(
  ligandReceptorResults,
  clusters,
  colours = c(),
  pValCutoffClusterPair = 0.05,
  pValCutoffLigRec = 0.05,
  labelClusterPairs = TRUE
)
```

## Arguments

`ligandReceptorResults`
- as returned by performLigandReceptorAnalysis()

`clusters`
- named vector of cell types where names are each cell and clusters are a factor

`colours`
- a named list of colours where names are clusters. If not specified the default pheatmap colour scheme will be used.

`pValCutoffClusterPair`
- a cutoff for showing interactions between two clusters. A cluster pair must have at least one ligand-receptor interaction pvalue < pValCutoffClusterPair. Defaults to 0.05.

`pValCutoffLigRec`
- a cutoff for showing interactions between a ligand and receptor. At least one cluster pair must have pvalue < pValCutoffLigRec for ligand-receptor pair. Defaults to 0.05.

`labelClusterPairs`
- show labels for cluster pairs. Defaults to TRUE.

## Value

matrix of -log10(pvalues) that underlies the heatmap.

## Examples

```
getExample = make.getExample()
clusters = getExample('clusters')
ligandReceptorResults = getExample('ligandReceptorResults')
cellTypePerCellTypeLigRecMatrix =
makeSummedLRInteractionHeatmap(ligandReceptorResults, clusters, "mean")
```

---

`makeSummedLRInteractionHeatmap` *This function takes ligandReceptorResults and plots a heatmap of the total number of ligand receptor interactions between clusters.*

---

## Description

This function takes ligandReceptorResults and plots a heatmap of the total number of ligand receptor interactions between clusters.

## Usage

```
makeSummedLRInteractionHeatmap(
  ligandReceptorResults,
  clusters,
  type,
  logScale = TRUE
)
```

## Arguments

| | |
|---|---|
| `ligandReceptorResults` | • as returned by performLigandReceptorAnalysis() |
| `clusters` | • named vector of cell types where names are each cell and clusters are a factor |
| `type` | • "total" or "mean" to plot raw total interactions or mean interactions per edge. |
| `logScale` | • plot heatmap using log scale (defaults to TRUE) |

## Value

matrix of total ligand receptor interactions that underlies t he heatmap.

## Examples

```
getExample = make.getExample()
clusters = getExample('clusters')
ligandReceptorResults = getExample('ligandReceptorResults')
cellTypePerCellTypeLigRecMatrix =
makeSummedLRInteractionHeatmap(ligandReceptorResults, clusters, "mean")
```

meanGeneClusterOnCellUMAP *Mean gene cluster on cell umap*

## Description

This function paints gene expression for a given gene cluster on cell umap.

## Usage

```
meanGeneClusterOnCellUMAP(f, fPrime, geneCluster)
```

## Arguments

| | |
|---|---|
| f | • a Seurat object of cells or SingleCellExperiment to be converted to a Seurat object |
| fPrime | • the corresponding Seurat object of genes SingleCellExperiment to be converted to a Seurat object |
| geneCluster | • a gene cluster of fPrime |

## Value

This returns a ggplot object

## Examples

```
getExample = make.getExample()
exSeuratObj = getExample('exSeuratObj',toy=TRUE)
STranspose = getExample('STranspose',toy=TRUE)
g = meanGeneClusterOnCellUMAP(exSeuratObj,STranspose,geneCluster=0)
```

meanZPerCluster *This finds the mean z-score for features in subsets of cells e.g., in each of the seurat_clusters*

## Description

This finds the mean z-score for features in subsets of cells e.g., in each of the seurat_clusters

## Usage

```
meanZPerCluster(f, features, clusterBy = "seurat_clusters", layer = "data")
```

### Arguments

| | |
|---|---|
| f | • a Seurat object of cells or SingleCellExperiment to be converted to a Seurat object |
| features | • a set of features of f |
| clusterBy | • the name of the column of f@meta.data to be used to subset the cells |
| layer | • the data layer to be used for z-scores |

### Value

This returns a data frame each of whose columns corresponds to a value of the clusterBy data. In the case where the clusterBy data is a factor or numeric, it prepends cluster_ to the column name.

### Examples

```
getExample = make.getExample()
exSeuratObj = getExample('exSeuratObj',toy=TRUE)
STranspose = getExample('STranspose',toy=TRUE)
df = meanZPerCluster(exSeuratObj,features=colnames(STranspose),
                     clusterBy='shortName')
```

---

## meanZPerClusterOnUMAP *This collects together mean z-score data together with UMAP coordinates from the gene seurat object for plotting.*

---

### Description

This collects together mean z-score data together with UMAP coordinates from the gene seurat object for plotting.

### Usage

```
meanZPerClusterOnUMAP(f, fPrime, clusterBy = "seurat_clusters", layer = "data")
```

### Arguments

| | |
|---|---|
| f | • a Seurat object of cells or SingleCellExperiment to be converted to a Seurat object |
| fPrime | • the corresponding Seurat object of genes SingleCellExperiment to be converted to a Seurat object |
| clusterBy | • the name of the column of f@meta.data to be used to subset the cells |
| layer | • the data layer to be used for z-scores |

### Value

This returns a data frame with the UMAP coordinates of the gene Seurat object and the average z-score for each gene within each of the cell clusters defined by the clusterBy column of the meta.data of f.

## Examples

```
getExample = make.getExample()
exSeuratObj = getExample('exSeuratObj',toy=TRUE)
STranspose = getExample('STranspose',toy=TRUE)
df = meanZPerClusterOnUMAP(exSeuratObj,STranspose,clusterBy='shortName')
```

---

`medianComplementDistance`

*This takes a set S of n points in dimension d given by an n x d matrix and a subset A given by a logical and returns the median distance from the complement to the given subset.*

---

## Description

This takes a set S of n points in dimension d given by an n x d matrix and a subset A given by a logical and returns the median distance from the complement to the given subset.

## Usage

```
medianComplementDistance(S, idx)
```

## Arguments

| | |
|---|---|
| `S` | • an n x d matrix representing a set of n points in dimension d |
| `idx` | • a logical of length n representing a subset of S. This should not be the empty set or all of S. |

## Value

This returns the median distance from the complement to the subset

## Examples

```
S = matrix(seq_len(12),ncol=2)
idx = c(rep(FALSE,3),rep(TRUE,3))
compDist = medianComplementDistance(S,idx)
```

`medianComplementPValue` *This takes a set S of n points in dimension d and a subset A and computes a p-value for the co-localization of the subset by comparing the median complement distance for the given set to values of the median complement distance computed for random subsets of the same size.*

## Description

This takes a set S of n points in dimension d and a subset A and computes a p-value for the co-localization of the subset by comparing the median complement distance for the given set to values of the median complement distance computed for random subsets of the same size.

## Usage

```
medianComplementPValue(S, idx, numTrials = 1000, returnTrials = FALSE)
```

## Arguments

| | |
|---|---|
| `S` | • an n x d matrix representing a set of n points in dimension d |
| `idx` | • a logical of length n representing a subset of S. This should not be the empty set or all of S. |
| `numTrials` | • the number of random trials to perform, defaults to 1000 |
| `returnTrials` | • whether to report the real and random median complement distances. |

## Value

By default this reports a p-value. If returnTrials is set, this returns a list giving the p-value, the actual complement distance and the random complement distances.

## Examples

```
library(Seurat)
getExample = make.getExample()
STranspose = getExample('STranspose',toy=TRUE)
hallmark = getExample('hallmark')
S = data.matrix(FetchData(STranspose,c('umap_1','umap_2')))
idx = colnames(STranspose) %in% hallmark[["HALLMARK_TNFA_SIGNALING_VIA_NFKB"]]
mcpv = medianComplementPValue(S,idx,numTrials=100)
```

`moransI` *moransI*

## Description

A data fame containing Moran's I and related pvalues.

## Usage

```
moransI
```

## Format

A data fame containing Moran's I and related pvalues.

Moran's I values calculated for the genes in smallXenium (using the SCT assay). Pvalues derived using 100 permutations.

## Source

Created from smallXenium and delaunayNeighbours by using runMoransI()

`moransILigandReceptor` *moransILigandReceptor*

## Description

Moran's I for the ligand receptor pairs

## Usage

```
moransILigandReceptor
```

## Format

A data frame showing the spatial autocorrelation of the 28 ligand receptor pairs

A data frame with rownames giving the 28 ligand-receptor pairs and columns moransI and pValues

## Source

Computed using the function runMoransI on the object edgeSeurat and neighbours edgeNeighbours = computeEdgeGraph(delaunayNeighbours) with 100 trials. For more informations see the CatsCradleSpatial vignette.

mouseLRN *mouseLRN*

## Description

A data frame giving 11592 mouse ligand receptor pairs

## Usage

```
mouseLRN
```

## Format

a data frame with two columns, 'from' and 'to'

A data frame with two columns, 'from' and 'to'. Each row represents a mouse ligand - receptor pair.

## Source

This is taken from the nichenetr package, url = https://www.nature.com/articles/s41592-019-0667-5. Specifically, we use the mouse ligand - receptor network.

nbhdsAsEdgesToNbhdsAsList *nbhdsAsEdgesToNbhdsAsList*

## Description

This function takes a set of neighbourhoods given by edges and turns it into a named list giving the memberships of each neighbourhood

## Usage

```
nbhdsAsEdgesToNbhdsAsList(cells, neighbourhoods, self = FALSE)
```

## Arguments

| | |
|---|---|
| cells | • The cells whose neighbourhoods to extract. |
| neighbourhoods | • neighbourhoods given as a data frame with columns nodeA and nodeB, for example the output of collapseNeighbourhoods |
| self | • include cell in its neighbourhood, defaults to FALSE |

## Value

a named list with memberships of the neighbourhoods of cells

## Examples

```
delaunayNeighbours = make.getExample()('delaunayNeighbours')
cells = unique(c(delaunayNeighbours[,'nodeA'],delaunayNeighbours[,'nodeB']))
nbhdsList = nbhdsAsEdgesToNbhdsAsList(cells,delaunayNeighbours)
```

---

# neighbourhoodDiameter *neighbourhoodDiameter*

---

## Description

This function takes a list of neighbourhoods and and the centroids of the cells and finds their diameters, i.e., for each neighbourhood, the maximum distance between.

## Usage

```
neighbourhoodDiameter(neighbourhoods, centroids)
```

## Arguments

| | |
|---|---|
| neighbourhoods | • a list of neighbourhoods as returned by nbhdsAsEdgesToNbhdsAsList |
| centroids | • the centroids of the cells |

## Value

a named numeric. The names are the names of the list neighbourhoods and the values are the maximum distance within each neighbourhood

## Examples

```
getExample = make.getExample()
centroids = getExample('centroids')
delaunayNeighbours = getExample('delaunayNeighbours')
cells = unique(c(delaunayNeighbours[,'nodeA'],delaunayNeighbours[,'nodeB']))
nbhds = nbhdsAsEdgesToNbhdsAsList(cells,delaunayNeighbours)
diameters = neighbourhoodDiameter(nbhds[seq_len(100)],centroids)
```

`orderGeneSetPValues` *This orders the gene set p-values (or -log10 p-values) and applies a cutoff (if given) to show only the significant gene sets for each gene cluster*

## Description

This orders the gene set p-values (or -log10 p-values) and applies a cutoff (if given) to show only the significant gene sets for each gene cluster

## Usage

```
orderGeneSetPValues(M, ascending = TRUE, cutoff = NULL, nameTag = "")
```

## Arguments

| | |
|---|---|
| `M` | • A matrix of gene set p-values (or their logs) to be ordered by their significance |
| `ascending` | • Direction in which to order the columns. Defaults to TRUE, so that p-values will be ordered according to decreasing significance, should be set to FALSE if ordering -log p-value |
| `cutoff` | • if non-null this is used to extract only significant cases |
| `nameTag` | • can be used to modify the names of the list. |

## Value

This returns a list of whose entries are data frames, one for each gene cluster, each giving the significant gene sets for that cluster and their significance.

`performLigandReceptorAnalysis` *Given a seurat object, a spatial graph, clusters and species this function identifies ligand-receptor interactions between neighbouring cells, identifies ligand-receptor interactions within and between clusters and calculates whether these are observed more frequently than expected by chance. If the "analytical" method is selected, an upper tail p-value for observing a given number of A-B edges positive for a given interaction is calculated using a binomial test (R pbinom function) where: q = number of A-B edges positive for an interaction size = total number of A-B edges prob = pL*pR Where pL is the probability of a cell expressing a specific ligand (number of cells positive for a ligand/total cells), and pR is the probability of a cell expressing a specific receptor (number of cells positive for a receptor/total cells). If conditional = True p-values will be calculated given the proportion of cells that express ligands and receptors in the specific clusters (pL = number of cells in cluster A positive for a ligand/number of cells in cluster A, pR = number of cells in cluster B positive for a receptor/number of cells in cluster B). We recommend to use the analytical method, which has a much faster runtime than the permutation-based method, however for legacy purposes and user flexibility we retain the permutation-based method.*

---

## Description

Given a seurat object, a spatial graph, clusters and species this function identifies ligand-receptor interactions between neighbouring cells, identifies ligand-receptor interactions within and between clusters and calculates whether these are observed more frequently than expected by chance. If the "analytical" method is selected, an upper tail p-value for observing a given number of A-B edges positive for a given interaction is calculated using a binomial test (R pbinom function) where: q = number of A-B edges positive for an interaction size = total number of A-B edges prob = pL*pR Where pL is the probability of a cell expressing a specific ligand (number of cells positive for a ligand/total cells), and pR is the probability of a cell expressing a specific receptor (number of cells positive for a receptor/total cells). If conditional = True p-values will be calculated given the proportion of cells that express ligands and receptors in the specific clusters (pL = number of cells in cluster A positive for a ligand/number of cells in cluster A, pR = number of cells in cluster B positive for a receptor/number of cells in cluster B).

We recommend to use the analytical method, which has a much faster runtime than the permutation-based method, however for legacy purposes and user flexibility we retain the permutation-based method.

## Usage

```
performLigandReceptorAnalysis(
  obj,
  spatialGraph,
  species,
  clusters,
```

```
  method = "analytical",
  conditional = FALSE,
  minEdgesPos = 10,
  nSim = 1000,
  lrn = getLigandReceptorNetwork(species),
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| obj | • a Seurat object |
| spatialGraph | • a data frame of neighbouring cell pairs. |
| species | • either 'human' or 'mouse' |
| clusters | • named vector of clusters where names are each cell and clusters are a factor |
| method | • method for computing p-values. Defaults to "analytical". If "permutation" is selected p-values are calculated by comparison to randomised graphs (note this is slower than the analytical approach). |
| conditional | • if method is "analytical" and conditional is true, p-values will be calculated given the proportion of cells that express ligands and receptors in the specific clusters. Otherwise global proportions of ligand and receptor expression are used. Defaults to FALSE. |
| minEdgesPos | • the minimum edges that need to be positive for a ligand-receptor interaction between two clusters for a p-value to be calculated. Only taken into consideration when the analytical method is selected. |
| nSim | • number of simulations to perform for pvalue calculation. |
| lrn | • a ligand-receptor network, i.e., a data frame with columns from and to. By default, it retrieves the nichenetr ligand receptor network |
| verbose | • whether to print trace, defaults to TRUE |

## Value

A list containing: interactionsOnEdges - a sparse matrix where the rownames give pairs of neighbouring cells and column names give ligand-receptor pairs. Entries are TRUE if the ligand is expressed in the first cell and the receptor is expressed in the second cell and FALSE if not. interactionsOnEdgesMeta - a dataframe where the first two columns are the cells that comprise the edges in interactionsOnEdges, and the next two columns are their clusters. totalInteractionsByCluster - a dataframe where the rownames are sender-receiver cluster pairs and column names are ligand receptor pairs. Entries are total numbers of edges on which particular ligand receptor interactions are present. meanInteractionsByCluster - a dataframe where the rownames are sender-receiver cluster pairs and column names are ligand receptor pairs. Entries are total numbers of edges on which particular ligand receptor i nteractions are present (for that cluster pair) divided by the total number of edges between those clusters. simResults - a dataframe where the rownames are sender-receiver cluster pairs and column names are ligand receptor pairs. Values give the number of simulations for which observed values are greater than simulated values. Only returned if method = "permutation". pValues - a dataframe where the rownames are sender-receiver cluster pairs and column names are ligand receptor pairs. Entries are uppertail p-values describing whether a particular ligand receptor

interaction is observed more frequently between 2 clusters than expected. totalEdges - a vector of total edges between cluster pairs.

## Examples

```
getExample = make.getExample()
smallXenium = getExample('smallXenium')
delaunayNeighbours = getExample('delaunayNeighbours')
clusters = getExample('clusters')
performLigandReceptorAnalysis(smallXenium, delaunayNeighbours,
                                      "mouse", clusters)
```

---

`performLigandReceptorAnalysisAnalytical`

*Given a seurat object, a spatial graph, clusters and species this function identifies ligand-receptor interactions between neighbouring cells, identifies ligand-receptor interactions within and between clusters and calculates whether these are observed more frequently than expected by chance using an analytical approach.*

---

## Description

Given a seurat object, a spatial graph, clusters and species this function identifies ligand-receptor interactions between neighbouring cells, identifies ligand-receptor interactions within and between clusters and calculates whether these are observed more frequently than expected by chance using an analytical approach.

## Usage

```
performLigandReceptorAnalysisAnalytical(
  obj,
  spatialGraph,
  species,
  clusters,
  conditional = FALSE,
  lrn = getLigandReceptorNetwork(species),
  minEdgesPos = 10
)
```

## Arguments

| | |
|---|---|
| `obj` | • a Seurat object |
| `spatialGraph` | • a data frame of neighbouring cell pairs. |
| `species` | • either 'human' or 'mouse' |
| `clusters` | • named vector of clusters where names are each cell and clusters are a factor |

| | |
|---|---|
| conditional | • if method is "analytical" and conditional is true, p-values will be calculated given the proportion of cells that express ligands and receptors in the specific clusters. Otherwise global proportions of ligand and receptor expression are used. Defaults to FALSE. |
| lrn | • a ligand-receptor network, i.e., a data frame with columns from and to. By default, it retrieves the nichenetr ligand receptor network |
| minEdgesPos | • the minimum edges that need to be positive for a ligand-receptor interaction between two clusters for a p-value to be calculated. |

## Value

A list containing: interactionsOnEdges - a sparse matrix where the rownames give pairs of neighbouring cells and column names give ligand-receptor pairs. Entries are TRUE if the ligand is expressed in the first cell and the receptor is expressed in the second cell and FALSE if not. interactionsOnEdgesMeta - a dataframe where the first two columns are the cells that comprise the edges in interactionsOnEdges, and the next two columns are their clusters. totalInteractionsByCluster - a dataframe where the rownames are sender-receiver cluster pairs and column names are ligand receptor pairs. Entries are total numbers of edges on which particular ligand receptor interactions are present. meanInteractionsByCluster - a dataframe where the rownames are sender-receiver cluster pairs and column names are ligand receptor pairs. Entries are total numbers of edges on which particular ligand receptor interactions are present (for that cluster pair) divided by the total number of edges between those clusters. pValues - a dataframe where the rownames are sender-receiver cluster pairs and column names are ligand receptor pairs. Entries are uppertail p-values describing whether a particular ligand receptor interaction is observed more frequently between 2 clusters than expected. totalEdges - a vector of total edges between cluster pairs.

---

`performLigandReceptorAnalysisPermutation`

*Given a seurat object, a spatial graph, clusters and species this function identifies ligand-receptor interactions between neighbouring cells, identifies ligand-receptor interactions within and between clusters and calculates whether these are observed more frequently than expected by chance using a permutation-based approach.*

---

## Description

Given a seurat object, a spatial graph, clusters and species this function identifies ligand-receptor interactions between neighbouring cells, identifies ligand-receptor interactions within and between clusters and calculates whether these are observed more frequently than expected by chance using a permutation-based approach.

## Usage

```
performLigandReceptorAnalysisPermutation(
  obj,
  spatialGraph,
```

```
  species,
  clusters,
  nSim = 1000,
  lrn = getLigandReceptorNetwork(species),
  minEdgesPos = 10,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| obj | • a Seurat object |
| spatialGraph | • a data frame of neighbouring cell pairs. |
| species | • either 'human' or 'mouse' |
| clusters | • named vector of clusters where names are each cell and clusters are a factor |
| nSim | • number of simulations to perform for p value calculation. |
| lrn | • a ligand-receptor network, i.e., a data frame with columns from and to. By default, it retrieves the nichenetr ligand receptor network |
| minEdgesPos | • the minimum edges that need to be positive for a ligand-receptor interaction between two clusters for a p-value to be calculated. Only taken into consideration when the analytical method is selected. |
| verbose | • whether to print trace, defaults to TRUE |

## Value

A list containing: interactionsOnEdges - a sparse matrix where the rownames give pairs of neighbouring cells and column names give ligand-receptor pairs. Entries are TRUE if the ligand is expressed in the first cell and the receptor is expressed in the second cell and FALSE if not. interactionsOnEdgesMeta - a dataframe where the first two columns are the cells that comprise the edges in interactionsOnEdges, and the next two columns are their clusters. totalInteractionsByCluster - a dataframe where the rownames are sender-receiver cluster pairs and column names are ligand receptor pairs. Entries are total numbers of edges on which particular ligand receptor interactions are present. meanInteractionsByCluster - a dataframe where the rownames are sender-receiver cluster pairs and column names are ligand receptor pairs. Entries are total numbers of edges on which particular ligand receptor i nteractions are present (for that cluster pair) divided by the total number of edges between those clusters. simResults - a dataframe where the rownames are sender-receiver cluster pairs and column names are ligand receptor pairs. Values give the number of simulations for which observed values are greater than simulated values. pValues - a dataframe where the rownames are sender-receiver cluster pairs and column names are ligand receptor pairs. Entries are uppertail p-values describing whether a particular ligand receptor interaction is observed more frequently between 2 clusters than expected. totalEdges - a vector of total edges between cluster pairs.

## Examples

```
getExample = make.getExample()
smallXenium = getExample('smallXenium')
delaunayNeighbours = getExample('delaunayNeighbours')
clusters = getExample('clusters')
```

```
performLigandReceptorAnalysis(smallXenium, delaunayNeighbours,
                                        "mouse", clusters,  minEdgesPos = 10, nSim = 10,
                                         verbose=FALSE)
```

---

`permuteColumns` *This function permutes the columns of a sparse dcG matrix.*

---

## Description

This function permutes the columns of a sparse dcG matrix.

## Usage

```
permuteColumns(M)
```

## Arguments

| | |
|---|---|
| `M` | • a binarised expression matrix in sparse format where rows are cells and columns |

## Value

This returns a matrix in which the values have been permuted within columns.

---

`permuteMatrix` *This function permutes the rows of a matrix.*

---

## Description

This function permutes the rows of a matrix.

## Usage

```
permuteMatrix(M)
```

## Arguments

| | |
|---|---|
| `M` | • a binarised expression matrix where rows are genes and columns |

## Value

This returns a matrix in which the values have been permuted within rows.

plotLRDotplot *This is a function to create a dotplot using the ligand receptor results*

## Description

This is a function to create a dotplot using the ligand receptor results

## Usage

```
plotLRDotplot(
  ligandReceptorResults,
 senderClusters = unique(ligandReceptorResults$interactionsOnEdgesMeta$cellTypeA),
 receiverClusters = unique(ligandReceptorResults$interactionsOnEdgesMeta$cellTypeB),
  padjCutoff = 0.05,
  pvalCutoff = F,
  splitBy = "sender"
)
```

## Arguments

| | |
|---|---|
| ligandReceptorResults | • ligandReceptorResults calculated using performLigandReceptorAnalysis(). |
| senderClusters | • sender clusters to plot (defaults to all). |
| receiverClusters | • receiver clusters to plot (defaults to all). |
| padjCutoff | • only plot results with p-adj < padjCutoff (defaults to 0.05). |
| pvalCutoff | • only plot results with p-value < pvalCutoff (defaults to False in which case padjCutoff is used). |
| splitBy | • split plots by "sender" or "receiver" cell types (defaults to sender). |

## Value

matrix of total ligand receptor interactions that underlies the heatmap.

## Examples

```
getExample = make.getExample()
centroids = getExample('centroids')
ligandReceptorResults = getExample('ligandReceptorResults')
p = plotLRDotplot(ligandReceptorResults)
```

`predictAnnotation` *This function makes annotation predictions for a set of genes based on gene sets (e.g., hallmark) and a CatsCradle object by considering the annotations of its neighboring genes.*

## Description

This function makes annotation predictions for a set of genes based on gene sets (e.g., hallmark) and a CatsCradle object by considering the annotations of its neighboring genes.

## Usage

```
predictAnnotation(
  genes,
  geneSets,
  fPrime,
  radius,
  metric = "umap",
  numPCs = NULL,
  normaliseByGeneSet = TRUE,
  normaliseByDistance = TRUE,
  normaliseToUnitVector = TRUE
)
```

## Arguments

| | |
|---|---|
| `genes` | • a character vector of genes |
| `geneSets` | • a set of annotations, e.g., hallmark or GO |
| `fPrime` | • a Seurat object of genes SingleCellExperiment to be converted to a Seurat object |
| `radius` | • radius for prediction neighborhood |
| `metric` | • reduction or NN, defaults to umap |
| `numPCs` | • used only if reduction is pca, defaults to NULL |
| `normaliseByGeneSet` | • determines whether vector annotations are normalised by gene set size. Defaults to TRUE |
| `normaliseByDistance` | • determines whether neighbor contributions are normalised by edge weight. Defaults to TRUE. |
| `normaliseToUnitVector` | • determines whether to normalise returned values to unit length. Defaults to TRUE |

## Value

This returns a list of prediction vectors, one vector for each gene in genes, each vector corresponding to the sets in geneSets

## Examples

```
getExample = make.getExample()
STranspose = getExample('STranspose',toy=TRUE)
STranspose_sce = getExample('STranspose_sce',toy=TRUE)
hallmark = getExample('hallmark',toy=TRUE)
set.seed(100)
genes = sample(colnames(STranspose),5)
predictions = predictAnnotation(genes,hallmark,STranspose,radius=.5)
predictions_sce = predictAnnotation(genes,hallmark,STranspose_sce,radius=.5)
```

---

# predictAnnotationAllGenes

*This function predicts the functions of all genes based on the functions of their neighbours.*

---

## Description

This function predicts the functions of all genes based on the functions of their neighbours.

## Usage

```
predictAnnotationAllGenes(
  geneSets,
  fPrime,
  radius,
  metric = "umap",
  normaliseByGeneSet = TRUE,
  normaliseByDistance = TRUE,
  normaliseToUnitVector = TRUE
)
```

## Arguments

| | |
|---|---|
| geneSets | • a set of gene sets, e.g., hallmark |
| fPrime | • a transposed Seurat object (generated with transposeObject()) or SingleCellExperiment to be converted to a Seurat object |
| radius | • radius of the region to use for prediction |
| metric | • reduction or NN, defaults to umap |
| normaliseByGeneSet | • normalise by size of each gene set, defaults to TRUE |
| normaliseByDistance | • attenutate neighbour contributions based on distance, defaults to TRUE |
| normaliseToUnitVector | • return results as unit vectors, defaults to TRUE |

### Value

- A list where names are genes and values are vectors of gene annotations whose entries correspond to the geneSets

### Examples

```
getExample = make.getExample()
STranspose = getExample('STranspose',toy=TRUE)
hallmark = getExample('hallmark',toy=TRUE)
predictions = predictAnnotationAllGenes(hallmark,STranspose,radius=.5)
```

---

## predictGeneAnnotationImpl

*This function is the implementation for predicting the functions of a gene based on the functions of its neighbours.*

---

### Description

This function is the implementation for predicting the functions of a gene based on the functions of its neighbours.

### Usage

```
predictGeneAnnotationImpl(
  gene,
  fPrime,
  genesAnno,
  radius,
  metric,
  numPCs = NULL,
  normaliseByDistance = TRUE
)
```

### Arguments

| | |
|---|---|
| gene | • gene to annotate |
| fPrime | • a Seurat object of genes or SingleCellExperiment to be converted to a Seurat object |
| genesAnno | • genes annotated with gene sets |
| radius | • radius of neighbours to consider |
| metric | • which metric to use to discover neighbours, can be one of 'umap', 'tsne', 'pca', 'NN', defaults to umap |
| numPCs | • used only if metric is pca. Defaults to NULL |
| normaliseByDistance | • choose whether to normalise contributions of neighbors by their distance, defaults to TRUE |

## Value

This returns a named list. The names are the anotations that apply to the neighbour genes, the values are the relative wieghts of the contributions.

## Examples

```
getExample = make.getExample()
STranspose = getExample('STranspose',toy=TRUE)
hallmark = getExample('hallmark',toy=TRUE)
genesAnno = annotateGenesByGeneSet(hallmark)
predictions = predictGeneAnnotationImpl('Myc',STranspose,genesAnno,
radius=.5,metric='umap')
```

---

`randomiseGraph` *This function performs degree-preserving randomisation of neighbour graphs.*

---

## Description

This function performs degree-preserving randomisation of neighbour graphs.

## Usage

```
randomiseGraph(spatialGraph, maxTries = 1000)
```

## Arguments

| | |
|---|---|
| `spatialGraph` | • a spatial graph in neighbour list format. |
| `maxTries` | • the maximum number of tries to remove self edges during graph randomisation. If self edges are remaining this will be reported. |

## Value

A randomised graph where degree from the original graph is preserved. We also report any duplicated edges.

## randomiseNodeIndices *This function generates random indices for node B*

### Description

This function generates random indices for node B

### Usage

```
randomiseNodeIndices(neighborListDf, n = 100, useWeights = FALSE)
```

### Arguments

| | |
|---|---|
| neighborListDf | • a dataframe containing the neighborlist |
| n | • the number of times to randomise indices |
| useWeights | • whether to preserve edgeweights. |

### Value

- a matrix with randomised indices for node B

### Examples

```
NN = make.getExample()('NN')
NN = desymmetriseNN(NN)
randomIndices = randomiseNodeIndices(NN,10,TRUE)
```

## readGmt *This function reads in gene sets in .gmt format*

### Description

This function reads in gene sets in .gmt format

### Usage

```
readGmt(gmtFile, addDescr = FALSE)
```

### Arguments

| | |
|---|---|
| gmtFile | • a .gmt file containing gene sets, e.g., Hallmark of GO |
| addDescr | • include gene set description (2nd column in .gmt file) in gene set name |

### Value

- A named list of gene sets

`runGeometricClusteringTrials` *This runs random trials to determine the statistical significance of the clustering of a set of points within a larger set.*

## Description

This function takes a matrix whose rows are geometric coordinates and a subset of these points either given as a character vector which is a subset of the rownames or as a logical vector. It returns statistics on the mean distance of the complement to the subset.

## Usage

```
runGeometricClusteringTrials(S, geneSubset, numTrials)
```

## Arguments

| | |
|---|---|
| `S` | • a set of points given as a matrix. The rows are the coordinates of these points |
| `geneSubset` | • this is either a subset of the rownames of S or a logical whose length is nrow(S) |
| `numTrials` | • the number or random trials to perform |

## Value

This returns a list. subsetDistance gives the median complement distance for the actual set, randomSubsetDistance gives the complement distances for the numTrials random sets, pValue gives a p-value based on the rank of the actual distance among the random distances and zScore gives its z-score.

## Examples

```
library(Seurat)
getExample = make.getExample()
STranspose = getExample('STranspose',toy=TRUE)
hallmark = getExample('hallmark')
S = data.matrix(FetchData(STranspose,c('umap_1','umap_2')))
geneSubset = rownames(S) %in% hallmark[["HALLMARK_TNFA_SIGNALING_VIA_NFKB"]]
geneClustering = runGeometricClusteringTrials(S,geneSubset,100)
```

runMoransI *This function takes a matrix where rows are features and columns are cells, and a neighbourhood list, and computes Moran's I.*

## Description

This function takes a matrix where rows are features and columns are cells, and a neighbourhood list, and computes Moran's I.

## Usage

```
runMoransI(
  obj,
  spatialGraph,
  assay = "RNA",
  layer = "data",
  nSim = 100,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| obj | • a Seurat object |
| spatialGraph | • a data frame of neighbouring cell pairs. |
| assay | • assay to pull data from, defaults to RNA. |
| layer | • layer to pull data from, defaults to data. |
| nSim | • number of simulations to perform for p value calculation. Defaults to 100. |
| verbose | • whether to print trace, defaults to TRUE |

## Value

a dataframe containing Moran's I and p values for each feature.

## Examples

```
getExample = make.getExample()
smallXenium = getExample('smallXenium',toy=TRUE)
delaunayNeighbours = getExample('delaunayNeighbours',toy=TRUE)
moransI = runMoransI(smallXenium, delaunayNeighbours, assay = "SCT",
layer = "data", nSim = 10, verbose = FALSE)
```

sankeyFromMatrix *This makes a sankey graph from a matrix of average expression. Our "Cat's Cradle".*

## Description

This makes a sankey graph from a matrix of average expression. Our "Cat's Cradle".

## Usage

```
sankeyFromMatrix(
  M,
  disambiguation = c("R_", "C_"),
  fontSize = 20,
  minus = "red",
  plus = "blue",
  height = 1200,
  width = 900
)
```

## Arguments

| | |
|---|---|
| M | • a matrix of gene expression |
| disambiguation | • used to distinguish between the row names and the column names if these overlap |
| fontSize | • defaults to 20 |
| minus | • colour to use for links with negative values |
| plus | • colour for positive values |
| height | • height in pixels, defaults to 1200 |
| width | • width in pixels, defaults to 900 |

## Value

A sankey graph

## Examples

```
set.seed(100)
M = matrix(runif(12)-.3,nrow=3)
rownames(M) = as.character(seq_len(3))
colnames(M) = as.character(seq_len(4))
sankey = sankeyFromMatrix(M)
```

`seuratCells` *seuratCells*

## Description

A vector of cells used for subsetting exSeuratObj

## Usage

```
seuratCells
```

## Format

A vector of cells

A vector of cells consisting of half the cells from each seurat_cluster in exSeuratObj used to subset this object to give toy examples.

## Source

Computed by retrieving half the cells from each cluster in exSeuratObj

`seuratGenes` *seuratGenes*

## Description

A vector of genes used for subsetting exSeuratObj

## Usage

```
seuratGenes
```

## Format

A vector of genes

A vector of the top 100 most variable genes in exSeuratObj used to subset this object to give toy examples.

## Source

Computed by retrieving the data layer from exSeuratObj and subsetting to the 100 genes with the highest standard deviation.

`smallXenium` *smallXenium*

## Description

A spatial Seurat object of 4261 cells and 248 genes

## Usage

```
smallXenium
```

## Format

A Seurat object

A spatial Seurat object subset from the Xenium object used in https://satijalab.org/seurat/articles/seurat5_spatial_vignett

## Source

This is subset from the Xenium spatial Seurat object https://cf.10xgenomics.com/samples/xenium/1.0.2/Xenium_V1_FF_Mo
to include a small region of the field of view surrounding the dentate gyrus.

`stripGeneSet` *This function strips out non-gene information from the beginning of GO sets, etc.*

## Description

This function strips out non-gene information from the beginning of GO sets, etc.

## Usage

```
stripGeneSet(geneSet)
```

## Arguments

| | |
|---|---|
| `geneSet` | • a list of gene sets |

## Value

a named list of gene sets

## symmetriseNN *This symmetrises a nearest neighbors graph.*

### Description

This first checks to see if the NN graph is symmetric and if not symmetrises it.

### Usage

```
symmetriseNN(NN)
```

### Arguments

| | |
|---|---|
| NN | • a nearest neighbors graph as returned by getNearestNeighbourLists |

### Value

a nearest neighbors graph

### Examples

```
NN = make.getExample()('NN',toy=TRUE)
NNStar = symmetriseNN(NN)
```

## symmetryCheckNN *Tests whether a nearest neighbor graph is symmetric*

### Description

The nearest neighbor relationship is not inherently symmetric. This tests whether the nearest neighbor graph retrieved from a Seurat object is.

### Usage

```
symmetryCheckNN(NN)
```

### Arguments

| | |
|---|---|
| NN | • a nearest neighbor graph. This is in the form of a data frame as returned by getNearestNeighbourLists. Its coloumns include nodeA and nodeB. |

### Value

TRUE or FALSE

### Examples

```
 NN = make.getExample()('NN',toy=TRUE)
symmetryTest = symmetryCheckNN(NN)
```

## tagRowAndColNames *This gussies up the rownames and colnames of M*

### Description

This gussies up the rownames and colnames of M

### Usage

```
tagRowAndColNames(M, ccTag = "CC_", gcTag = "GC_")
```

### Arguments

| | |
|---|---|
| M | • a matrix, typically the average expression matrix |
| ccTag | • a prefix for the row (cell cluster) names |
| gcTag | • a prefix for the column (gene cluster) names |

### Value

The same matrix with fancier row and col names

### Examples

```
getExample = make.getExample()
averageExpMatrix = getExample('averageExpMatrix',toy=TRUE)
averageExpMatrix = tagRowAndColNames(averageExpMatrix,'cellCluster_','geneCluster_')
```

## transposeObject *Create the transpose of a Seurat object*

### Description

This takes a Seurat object f and creates a new Seurat object whose expression matrix is the transpose of that of f. This can also be a SingleCellExperiment which will be converted to a Seurat object

### Usage

```
transposeObject(
  f,
  active.assay = "RNA",
  npcs = 30,
  dims = seq_len(20),
  res = 1,
  returnType = "Seurat",
  verbose = FALSE
)
```

### Arguments

| | |
|---|---|
| f | • a Seurat object |
| active.assay | • the assay to use. Defaults to 'RNA' |
| npcs | • number of principal components, defaults to 30 |
| dims | • dimensions to use for umap and nearest neighbors, defaults to 1:20 |
| res | • the clustering resolution, defaults to 1 |
| returnType | • Will return a SingleCellExperiment if this is either of SCE, SingleCellExperiment or their lower-case equivalents. Otherwise, returns a Seurat object |
| verbose | • Controls whether to display trace from the Seurat functions. Defaults to FALSE |

### Value

A Seurat object or SingleCellExperiment

### Examples

```
exSeuratObj = make.getExample()('exSeuratObj',toy=TRUE)
STranspose = transposeObject(exSeuratObj)
STransposeAsSCE = transposeObject(exSeuratObj,returnType='SCE')
```

---

## xeniumCells *xeniumCells*

---

### Description

A vector of cells used for subsetting exSeuratObj

### Usage

```
xeniumCells
```

### Format

A vector of cells

A vector of cells consisting of approximately one quarter of the cells in smallXenium used to subset this object to give toy examples.

### Source

We extracted a rectangle whose width and height were one half the width and height of smallXenium and which was centered in the field of view of smallXenium

# Index


∗ **datasets**
exSeuratObj, 28
humanLRN, 42
ligandReceptorResults, 42
moransI, 50
moransILigandReceptor, 50
mouseLRN, 51
seuratCells, 69
seuratGenes, 69
smallXenium, 70
xeniumCells, 73

aggregateFeatureMatrix, 4
aggregateGeneExpression, 5
annotateGeneAsVector, 6
annotateGenesByGeneSet, 6
annotateLRInteractionCounts, 7

cellTypesPerCellTypeGraphFromCellMatrix, 8
cellTypesPerCellTypeGraphFromNbhdMatrix, 9
collapseExtendedNBHDs, 10
combinatorialSpheres, 11
computeCellTypesPerCellTypeMatrix, 12
computeEdgeGraph, 12
computeEdgeObject, 13
computeGraphEmbedding, 14
computeMoransI, 15
computeNBHDByCTMatrix, 15
computeNBHDVsCTObject, 16
computeNeighbourEnrichment, 17
computeNeighboursDelaunay, 18
computeNeighboursEuclidean, 19
convertToLong, 19
countLRInteractionsPerCell, 20
cullEdges, 21

desymmetriseNN, 21
directedHausdorfDistance, 22

edgeCutoffsByClustering, 23
edgeCutoffsByPercentile, 24
edgeCutoffsByWatershed, 24
edgeCutoffsByZScore, 25
edgeLengthPlot, 26
edgeLengthsAndCellTypePairs, 27
exampleObjects, 27
exSeuratObj, 28

formatData, 28

geneSetsVsGeneClustersPValueMatrix, 29
getAverageExpressionDF, 30
getAverageExpressionMatrix, 30
getBinarisedMatrix, 31
getClusterOrder, 32
getExtendedNBHDs, 32
getFeatureZScores, 33
getGeneClusterAveragesPerCell, 34
getGeneNeighbors, 35
getInteractionsOnEdges, 35
getLigandReceptorNetwork, 36
getLigandReceptorPairsInPanel, 37
getNearbyGenes, 37
getNearestNeighbourLists, 38
getObjectSubsetClusteringPValue, 39
getObjectSubsetClusteringStatistics, 40
getSubsetComponents, 41

humanLRN, 42

ligandReceptorResults, 42

make.getExample, 43
makeLRInteractionHeatmap, 44
makeSummedLRInteractionHeatmap, 45
meanGeneClusterOnCellUMAP, 46
meanZPerCluster, 46
meanZPerClusterOnUMAP, 47
medianComplementDistance, 48

medianComplementPValue, 49
moransI, 50
moransILigandReceptor, 50
mouseLRN, 51

nbhdsAsEdgesToNbhdsAsList, 51
neighbourhoodDiameter, 52

orderGeneSetPValues, 53

performLigandReceptorAnalysis, 53
performLigandReceptorAnalysisAnalytical, 56
performLigandReceptorAnalysisPermutation, 57
permuteColumns, 59
permuteMatrix, 59
plotLRDotplot, 60
predictAnnotation, 61
predictAnnotationAllGenes, 62
predictGeneAnnotationImpl, 63

randomiseGraph, 64
randomiseNodeIndices, 65
readGmt, 65
runGeometricClusteringTrials, 66
runMoransI, 67

sankeyFromMatrix, 68
seuratCells, 69
seuratGenes, 69
smallXenium, 70
stripGeneSet, 70
symmetriseNN, 71
symmetryCheckNN, 71

tagRowAndColNames, 72
transposeObject, 72

xeniumCells, 73