Package ‘BayesSpace’

January 26, 2026

Version 1.21.2

Date 2026-01-08

Title Clustering and Resolution Enhancement of Spatial Transcriptomes

Description Tools for clustering and enhancing the resolution of spatial gene expression experiments. BayesSpace clusters a low-dimensional representation of the gene expression matrix, incorporating a spatial prior to encourage neighboring spots to cluster together. The method can enhance the resolution of the low-dimensional representation into ``sub-spots'', for which features such as gene expression or cell type composition can be imputed.

Depends R (>= 4.0.0), SingleCellExperiment

Imports Rcpp (>= 1.0.4.6), stats, methods, purrr, scater, scran, SummarizedExperiment, coda, rhdf5, S4Vectors, Matrix, magrittr, assertthat, arrow, mclust, RCurl, DirichletReg, xgboost (>= 3.0.0), utils, dplyr, rlang, ggplot2, tibble, rjson, tidyr, scales, microbenchmark, BiocFileCache, BiocSingular, BiocParallel

License MIT + file LICENSE

RoxygenNote 7.3.3

LinkingTo Rcpp, RcppArmadillo, RcppDist, RcppProgress

NeedsCompilation yes

SystemRequirements C++17

Encoding UTF-8

Suggests testthat, knitr, rmarkdown, igraph, spatialLIBD, viridis, patchwork, RColorBrewer, Seurat

VignetteBuilder knitr

biocViews Software, Clustering, Transcriptomics, GeneExpression, SingleCell, ImmunoOncology, DataImport

BugReports https://github.com/edward130603/BayesSpace/issues

URL edward130603.github.io/BayesSpace

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

git_branch devel

git_last_commit ba8b422

git_last_commit_date 2026-01-08

Repository Bioconductor 3.23

Date/Publication 2026-01-26

Author Edward Zhao [aut],
Senbai Kang [aut, cre],
Matt Stone [aut],
Xing Ren [ctb],
Raphael Gottardo [ctb]

Maintainer Senbai Kang <senbai.kang@chuv.ch>

Contents

`.adjust_hex_centers` *Adjust hex spot positions so hexagons are adjacent to each other in plot*

Description

Spots are regular hexagons with one unit of horizontal distance between centers

Usage

```
.adjust_hex_centers(spot_positions)
```

Value

Shifted spot centers

`.bsData` *Access BayesSpace metadata*

Description

Access BayesSpace metadata

Usage

```
.bsData(sce, name, default = NULL, warn = FALSE)
```

Arguments

`sce`	SingleCellExperiment
`name`	Metadata name

Value

Requested metadata

`.clean_chain` *Tidy C++ outputs before writing to disk.*

Description

1) Convert each parameter to matrix (n_iterations x n_indices) 2) Add appropriate colnames 3) Thin evenly (for enhance)

Usage

```
.clean_chain(out, method = c("cluster", "enhance"))
```

Arguments

`out`	List returned by `cluster()` or `deconvolve()`.
`method`	Whether the output came from clustering or enhancement. (Different params are included in each.)

Value

List with standardized parameters

`.compute_interspot_distances` *Estimate the distance between two neighboring spots*

Description

Fit linear models between each image pixel coordinate and its corresponding array coordinate to estimate the pixel distance between two spots along each axis. Add these distances to estimate the L1 distance between two spots, then add a small buffer.

Usage

```
.compute_interspot_distances(sce)
```

Arguments

`sce`	SingleCellExperiment (must include array_row, array_col, pxl_row_in_fullres, pxl_col_in_fullres in colData)

Value

doubles xdist, ydist

`.extract_indices` *Extract row and column indices of the count matrix from h5 file.*

Description

Extract row and column indices of the count matrix from h5 file.

Usage

```
.extract_indices(idx, new.start, zero.based = TRUE)
```

Arguments

`idx`	Row index of corresponding element in the non-zero count matrix.
`new.start`	Index of the start of each column corresponding to `idx` and the non-zero count matrix.
`zero.based`	Whether the and are zero-based or not. (By default is TRUE)

Value

List of row (i) and column (j) indices of the non-zero elements in the count matrix.

`.find_neighbors` *Find neighboring spots based on array coordinates*

Description

Find neighboring spots based on array coordinates

Usage

```
.find_neighbors(sce, platform)
```

Arguments

`sce`	SingleCellExperiment
`platform`	If "Visium", select six neighboring spots around center; if "ST", select four adjacent spots.

Value

`df_j` a list of neighbor indices (zero-indexed) for each spot

`.flatten_matrix_list` *Convert a list of matrices to a single matrix, where each row is a flattened matrix from the original list*

Description

Convert a list of matrices to a single matrix, where each row is a flattened matrix from the original list

Usage

```
.flatten_matrix_list(xs, ...)
```

Arguments

`xs`	List of matrices

Value

Matrix

`.flip_axis` *Whether to flip x and y axis to align the plot with the corresponding image.*

Description

Whether to flip x and y axis to align the plot with the corresponding image.

Usage

```
.flip_axis(sce, platform)
```

Value

A list indicates the multiplier for each axis.

`.infer_param_dims` *Infer original dimensions of parameter (per iteration) from colnames*

Description

Used to avoid writing colnames directly to HDF5 as attribute, which fails for large parameters (e.g. Y)

Usage

```
.infer_param_dims(cnames)
```

Arguments

`cnames`	List of column names

Value

Numeric vector (nrow, ncol)

`.init_cluster` *Initialize cluster assignments*

Description

Initialize cluster assignments

Usage

```
.init_cluster(Y, q, init = NULL, init.method = c("mclust", "kmeans"))
```

Arguments

`Y`	Representation of reduced dimensions
`q`	Number of clusters
`init`	Vector of initial cluster assignments
`init.method`	Initialization clustering algorithm

Value

Vector of cluster assignments.

`.list2vec` *Convert a list into vectors for easier output.*

Description

Convert a list into vectors for easier output.

Usage

```
.list2vec(X, sep = "=", collapse = ",", use_names = TRUE)
```

Arguments

`X`	A list.

Value

A vector converted from the input list `X`.

`.make_hex_spots` *Make vertices for each hex spot*

Description

Make vertices for each hex spot

Usage

```
.make_hex_spots(cdata, fill, coord.multiplier = list(x = 1, y = 1))
```

Value

Table of (x.pos, y.pos, spot, fill); where `spot` groups the vertices outlining the spot's border

`.make_index_names` *Make colnames for parameter indices.*

Description

Scalar parameters are named `"name"`. Vector parameters are named `"name[i]"`. Matrix parameters are named `"name[i,j]"`.

Usage

```
.make_index_names(name, m = NULL, n = NULL, dim = 1)
```

Arguments

`name`	Parameter name
`m, n`	Dimensions of parameter (m=nrow, n=ncol)
`dim`	Dimensionality of parameter (0=scalar, 1=vector, 2=matrix)

Value

List of names for parameter values

`.make_spot_vertices` *Compute vertex coordinates for each spot in frame of plot*

Description

Compute vertex coordinates for each spot in frame of plot

Usage

```
.make_spot_vertices(spot_positions, vertex_offsets)
```

Arguments

`spot_positions`	Center for hex, top left for square
`vertex_offsets`	Data frame of (x, y) offsets wrt spot position for each vertex of spot

Value

Cartesian product of positions and offsets, with coordinates computed as (pos + offset)

`.make_square_spots` *Make vertices for each square spot*

Description

Squares are simple, just make a unit square at each array coordinate

Usage

```
.make_square_spots(
  cdata,
  fill = "spatial.cluster",
  scale.factor = 1,
  offset = 0,
  coord.multiplier = list(x = 1, y = 1)
)
```

Value

Table of (x.pos, y.pos, spot, fill); where `spot` groups the vertices outlining the spot's border

`.make_subspots` *Define offsets and Manhattan distances for each subspot layout.*

Description

Hex spots are divided into 6 triangular subspots, square spots are divided into 9 squares. Offsets are relative to the spot center. A unit corresponds to the diameter of a spot.

Usage

```
.make_subspots(
  platform,
  xdist,
  ydist,
  force = FALSE,
  nsubspots.per.edge = 3,
  tolerance = 1.05
)
```

Details

Manhattan distance is used here instead of Euclidean to avoid numerical issues.

`.make_subspot_coldata` *Add subspot labels and offset row/col locations before making enhanced SCE.*

Description

Subspots are stored as (1.1, 2.1, 3.1, ..., 1.2, 2.2, 3.2, ...)

Usage

```
.make_subspot_coldata(
  cdata,
  sce,
  subspot_neighbors,
  platform,
  nsubspots.per.edge = 3
)
```

Arguments

`cdata`	Table of colData (imagerow and imagecol; from deconv$positions)
`sce`	Original sce (to obtain number of spots and original row/col)
`subspot_neighbors`	Neighbors for subspots
`platform`	Spatial transcriptomic platform
`nsubspots.per.edge`	Number of subspots per edge if the spot is squared

Value

Data frame with added subspot names, parent spot indices, and offset row/column coordinates

`.make_triangle_subspots` *Make vertices for each triangle subspot of a hex*

Description

Make vertices for each triangle subspot of a hex

Usage

```
.make_triangle_subspots(
  cdata,
  fill = "spatial.cluster",
  coord.multiplier = list(x = 1, y = 1)
)
```

Value

Table of (x.pos, y.pos, spot, fill); where `spot` groups the vertices outlining the spot's border

---

`.make_vertices` *Make vertices outlining spots/subspots for geom_polygon()*

Description

Make vertices outlining spots/subspots for geom_polygon()

Usage

```
.make_vertices(sce, fill, platform, is.enhanced, nsubspots.per.edge = 3)
```

Arguments

`sce`	SingleCellExperiment with row/col in colData
`fill`	Name of a column in `colData(sce)` or a vector of values to use as fill for each spot
`platform`	"Visium", "VisiumHD" or"ST", used to determine spot layout
`is.enhanced`	If true, `sce` contains enhanced subspot data instead of spot-level expression. Used to determine spot layout.

Value

Table of (x.pos, y.pos, spot, fill); where `spot` groups the vertices outlining the spot's border

---

`.prepare_inputs` *Prepare cluster/deconvolve inputs from SingleCellExperiment object*

Description

Prepare cluster/deconvolve inputs from SingleCellExperiment object

Usage

```
.prepare_inputs(
  sce,
  use.dimred = "PCA",
  d = 15,
  positions = NULL,
  position.cols = c("pxl_col_in_fullres", "pxl_row_in_fullres"),
  xdist = NULL,
  ydist = NULL
)
```

Value

List of PCs, names of columns with x/y positions, and inter-spot distances

---

`.read_chain` *Load saved chain from disk.*

Description

Load saved chain from disk.

Usage

```
.read_chain(h5.fname, params = NULL, is.enhanced = FALSE)
```

Arguments

`h5.fname`	Path to hdf5 file containing chain
`params`	List of parameters to read from file (will read all by default)

Value

MCMC chain, represented as a `coda::mcmc` object

`.read_spot_pos` *Load spot positions.*

Description

Load spot positions.

Usage

```
.read_spot_pos(dirname, barcodes = NULL)
```

Arguments

`dirname`	Path to spaceranger outputs of spatial pipeline, i.e., "outs/spatial". This directory must contain a file for the spot positions at `tissue_positions_list.csv` (before Space Ranger V2.0) or `tissue_positions.csv` (since Space Ranger V2.0).

Value

Data frame of spot positions.

.select_spot_positions

Helper to extract x, y, fill ID from colData

Description

Helper to extract x, y, fill ID from colData

Usage

```
.select_spot_positions(
  cdata,
  x = "array_col",
  y = "array_row",
  fill = "spatial.cluster"
)
```

Value

Dataframe of (x.pos, y.pos, fill) for each spot

.select_subspot_positions

Helper to pull out subspot position columns Probably redundant with select_spot_positions above, but we need subspot.idx

Description

Helper to pull out subspot position columns Probably redundant with select_spot_positions above, but we need subspot.idx

Usage

```
.select_subspot_positions(
  cdata,
  x = "spot.col",
  y = "spot.row",
  fill = "spatial.cluster"
)
```

Value

Dataframe of (x.pos, y.pos, fill) for each spot

`BayesSpace` *BayesSpace: A package for processing spatial transcriptomes*

Description

Tools for clustering and enhancing the resolution of spatial gene expression experiments. BayesSpace clusters a low-dimensional representation of the gene expression matrix, incorporating a spatial prior to encourage neighboring spots to cluster together. The method can enhance the resolution of the low-dimensional representation into "sub-spots", for which features such as gene expression or cell type composition can be imputed.

Details

For an overview of the functionality provided by the package, please see the vignette: `vignette("BayesSpace", package="BayesSpace")`

Author(s)

Maintainer: Senbai Kang <`senbai.kang@chuv.ch`>

Authors:

- Edward Zhao <`edward130603@gmail.com`>
- Matt Stone <`mstone@fredhutch.org`>

Other contributors:

- Xing Ren <`xren2@fredhutch.org`> [contributor]
- Raphael Gottardo <`rgottard@fredhutch.org`> [contributor]

See Also

Useful links:

- `edward130603.github.io/BayesSpace`
- Report bugs at `https://github.com/edward130603/BayesSpace/issues`

cluster — *Wrapper around C++ `iterate_*()` functions*

Description

Wrapper around C++ `iterate_*()` functions

Usage

```
cluster(
  Y,
  q,
  df_j,
  init = rep(1, nrow(Y)),
  model = c("t", "normal"),
  precision = c("equal", "variable"),
  mu0 = colMeans(Y),
  lambda0 = diag(0.01, nrow = ncol(Y)),
  gamma = 3,
  alpha = 1,
  beta = 0.01,
  nrep = 1000,
  thin = 100
)
```

Value

List of clustering parameter values at each iteration

clusterPlot — *Plot spatial cluster assignments.*

Description

Plot spatial cluster assignments.

Usage

```
clusterPlot(
  sce,
  label = "spatial.cluster",
  palette = NULL,
  color = NULL,
  platform = NULL,
  is.enhanced = NULL,
  nsubspots.per.edge = 3,
  ...
)
```

Arguments

`sce`	SingleCellExperiment. If `fill` is specified and is a string, it must exist as a column in `colData(sce)`.
`label`	Labels used to color each spot. May be the name of a column in `colData(sce)`, or a vector of discrete values.
`palette`	Optional vector of hex codes to use for discrete spot values.
`color`	Optional hex code to set color of borders around spots. Set to `NA` to remove borders.
`platform`	Spatial sequencing platform. If "Visium", the hex spot layout will be used, otherwise square spots will be plotted. NOTE: specifying this argument is only necessary if `sce` was not created by `spatialCluster()` or `spatialEnhance()`.
`is.enhanced`	True if `sce` contains subspot-level data instead of spots. Spatial sequencing platform. If true, the respective subspot lattice for each platform will be plotted. NOTE: specifying this argument is only necessary if `sce` was not created by `spatialCluster()` or `spatialEnhance()`.
`nsubspots.per.edge`	Number of subspots per edge of the square. Only valid when `platform` is 'ST' or 'VisiumHD'.
`...`	Additional arguments for `geom_polygon()`. `size`, to specify the linewidth of these borders, is likely the most useful.

Value

Returns a ggplot object.

See Also

Other spatial plotting functions: `featurePlot()`

Examples

```
sce <- exampleSCE()
clusterPlot(sce)
```

deconvolve — *Wrapper around C++* `iterate_deconv()` *function*

Description

Wrapper around C++ `iterate_deconv()` function

Usage

```
deconvolve(
  Y,
  positions,
  xdist,
  ydist,
  scalef,
  q,
  spot_neighbors,
  init,
  nrep = 1000,
  thin = 100,
  model = "normal",
  platform = c("Visium", "VisiumHD", "ST"),
  nsubspots.per.edge = 3,
  verbose = TRUE,
  jitter.scale = 5,
  jitter.prior = 0.01,
  adapt.before = 100,
  mu0 = colMeans(Y),
  gamma = 2,
  lambda0 = diag(0.01, nrow = ncol(Y)),
  alpha = 1,
  beta = 0.01,
  cores = 1
)
```

Value

List of enhancement parameter values at each iteration

enhanceFeatures — *Predict feature vectors from enhanced PCs.*

Description

Predict feature vectors from enhanced PCs.

Usage

```
enhanceFeatures(
  sce.enhanced,
  sce.ref,
  feature_names = NULL,
  model = c("xgboost", "dirichlet", "lm"),
  use.dimred = "PCA",
  assay.type = "logcounts",
  altExp.type = NULL,
  feature.matrix = NULL,
  nrounds = 0,
  train.n = round(ncol(sce.ref) * 2/3),
  nthread = 1L
)
```

Arguments

`sce.enhanced`	SingleCellExperiment object with enhanced PCs.
`sce.ref`	SingleCellExperiment object with original PCs and expression.
`feature_names`	List of genes/features to predict expression/values for.
`model`	Model used to predict enhanced values.
`use.dimred`	Name of dimension reduction to use.
`assay.type`	Expression matrix in `assays(sce.ref)` to predict.
`altExp.type`	Expression matrix in `altExps(sce.ref)` to predict. Overrides `assay.type` if specified.
`feature.matrix`	Expression/feature matrix to predict, if not directly attached to `sce.ref`. Must have columns corresponding to the spots in `sce.ref`. Overrides `assay.type` and `altExp.type` if specified.
`nrounds`	Nonnegative integer to set the `nrounds` parameter (max number of boosting iterations) for xgboost. `nrounds = 100` works reasonably well in most cases. If `nrounds` is set to 0, the parameter will be tuned using a train-test split. We recommend tuning `nrounds` for improved feature prediction, but note this will increase runtime.
`train.n`	Number of spots to use in the training dataset for tuning nrounds. By default, 2/3 the total number of spots are used.
`nthread`	Number of threads to use for xgboost. By default, 1 thread is used.

Details

Enhanced features are computed by fitting a predictive model to a low-dimensional representation of the original expression vectors. By default, a linear model is fit for each gene using the top 15 principal components from each spot, i.e. `lm(gene ~ PCs)`, and the fitted model is used to predict the enhanced expression for each gene from the subspots' principal components.

Diagnostic measures, such as RMSE for `xgboost` or R.squared for linear regression, are added to the 'rowData' of the enhanced experiment if the features are an assay of the original experiment. Otherwise they are stored as an attribute of the returned matrix/altExp.

Note that feature matrices will be returned and are expected to be input as $p \times n$ matrices of p-dimensional feature vectors over the n spots.

Value

If `assay.type` or `altExp.type` are specified, the enhanced features are stored in the corresponding slot of `sce.enhanced` and the modified SingleCellExperiment object is returned.

If `feature.matrix` is specified, or if a subset of features are requested, the enhanced features are returned directly as a matrix.

Examples

```
set.seed(149)
sce <- exampleSCE()
sce <- spatialCluster(sce, 7, nrep = 100, burn.in = 10)
enhanced <- spatialEnhance(sce, 7, init = sce$spatial.cluster, nrep = 100, burn.in = 10)
enhanced <- enhanceFeatures(enhanced, sce, feature_names = c("gene_1", "gene_2"))
```

`exampleSCE` *Create minimal* `SingleCellExperiment` *for documentation examples.*

Description

Create minimal `SingleCellExperiment` for documentation examples.

Usage

```
exampleSCE(nrow = 8, ncol = 12, n_genes = 100, n_PCs = 10)
```

Arguments

`nrow`	Number of rows of spots
`ncol`	Number of columns of spots
`n_genes`	Number of genes to simulate
`n_PCs`	Number of principal components to include

Details

Inspired by scuttle's `mockSCE()`.

Value

A SingleCellExperiment object with simulated counts, corresponding logcounts and PCs, and positional data in `colData`. Spots are distributed over an (`nrow` x `ncol`) rectangle.

Examples

```
set.seed(149)
sce <- exampleSCE()
```

`featurePlot` *Plot spatial gene expression.*

Description

Plot spatial gene expression.

Usage

```
featurePlot(
  sce,
  feature,
  assay.type = "logcounts",
  diverging = FALSE,
  low = NULL,
  high = NULL,
  mid = NULL,
  color = NULL,
  platform = NULL,
  is.enhanced = NULL,
  nsubspots.per.edge = 3,
  ...
)
```

Arguments

`sce`	SingleCellExperiment. If `feature` is specified and is a string, it must exist as a row in the specified assay of `sce`.
`feature`	Feature vector used to color each spot. May be the name of a gene/row in an assay of `sce`, or a vector of continuous values.
`assay.type`	String indicating which assay in `sce` the expression vector should be taken from.
`diverging`	If true, use a diverging color gradient in `featurePlot()` (e.g. when plotting a fold change) instead of a sequential gradient (e.g. when plotting expression).
`low, mid, high`	Optional hex codes for low, mid, and high values of the color gradient used for continuous spot values.
`color`	Optional hex code to set color of borders around spots. Set to `NA` to remove borders.
`platform`	Spatial sequencing platform. If "Visium", the hex spot layout will be used, otherwise square spots will be plotted. NOTE: specifying this argument is only necessary if `sce` was not created by `spatialCluster()` or `spatialEnhance()`.

`is.enhanced`	True if `sce` contains subspot-level data instead of spots. Spatial sequencing platform. If true, the respective subspot lattice for each platform will be plotted. NOTE: specifying this argument is only necessary if `sce` was not created by `spatialCluster()` or `spatialEnhance()`.
`nsubspots.per.edge`	Number of subspots per edge of the square. Only valid when `platform` is 'ST' or 'VisiumHD'.
`...`	Additional arguments for `geom_polygon()`. `size`, to specify the linewidth of these borders, is likely the most useful.

Value

Returns a ggplot object.

See Also

Other spatial plotting functions: `clusterPlot()`

Examples

```
sce <- exampleSCE()
featurePlot(sce, "gene_2")
```

`find_neighbors` *Compute pairwise distances between all spots and return list of neighbors for each spot.*

Description

Compute pairwise distances between all spots and return list of neighbors for each spot.

Usage

```
find_neighbors(positions, radius, method = c("manhattan", "euclidean"))
```

Arguments

`positions`	(n x 2) matrix of spot coordinates.
`radius`	The maximum distance for two spots to be considered neighbors.
`method`	Distance metric to use.

Value

List df_j, where `df_j[[i]]` is a vector of zero-indexed neighbors of i.

getRDS — *Download a processed sample from our S3 bucket*

Description

Datasets are cached locally using `BiocFileCache`. The first time using this function, you may need to consent to creating a BiocFileCache directory if one does not already exist.

Usage

```
getRDS(dataset, sample, cache = TRUE)
```

Arguments

`dataset`	Dataset identifier
`sample`	Sample identifier
`cache`	If true, cache the dataset locally with `BiocFileCache`. Otherwise, download directly from our S3 bucket. Caching saves time on subsequent loads, but consumes disk space.

Details

The following datasets are available via `getRDS`.

Dataset	Sample(s)
2018_thrane_melanoma	ST_mel1_rep2
2020_maynard_prefrontal-cortex	151507, 151508, 151509, 151510, 151669, 151670, 151671, 151672, 151673, 151674, 1
2020_ji_squamous-cell-carcinoma	P4_rep1
2020_10X-IDC	IDC1
2020_10X-demo_ovarian-cancer	whole_transcriptome

Value

sce A SingleCellExperiment with positional information in colData and PCs based on the top 2000 HVGs

Examples

```
sce <- getRDS("2018_thrane_melanoma", "ST_mel1_rep2", cache = FALSE)
```

`mcmcChain` *Read MCMC chain associated with a BayesSpace clustering or enhancement*

Description

BayesSpace stores the MCMC chain associated with a clustering or enhancement on disk in an HDF5 file. The `mcmcChain()` function reads any parameters specified by the user into a `coda::mcmc` object compatible with TidyBayes.

Usage

```
mcmcChain(sce, params = NULL)

removeChain(sce)
```

Arguments

`sce`	SingleCellExperiment with a file path stored in its metadata.
`params`	List of model parameters to read

Details

To interact with the HDF5 file directly, obtain the filename from the SingleCellExperiment's metadata: `metadata(sce)$chain.h5`. Each parameter is stored as a separate dataset in the file, and is represented as a matrix of size (n_iterations x n_parameter_indices). Parameter choices for the spot-level clustering include:

- `z` (cluster assignments)
- `weights` (w_i)
- `mu` (mean vectors)
- `lambda` (precision matrix)
- `plogLik` (pseudo-log-likelihood)

Parameter choices for the subspot-level enhanced clustering include:

- `z` (cluster assignments)
- `weights` (w_i)
- `Y` (enhanced PCs)
- `mu` (mean vectors)
- `lambda` (precision matrix)
- `Ychange` (acceptance rate for the jittering of PCs)

For best results, `Ychange` should average between 0.25 and 0.40.

Value

Returns an mcmc object containing the values of the requested parameters over the constructed chain.

Examples

```
set.seed(149)
sce <- exampleSCE()
sce <- spatialCluster(sce, 7, nrep=100, burn.in=10, save.chain=TRUE)
chain <- mcmcChain(sce)
removeChain(sce)
```

`Mode` *Find the mode*

Description

Used for finding the most frequent cluster for each z

Usage

```
Mode(x)
```

Arguments

x	Numeric vector

Value

mode Numeric scalar, most frequent element in x

`parallelize` *Parallelization*

Description

A convenient wrapper function of `BiocParallel` providing easy parallelization.

Usage

```
paraLapply(
  X,
  FUN,
  BPPARAM = NULL,
  cores = 1L,
  type = c("serial", "fork", "sock"),
  verbose = FALSE,
  ...
)
```

Arguments

X	Any object for which methods `length`, `[`, and `[[` are implemented (passed to `bplapply`).
FUN	The `function` to be applied to each element of X (passed to `bplapply`).
BPPARAM	An optional `BiocParallelParam` instance determining the parallel back-end to be used during evaluation, or a list of `BiocParallelParam` instances, to be applied in sequence for nested calls to `BiocParallel` functions.
cores	The number of threads to use. The results are invariate to the value of `cores`.
type	One of "serial", "fork", or "sock". When `cores` is one, `type` is always "serial". Both "fork" and "sock" are for multi-threading. "fork" is faster, but only supports linux and macos. "sock" supports linux, macos, and windows.
verbose	Whether to print debug information or not.
...	Additional parameters passed to `bplapply`.

Value

See `lapply`.

---

qTune *Tuning the choice of q (number of clusters) before running spatialCluster*

Description

Before running `spatialCluster()`, we recommend tuning the choice of `q` by choosing the `q` that minimizes the model's negative log likelihood over early iterations. `qTune()` computes the average negative log likelihood for a range of q values over iterations 100:1000, and `qPlot()` displays the results.

Usage

```
qPlot(sce, qs = seq(3, 7), force.retune = FALSE, ...)

qTune(sce, qs = seq(3, 7), burn.in = 100, nrep = 1000, cores = 1L, ...)
```

Arguments

sce	A SingleCellExperiment object containing the spatial data.
qs	The values of q to evaluate.
force.retune	If specified, existing tuning values in `sce` will be overwritten.
...	Other parameters are passed to `spatialCluster()`.
burn.in, nrep	Integers specifying the range of repetitions to compute.
cores	The number of threads to use. The results are invariate to the value of `cores`.

Details

`qTune()` takes the same parameters as `spatialCluster()` and will run the MCMC clustering algorithm up to `nrep` iterations for each value of `q`. The first `burn.in` iterations are discarded as burn-in and the log likelihood is averaged over the remaining iterations.

`qPlot()` plots the computed negative log likelihoods as a function of q. If `qTune()` was run previously, i.e. there exists an attribute of `sce` named `"q.logliks"`, the pre-computed results are displayed. Otherwise, or if `force.retune` is specified, `qplot()` will automatically run `qTune()` before plotting (and can take the same parameters as `spatialCluster()`.

Value

`qTune()` returns a modified `sce` with tuning log likelihoods stored as an attribute named `"q.logliks"`.

`qPlot()` returns a ggplot object.

Examples

```
set.seed(149)
sce <- exampleSCE()
sce <- qTune(sce, seq(3, 7), burn.in = 10, nrep = 100)
qPlot(sce)
```

`readVisium` *Load a Visium spatial dataset as a SingleCellExperiment.*

Description

Load a Visium spatial dataset as a SingleCellExperiment.

Usage

```
readVisium(
  dirname,
  rm.feats.pat = c("^NegControl.*", "^BLANK.*", "^DEPRECATED.*")
)

read10Xh5(
  dirname,
  fname = "filtered_feature_bc_matrix.h5",
  rm.feats.pat = c("^NegControl.*", "^BLANK.*", "^DEPRECATED.*")
)

counts2h5(dirname)
```

Arguments

`dirname`	Path to spaceranger output directory (e.g. "sampleID/outs/"). This directory must contain the counts matrix and feature/barcode TSVs in `filtered_feature_bc_matrix/` for `readVisium`, or in `filtered_feature_bc_matrix.h5` for `read10Xh5`. Besides, it must also contain a file for spot positions named `spatial/tissue_positions_list.csv` (before Space Ranger V2.0) or `spatial/tissue_positions.csv` (since Space Ranger V2.0), as well as a file containing scale factors named `spatial/scalefactors_json.json`. (To understand the output directory, refer to the corresponding 10X Genomics help page.)
`rm.feats.pat`	Patterns for features (genes) to remove.
`fname`	File name of the h5 file. It should be inside `dirname`. (By default "filtered_feature_bc_matrix.h5")

Details

We store two variables associated with downstream BayesSpace functions in a list called `BayesSpace.data` in the SingleCellExperiment's `metadata`.

- `platform` is set to "Visium", and is used to determine spot layout and neighborhood structure.
- `is.enhanced` is set to `FALSE` to denote the object contains spot-level data.

Value

SingleCellExperiment containing the counts matrix in `counts` and spatial data in `colData`. Array coordinates for each spot are stored in columns `array_row` and `array_col`, while image coordinates are stored in columns `pxl_row_in_fullres` and `pxl_col_in_fullres`.

Examples

```
## Not run:
sce <- readVisium("path/to/outs/")

## End(Not run)
```

`spatialCluster` *Spatial clustering*

Description

Cluster a spatial expression dataset.

Usage

```
spatialCluster(
  sce,
  q,
  use.dimred = "PCA",
  d = 15,
  platform = c("Visium", "VisiumHD", "ST"),
  init = NULL,
  init.method = c("mclust", "kmeans"),
  model = c("t", "normal"),
  precision = c("equal", "variable"),
  nrep = 50000,
  burn.in = 1000,
  thin = 100,
  gamma = NULL,
  mu0 = NULL,
  lambda0 = NULL,
  alpha = 1,
  beta = 0.01,
  save.chain = FALSE,
  chain.fname = NULL
)
```

Arguments

`sce`	A SingleCellExperiment object containing the spatial data.
`q`	The number of clusters.
`use.dimred`	Name of a reduced dimensionality result in `reducedDims(sce)`. If provided, cluster on these features directly.
`d`	Number of top principal components to use when clustering.
`platform`	Spatial transcriptomic platform. Specify 'Visium' for hex lattice geometry or 'ST' and 'VisiumHD' for square lattice geometry. Specifying this parameter is optional when analyzing SingleCellExperiments processed using `readVisium`, `spatialPreprocess`, or `spatialCluster`, as this information is included in their metadata.
`init`	Initial cluster assignments for spots.
`init.method`	If `init` is not provided, cluster the top `d` PCs with this method to obtain initial cluster assignments.
`model`	Error model. ('normal' or 't')
`precision`	Covariance structure. ('equal' or 'variable' for EEE and VVV covariance models, respectively.)
`nrep`	The number of MCMC iterations.
`burn.in`	The number of MCMC iterations to exclude as burn-in period.
`thin`	Thinning rate.

`gamma`	Smoothing parameter. Defaults to 2 for `platform="ST"` and 3 for `platform="Visium"`. (Values in range of 1-3 seem to work well.)
`mu0`	Prior mean hyperparameter for mu. If not provided, mu0 is set to the mean of PCs over all spots.
`lambda0`	Prior precision hyperparam for mu. If not provided, lambda0 is set to a diagonal matrix $0.01I$.
`alpha`	Hyperparameter for Wishart distributed precision lambda.
`beta`	Hyperparameter for Wishart distributed precision lambda.
`save.chain`	If true, save the MCMC chain to an HDF5 file.
`chain.fname`	File path for saved chain. Tempfile used if not provided.

Details

The input SCE must have `row` and `col` columns in its `colData`, corresponding to the array row and column coordinates of each spot. These are automatically parsed by `readVisium` or can be added manually when creating the SCE.

Cluster labels are stored in the `spatial.cluster` column of the SCE, and the cluster initialization is stored in `cluster.init`.

Value

Returns a modified `sce` with cluster assignments stored in `colData` under the name `spatial.cluster`.

See Also

`spatialPreprocess` for preparing the SCE for clustering, `spatialEnhance` for enhancing the clustering resolution, `clusterPlot` for visualizing the cluster assignments, `featurePlot` for visualizing expression levels in spatial context, and `mcmcChain` for examining the full MCMC chain associated with the clustering.

Examples

```
set.seed(149)
sce <- exampleSCE()
sce <- spatialCluster(sce, 7, nrep = 100, burn.in = 10)
```

`spatialEnhance` *Enhance spot resolution*

Description

Enhanced clustering of a spatial expression dataset to subspot resolution.

Usage

```
spatialEnhance(
  sce,
  q,
  platform = c("Visium", "VisiumHD", "ST"),
  use.dimred = "PCA",
  d = 15,
  nsubspots.per.edge = 3,
  init = NULL,
  init.method = c("spatialCluster", "mclust", "kmeans"),
  model = c("t", "normal"),
  nrep = 1e+05,
  gamma = NULL,
  mu0 = NULL,
  lambda0 = NULL,
  alpha = 1,
  beta = 0.01,
  save.chain = FALSE,
  chain.fname = NULL,
  burn.in = 10000,
  thin = 100,
  jitter.scale = 5,
  jitter.prior = 0.3,
  adapt.before = burn.in,
  cores = 1,
  verbose = FALSE
)

coreTune(sce, test.cores = detectCores(), test.times = 1, ...)

adjustClusterLabels(sce, burn.in)
```

Arguments

`sce`	A SingleCellExperiment object containing the spatial data.
`q`	The number of clusters.
`platform`	Spatial transcriptomic platform. Specify 'Visium' for hex lattice geometry or 'ST' and 'VisiumHD' for square lattice geometry. Specifying this parameter is optional when analyzing SingleCellExperiments processed using `readVisium`, `spatialPreprocess`, or `spatialCluster`, as this information is included in their metadata.
`use.dimred`	Name of a reduced dimensionality result in `reducedDims(sce)`. If provided, cluster on these features directly.
`d`	Number of top principal components to use when clustering.
`nsubspots.per.edge`	Number of subspots per edge of the square. Only valid when `platform` is 'ST' or 'VisiumHD'.

`init`	Initial cluster assignments for spots.
`init.method`	If `init` is not provided, cluster the top `d` PCs with this method to obtain initial cluster assignments.
`model`	Error model. ('normal' or 't')
`nrep`	The number of MCMC iterations.
`gamma`	Smoothing parameter. (Values in range of 1-3 seem to work well.)
`mu0`	Prior mean hyperparameter for mu. If not provided, mu0 is set to the mean of PCs over all spots.
`lambda0`	Prior precision hyperparam for mu. If not provided, lambda0 is set to a diagonal matrix $0.01I$.
`alpha`	Hyperparameter for Wishart distributed precision lambda.
`beta`	Hyperparameter for Wishart distributed precision lambda.
`save.chain`	If true, save the MCMC chain to an HDF5 file.
`chain.fname`	File path for saved chain. Tempfile used if not provided.
`burn.in`	Number of iterations to exclude as burn-in period. The MCMC iterations are currently thinned to every 100; accordingly `burn.in` is rounded down to the nearest multiple of 100. If a value no larger than 1 is set, it is considered as a percentage. It is always considered as percentage for `adjustClusterLabels`.
`thin`	Thinning rate.
`jitter.scale`	Controls the amount of jittering. Small amounts of jittering are more likely to be accepted but result in exploring the space more slowly. We suggest tuning `jitter.scale` so that Ychange is on average around 25%-40%. Ychange can be accessed via `mcmcChain()`. Alternatively, set it to 0 to activate adaptive MCMC.
`jitter.prior`	Scale factor for the prior variance, parameterized as the proportion (default = 0.3) of the mean variance of the PCs. We suggest making `jitter.prior` smaller if the jittered values are not expected to vary much from the overall mean of the spot.
`adapt.before`	Adapting the MCMC chain before the specified number or proportion of iterations (by default equal to `burn.in`; set to 0 to always adapt). Only valid when `jitter.scale` is 0.
`cores`	The number of threads to use. The results are invariate to the value of `cores`.
`verbose`	Log progress to stderr.
`test.cores`	Either a list of, or a maximum number of cores to test. In the latter case, a list of values (power of 2) will be created
`test.times`	Times to repeat the benchmarking with microbenchmark.
`...`	Arguments for `spatialEnhance` (except for cores).

Details

The enhanced `SingleCellExperiment` has most of the properties of the input SCE - `rowData`, `colData`, `reducedDims` - but does not include expression data in `counts` or `logcounts`. To impute enhanced expression vectors, please use [enhanceFeatures()] after running `spatialEnhance`.

The `colData` of the enhanced `SingleCellExperiment` includes the following columns to permit referencing the subspots in spatial context and linking back to the original spots:

- `spot.idx`: Index of the spot this subspot belongs to (with respect to the input SCE).
- `subspot.idx`: Index of the subspot within its parent spot.
- `spot.row`: Array row of the subspot's parent spot.
- `spot.col`: Array col of the subspot's parent spot.
- `array_row`: Array row of the subspot. This is the parent spot's row plus an offset based on the subspot's position within the spot.
- `array_col`: Array col of the subspot. This is the parent spot's col plus an offset based on the subspot's position within the spot.
- `pxl_row_in_fullres`: Pixel row of the subspot. This is the parent spot's row plus an offset based on the subspot's position within the spot.
- `pxl_col_in_fullres`: Pixel col of the subspot. This is the parent spot's col plus an offset based on the subspot's position within the spot.

Value

`spatialEnhance` returns a new SingleCellExperiment object. By default, the `assays` of this object are empty, and the enhanced resolution PCs are stored as a reduced dimensionality result accessible with `reducedDim(sce, 'PCA')`.

`coresTune` returns the output of `microbenchmark`.

`adjustClusterLabels` adjusts the cluster labels from the MCMC samples via `burn.in`, the percentage of samples to drop. The MCMC chain must be retained.

See Also

`spatialCluster` for clustering at the spot level before enhancing, `clusterPlot` for visualizing the cluster assignments, `enhanceFeatures` for imputing enhanced expression, and `mcmcChain` for examining the full MCMC chain associated with the enhanced clustering. .

Examples

```
set.seed(149)
sce <- exampleSCE()
sce <- spatialCluster(sce, 7, nrep = 100, burn.in = 10)
enhanced <- spatialEnhance(sce, 7, nrep = 100, burn.in = 10)
```

`spatialPlot` *Spatial plotting functions*

Description

Spatial plotting functions

Arguments

`color`	Optional hex code to set color of borders around spots. Set to `NA` to remove borders.
`...`	Additional arguments for `geom_polygon()`. `size`, to specify the linewidth of these borders, is likely the most useful.
`platform`	Spatial sequencing platform. If "Visium", the hex spot layout will be used, otherwise square spots will be plotted. NOTE: specifying this argument is only necessary if `sce` was not created by `spatialCluster()` or `spatialEnhance()`.
`is.enhanced`	True if `sce` contains subspot-level data instead of spots. Spatial sequencing platform. If true, the respective subspot lattice for each platform will be plotted. NOTE: specifying this argument is only necessary if `sce` was not created by `spatialCluster()` or `spatialEnhance()`.
`nsubspots.per.edge`	Number of subspots per edge of the square. Only valid when `platform` is 'ST' or 'VisiumHD'.

`spatialPreprocess` *Preprocess a spatial dataset for BayesSpace*

Description

Adds metadata required for downstream analyses, and (optionally) performs PCA on log-normalized expression of top HVGs.

Usage

```
spatialPreprocess(
  sce,
  platform = c("Visium", "VisiumHD", "ST"),
  n.PCs = 15,
  n.HVGs = 2000,
  skip.PCA = FALSE,
  log.normalize = TRUE,
  assay.type = "logcounts",
  BSPARAM = ExactParam(),
  BPPARAM = SerialParam()
)
```

Arguments

`sce`	SingleCellExperiment to preprocess
`platform`	Spatial sequencing platform. Used to determine spot layout and neighborhood structure (Visium = hex, VisiumHD = square, ST = square).

`n.PCs`	Number of principal components to compute. We suggest using the top 15 PCs in most cases.
`n.HVGs`	Number of highly variable genes to run PCA upon.
`skip.PCA`	Skip PCA (if dimensionality reduction was previously computed.)
`log.normalize`	Whether to log-normalize the input data with scater. May be omitted if log-normalization previously computed.
`assay.type`	Name of assay in `sce` containing normalized counts. Leave as "logcounts" unless you explicitly pre-computed a different normalization and added it to `sce` under another assay. Note that we do not recommend running BayesSpace on PCs computed from raw counts.
`BSPARAM`	A BiocSingularParam object specifying which algorithm should be used to perform the PCA. By default, an exact PCA is performed, as current spatial datasets are generally small (<10,000 spots). To perform a faster approximate PCA, please specify `FastAutoParam()` and set a random seed to ensure reproducibility.
`BPPARAM`	A BiocParallelParam object specifying whether to model the gene variation in parallel or not (default to `SerialParam()`). To perform faster modeling, please specify `SnowParam()` or `MulticoreParam()`.

Value

SingleCellExperiment with PCA and BayesSpace metadata

Examples

```
sce <- exampleSCE()
sce <- spatialPreprocess(sce)
```

Index