Package 'SigDoTS'

November 9, 2026

Type Package

Title Hierarchical Significance Test for Spatial Domains

Version 0.2.5.10

Description Tests, from the top of a tree of candidate spatial domains downwards, whether each proposed split of spots in a spatial transcriptomics sample is significant. The null model at each split has no domain structure but keeps the spatial correlation of the noise, estimated from pairs of spots within the same candidate domain.

License MIT + file LICENSE

URL https://github.com/Yan-Lab-MSU/SigDoTS, https://yan-lab-msu.github.io/SigDoTS/

BugReports https://github.com/Yan-Lab-MSU/SigDoTS/issues

Encoding UTF-8

LazyData true

Depends R (>= 4.3.0)

Imports fields,
ggplot2,
GpGp,
grDevices,
igraph,
Matrix,
parallel,
RANN,
stats,
utils

Suggests BayesSpace,
Banksy,
knitr,
rmarkdown,
scran,
scuttle,
Seurat,
SeuratObject,
SingleCellExperiment,
SpatialExperiment,
SummarizedExperiment,
testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

R topics documented:

add_domains 2
domains 3
plot.sigdots 4
sigdots 6
sigdots_control 13
sigdots_example 16
sigdots_prepare 17
summary.sigdots 18

Index **21**

`add_domains` *Add the domains to a Seurat or SpatialExperiment object*

Description

`add_domains()` writes the result of `sigdots()` into the per-spot metadata of the object the data came from, so that the domains can be used like any other per-spot label (plotting, subsetting, differential expression). Two columns are added: `sigdots_domain`, the domain number of every spot (`NA` for a spot in a part of the tree that could not be tested), and `sigdots_candidate_domain`, the candidate domain the spot was in.

Usage

```
add_domains(object, fit, prefix = "sigdots")
```

Arguments

`object`	A Seurat object, or a SpatialExperiment, SingleCellExperiment or SummarizedExperiment object. Its cells or spots must be exactly those `fit` was computed on, in the same order (which is the case when `fit` came from `sigdots(object, ...)`).
`fit`	A result of `sigdots()`.
`prefix`	The first part of the two column names (`"sigdots"` by default, giving `sigdots_domain` and `sigdots_candidate_domain`). Use another prefix to keep the results of several runs side by side.

Details

The spots are matched by their identifiers (the column names of the object and the spot identifiers of `fit`), so the object may hold its spots in a different order, but it must not hold spots that `fit` does not have; such spots stop the function with an error naming the first of them. Existing columns with the same names are replaced.

Value

The object with the two columns added: in `object[[]]` (the metadata) for a Seurat object, in `colData(object)` for the others.

See Also

`domains()` for the same information as a plain table.

Examples

```
if (requireNamespace("SpatialExperiment", quietly = TRUE)) {
  spe <- SpatialExperiment::SpatialExperiment(
    assays = list(logcounts = t(sigdots_example$x)), spatialCoords = sigdots_example$coor
  fit <- sigdots(spe, K0 = 5, verbose = FALSE)
  spe <- add_domains(spe, fit)
  table(SummarizedExperiment::colData(spe)$sigdots_domain)
}
```

`domains` *The domain of every spot*

Description

`domains(fit)` returns the result of `sigdots()` as a table with one row per spot, in the order of the input data, so that it can be joined to any other per-spot information (the true layers, cell types, a plot). For a Seurat or SpatialExperiment object, `add_domains()` does this joining for you.

Usage

```
domains(fit)
```

Arguments

`fit`	A result of `sigdots()`.

Details

The domains are numbered by their position in the split tree, from left to right as `print(fit)` draws it, not by size. The spots of domain `k` are `domains(fit)$spot[domains(fit)$domain == k]`.

Value

A data frame with one row per spot and three columns:

- `spot`: the spot identifier (the row name of the expression matrix, or the row number when there are no row names);
- `domain`: the number of the domain the spot belongs to, 1, 2, ... in the order of the split tree from left to right (as printed by `print(fit)`). `NA` for a spot in a part of the tree that could not be tested (see `summary(fit)` and the warning given at the end of the run);
- `candidate_domain`: the candidate domain the spot was in before the tests (its number, or your own label when `candidate_labels` was given).

See Also

`summary.sigdots()` for the tests, `plot.sigdots()` for a map of the domains.

Examples

```
fit <- sigdots(sigdots_example$x, sigdots_example$coords, K0 = 5, verbose = FALSE)
head(domains(fit))
table(domains(fit)$domain)
table(domain = domains(fit)$domain, true_region = sigdots_example$region)
```

---

plot.sigdots *Plot a SigDoTS result*

Description

`plot(fit, type = ...)` draws one of four plots of a `sigdots()` result: a map of the domains, a map of the candidate domains, the split tree, or the spatial correlation estimated at a test. Each plot is a ggplot2 object, so titles, colours and themes can be changed by adding to it as usual, and it can be saved with `ggplot2::ggsave()`.

Usage

```
## S3 method for class 'sigdots'
plot(
  x,
  type = c("domains", "candidates", "tree", "correlogram"),
  test = NULL,
  point_size = NULL,
  ...
)
```

Arguments

`x`	A result of `sigdots()`.
`type`	Which plot to draw, one of:

- `"domains"` (the default): a map of the spots at their coordinates, each coloured by its domain, with the domain numbers and the boundaries between domains drawn in. Spots in a part of the tree that could not be tested are grey.
- `"candidates"`: the same map coloured by candidate domain, with the boundaries of the final domains drawn in, so that you can see which candidate domains were merged.
- `"tree"`: the split tree, as `print(x)` shows it, as a diagram: every tested part with its decision, p-value and level; the parts that became domains in the colours of `"domains"`; the candidate domains along the bottom.
- `"correlogram"`: for one test (`test`), the spatial correlation of the noise, as SigDoTS estimated it for the single-domain model of that test. The points are the correlation between pairs of spots at each distance, after removing

each candidate domain's mean and averaged over genes; the line is the correlation the fitted model gives (`eta` is the share of the noise variance that is spatially correlated, `ell` how far the correlation reaches). Distances are in units of the typical distance between neighbouring spots. The line following the points means the model describes the spatial correlation of the data.

`test` For `"correlogram"`: the node number of a tested part, as shown in brackets by `print(x)` and in the `node` column of `summary(x)$tests`. `NULL` (the default) means the first test, at the top of the tree. A part that was not tested has no correlogram and gives an error that lists the tested nodes.

`point_size` For the two maps: the size of the points, a positive number. `NULL` (the default) means `45 / sqrt(number of spots)`, kept between 0.2 and 3.

`...` Not used.

Details

The maps draw each spot at its coordinates with equal scales on the two axes, and the y axis pointing upwards. Image coordinates (such as the pixel rows of a Visium image) count downwards, so a map may appear upside down compared with the tissue image; add `ggplot2::scale_y_reverse()` to turn it. Domains are coloured in the order of their numbers with a fixed palette of eight colours (more domains use `grDevices::hcl.colors()`); the boundaries are drawn between neighbouring spots of different domains. Each domain's number is written inside the domain, not on a boundary: for every spot of the domain, the distance to the nearest spot of another domain (or untested spot) or to the edge of the tissue is measured, and among the spots where it is at least three quarters of its largest value, the number goes to the one nearest the domain's mean position. A domain in several separate pieces is numbered once. The candidate map numbers its candidate domains in the same way.

In the correlogram, points close to the line mean that the single-domain model describes the spatial correlation of the data at that test; points well above the line at short distances mean the model has less spatial correlation than the data, which can make splits easier to keep.

Value

A ggplot object; printing it draws the plot.

See Also

`summary.sigdots()` for the tests as a table, `domains()` for the domain of every spot.

Examples

```
fit <- sigdots(sigdots_example$x, sigdots_example$coords, K0 = 5, verbose = FALSE)
plot(fit)
plot(fit, type = "candidates")
plot(fit, type = "tree")
plot(fit, type = "correlogram")
plot(fit) + ggplot2::ggtitle("My title")
```

`sigdots` *Significance test for spatial domains*

Description

`sigdots()` finds the spatial domains of a spatial transcriptomics sample that are supported by the data. It starts from an over-segmentation of the sample into `K0` *candidate domains* (more than the number of domains you expect), arranges the candidate domains in a tree by spatial adjacency and expression similarity, and tests the splits of the tree from the top down, each against a model in which the part being split is a single domain. A split is kept only when it is significant; the parts of the tree that are not split further are the domains. The probability that the result contains any false split is at most `alpha`. The section "How SigDoTS works" below describes each step.

Usage

```
sigdots(
  x,
  coords = NULL,
  K0 = NULL,
  genes = NULL,
  assay = NULL,
  cluster_method = c("kmeans", "bayesspace", "banksy"),
  candidate_labels = NULL,
  cluster_args = list(),
  n_pcs = 15,
  spatial_method = c("vecchia", "exact"),
  vecchia_neighbors = 60,
  alpha = 0.05,
  pvalue_method = c("monte_carlo", "normal_adaptive"),
  B_min = 10,
  B_max = 10,
  normal_n_initial = 10,
  normal_n_refine = 50,
  n_cores = 1,
  seed = 1,
  verbose = TRUE,
  control = sigdots_control()
)
```

Arguments

`x` The data, in one of three forms.

- An **expression matrix**: one row per spot (or cell), one column per gene, with the spot identifiers as row names. The values must already be normalised, for example log(1 + 10,000 x count / total count of the spot), as `Seurat::NormalizeData()` or `scuttle::logNormCounts()` give; do not give raw counts (`sigdots_prepare()` makes the matrix from raw counts). Use the most variable genes, typically 1,000 to 3,000. A matrix, a data frame or a sparse `Matrix` is accepted. `coords` is then required.

- A **Seurat** object. The expression comes from the normalised `"data"` layer of the assay given by `assay` (by default the object's default assay), restricted to the genes in `genes` (by default the variable genes set by `Seurat::FindVariableFeatures()`). The spot positions come from the object's image (`Seurat::GetTissueCoordinates()`) or FOV unless `coords` is given.
- A **SpatialExperiment** object, or a SingleCellExperiment or SummarizedExperiment with `coords` given (for example a Scanpy `.h5ad` file read with `zellkonverter`). The expression comes from the `"logcounts"` assay (or the assay given by `assay`), restricted to the genes in `genes` (by default all genes when there are at most 5,000; otherwise `genes` is required). The positions come from `SpatialExperiment::spatialCoords()` unless `coords` is given.

The values are used as they are: genes are not scaled to variance 1, so a gene with a larger variance has more weight in the principal components and in the split statistic.

`coords` The positions of the spots: a matrix or data frame with one row per spot, in the same order as the rows of `x` (or, for an object, with the spot identifiers as row names), and two columns (x and y). Any unit can be used: SigDoTS measures distances in units of the typical distance between neighbouring spots (the median distance from a spot to its nearest other spot). Required for a matrix `x`; for an object it replaces the positions stored in the object.

`K0` The number of candidate domains, a whole number from 2 up and smaller than the number of spots. SigDoTS merges candidate domains into domains and never splits a candidate domain, so `K0` must be larger than the number of domains you expect, and large enough that every candidate domain lies inside one true domain; see "Choosing K0" below. Not needed when `candidate_labels` is given (if given as well, it must equal the number of distinct labels).

`genes` A character vector of gene names: the genes to use. For a matrix `x` these must be column names of `x`; for an object, row names of the object. `NULL` (the default) means all columns of a matrix, the variable genes of a Seurat object, or all genes of a SpatialExperiment object that has at most 5,000 genes.

`assay` For an object only: the name of the assay to take the expression from, a character string. `NULL` (the default) means the Seurat object's default assay (its `"data"` layer), or the `"logcounts"` assay of a SpatialExperiment object.

`cluster_method`

How the spots are divided into the `K0` candidate domains, and, in every test, how each simulated dataset is divided into candidate domains. One of

- `"kmeans"` (the default): k-means clustering of the first `n_pcs` principal components of `x` (10 random starts, at most 100 iterations). Fast; it groups spots by expression only, so on real tissue its candidate domains can be scattered across the section.
- `"banksy"`: the BANKSY method, which clusters each spot by its own expression together with the average expression of its neighbours and so gives spatially coherent candidate domains (k-means on the BANKSY principal components). Used in the SigDoTS paper for real tissue. Needs the Bioconductor package Banksy (R 4.4 or later).
- `"bayesspace"`: the BayesSpace method, a Bayesian model with a spatial prior designed for the regular spot arrays of Visium and ST. Needs the Bioconductor package BayesSpace; it runs once for every simulated dataset, which makes it slow.

candidate_labels

Your own candidate domains, instead of those of cluster_method: a vector with one label per spot (numbers, character strings or a factor, in the order of the spots, without missing values), or, for an object, the name of a metadata column that holds them (for example "seurat_clusters"). The tree is then built from these labels and K0 is their number of distinct values. The simulated datasets are still divided with cluster_method, so the test is calibrated for labels made by that method: when your labels come from BANKSY or BayesSpace, set cluster_method = "banksy" or "bayesspace" as well. NULL (the default) means the candidate domains are made with cluster_method.

cluster_args A named list of settings passed to BANKSY or BayesSpace (not used for k-means). For "banksy": lambda (the weight of the neighbourhood, 0.2 by default), k_geom (the number of neighbours, 15), npcs (the number of BANKSY principal components, min(n_pcs, 20)) and use_agf (FALSE). For "bayesspace": platform ("Visium" or "ST", "Visium" by default), init.method ("mclust"), model ("t"), nrep (5000 MCMC iterations), burn.in (100) and gamma (3). list() (the default) uses all these defaults.

n_pcs The number of principal components of x (a whole number, 15 by default) used by k-means to make the candidate domains, and to compute the mean of each candidate domain when the candidate domains are merged into the tree. The split statistic itself uses all genes.

spatial_method

How the spatial covariance of the simulated data is computed:

- "vecchia" (the default): the Vecchia approximation, in which each spot depends directly only on its vecchia_neighbors nearest spots. Memory and time grow in proportion to the number of spots, so it can be used for any sample size.
- "exact": the exact computation, which stores and decomposes a matrix with one row and one column per spot of the part being tested. Its memory grows with the square of the number of spots and its time with the cube; the run stops before it starts if a test would need more memory than max_exact_memory_gib of sigdots_control().

vecchia_neighbors

For spatial_method = "vecchia": the number of nearest spots each spot depends on (a whole number, 60 by default). More is closer to the exact computation and slower.

alpha The significance level for the whole tree, a number between 0 and 1 (0.05 by default): the probability that the result contains any false split is at most alpha. A test on m of the n spots uses the level alpha * (m - 1) / (n - 1).

pvalue_method

How the p-value of each test is computed from the simulated datasets:

- "monte_carlo" (the default, used for the main results of the SigDoTS paper): the p-value is (1 + the number of simulated datasets that split at least as clearly as the data) / (1 + the number of simulated datasets). It makes no assumption about the distribution of the split statistic. The number of simulated datasets is set by B_min and B_max.
- "normal_adaptive": the p-value is the probability, under a normal distribution with the mean and standard deviation of the simulated statistics, of a statistic at most as large as the observed one. Each test first simulates normal_n_initial datasets; when the p-value from these lies

between the test's level divided by `normal_band` and the level multiplied by `normal_band` (see `sigdots_control()`), it simulates more, up to `normal_n_refine` in all, and computes the p-value again from all of them. A test thus uses 10 or 50 simulated datasets with the defaults, whatever its level, whereas a Monte Carlo test needs about `1 / level` of them to keep a split; it is therefore much faster for the lower tests of a large tree. It relies on the simulated statistics having roughly a normal distribution, and on their mean and standard deviation, estimated from few datasets.

`B_min, B_max` For `pvalue_method = "monte_carlo"`: the smallest and the largest number of simulated datasets for one test (whole numbers with `1 <= B_min <= B_max`, both 10 by default).

- A test simulates at least `B_min` datasets and stops as soon as its decision can no longer change, so a split that is clearly not significant stops early.
- When `1 / (B_max + 1)` is larger than the test's level, the p-value could never reach the level; `B_max` is then raised for that test to `ceiling(1 / level) - 1` (19 at the level 0.05), up to `B_cap` of `sigdots_control()`.
- With the defaults, every test therefore uses the smallest number of simulated datasets that can establish its decision, as in the SigDoTS paper's simulations: a split is kept only when the data split more clearly than every simulated dataset.
- Larger values give finer p-values and a better chance of keeping a true split that is only moderately clear; the run time grows in proportion. The error guarantee holds for any values.

`normal_n_initial, normal_n_refine`
For `pvalue_method = "normal_adaptive"`: the number of simulated datasets every test starts with (a whole number from 2 up, 10 by default), and the number it uses in all when its first p-value is near the level (a whole number not smaller than `normal_n_initial`, 50 by default). The defaults are those described in the SigDoTS paper. Larger values make the estimated mean and standard deviation more precise.

`n_cores` The number of CPU cores used to simulate and cluster the datasets of each test in parallel (a whole number, 1 by default). The result does not depend on it. On Windows only 1 core is used, because the parallel computation relies on process forking (`parallel::mclapply()`).

`seed` The random seed, one number (1 by default). The same data, arguments and seed give the same result on any number of cores. Each simulated dataset has its own seed, derived from `seed`, the node and the dataset's number. The random-number state of your session is restored when `sigdots()` returns.

`verbose` How much progress to print: `TRUE` (the default) prints one line per test (its node, number of spots, decision, number of simulated datasets and time); `FALSE` prints nothing; `2` prints the detailed progress of every computation.

`control` The settings that rarely need changing, made with `sigdots_control()`.

Value

An object of class `"sigdots"`. Read it with `print(fit)` (the split tree), `summary(fit)` (the settings and one row per test; see `summary.sigdots()`), `domains()` (the domain of every spot as a table), `plot(fit)` (maps, the tree and the spatial correlation; see `plot.sigdots()`) and `add_domains()` (writes the domains into a Seurat or SpatialExperiment object). For programming, the object is a list with these components:

`domains` Integer vector, one value per spot in the order of the input: the domain of the spot, numbered 1, 2, ... from left to right in the printed tree; `NA` for a spot in a part of the tree that could not be tested.

`domain_nodes` The node numbers of the domains, in the order of their numbers.

`untested_nodes`, `untested_spot_node` The node numbers of the parts that could not be tested (their spatial parameters could not be estimated, or too many of their simulated datasets had a top split with a side smaller than `min_child_size`), and, for every spot, the node number of the untested part it lies in (`NA` otherwise).

`candidate_domains`, `candidate_names` The candidate domain of every spot, numbered 1 to `K0`, and the name of each candidate domain (your own label when `candidate_labels` was given).

`tree` A data frame with one row per node of the tree: `node`, `parent`, `left` and `right` (node numbers; `NA` where there is none), `is_candidate_domain` (a bottom end of the tree), `spots`, `candidate_domains` (how many lie under the node) and `merge_score` (the increase in the within-group sum of squares caused by the merge that made the node).

`root` The node number of the top of the tree.

`spot_indices`, `groups_under` Lists named by node number: the row numbers of the spots under each node, and the candidate domains under each node.

`tests` A data frame with one row per node the testing reached, from the top, with the columns of `summary(fit)$tests` and more: `parent`, `left_spots`, `right_spots`, `tested`, `skip_reason`, `null_median` (median statistic of the simulated datasets), `null_max` (the largest number of simulated datasets the test could have used), `null_replaced` (simulated datasets replaced because their top split left fewer than `min_child_size` spots on a side; they are not counted in `null_used`), `clearer` (simulated datasets that split at least as clearly as the data), `stop_reason`, `p_initial` and `refined` (normal approximation: the p-value from the first datasets, and whether more were simulated), `ell` and `eta` (the estimated spatial parameters), five flags for estimates at the ends of their ranges (`ell_at_lower_end`, `ell_at_upper_end`, `ell_capped`, `eta_at_lower_bound`, `eta_at_upper_bound`), `backend` and `vecchia_neighbors` (the spatial computation), `pieces` (sizes of separate pieces, see `summary.sigdots()`), and, for a part whose spatial parameters could not be estimated, `spatial_fit_reason`, `spatial_fit_usable_bins` and `min_pairs_per_bin`.

`null_statistics` A list named by node number: the split statistics of the simulated datasets of each test, in the order they were simulated.

`correlograms` A list named by node number: for each test, a data frame with the spatial correlation at each distance (`distance` in spot spacings, `correlation` observed, `fitted` by the model, `pairs` used), as drawn by `plot(fit, type = "correlogram")`.

`coords`, `spot_ids`, `n_spots`, `n_genes` The positions and identifiers of the spots, and the size of the data.

`spot_pieces` The sizes of the separate pieces of the spots' nearest-neighbour graph (one number when all spots are connected).

`settings` A list of the arguments used, including `control`.

`call`, `elapsed_sec`, `version` The call, the run time in seconds and the version of SigDoTS.

How SigDoTS works

1. **Candidate domains.** The spots are divided into `K0` candidate domains with `cluster_method`, or taken from `candidate_labels`. SigDoTS never splits a candidate domain; it decides which candidate domains belong together. It therefore assumes that each candidate domain lies inside one true domain.

2. **The tree.** Each spot is linked to its `k_nn` nearest spots (see `sigdots_control()`); two candidate domains are neighbours when some of their spots are linked. Of all pairs of neighbouring groups, the pair whose merge increases the within-group sum of squares the least (Ward's criterion, computed on the means of the groups in the first `n_pcs` principal components) is merged, and this is repeated until one group is left. The result is a binary tree whose top is the whole sample and whose bottom ends are the candidate domains; every merge in it joins two groups that touch in space.
3. **The statistic.** The splits are tested from the top. The statistic of a split is the within-group sum of squares of its two sides divided by the total sum of squares of the part, summed over all genes: a number between 0 and 1, smaller for a clearer split.
4. **The reference.** To judge whether the observed statistic is small, SigDoTS simulates datasets for the spots of the part from a model in which the part is a single domain (step 5). Each simulated dataset is divided into as many candidate domains as the part has, with the same `cluster_method`; its tree is built in the same way, and the statistic of its top split is computed. Because the simulated data go through the same clustering and the same tree building as the real data, the comparison accounts for the separation that clustering finds even in data with only one domain.
5. **The single-domain model.** In the model, every spot of the part has the same mean expression, and the noise is correlated both between genes and between nearby spots.
   - The spatial correlation is described by two numbers estimated from the part's own data: `eta`, the share of the noise variance that is spatially correlated (from `eta_min` to 1), and `ell`, the distance over which the correlation reaches (Matérn 3/2 correlation function, in spot spacings).
   - They are estimated from pairs of spots in the same candidate domain, so that differences between candidate domains are not taken for spatial correlation: after the mean of each candidate domain is removed, the correlation between such pairs at each distance is matched to the correlation the model predicts (`plot(fit, type = "correlogram")` shows both).
   - The correlation between genes is then estimated from the data, with the candidate-domain means and the fitted spatial correlation removed, as a few strong components (at most `r_cap`) plus independent noise of equal variance.
   - Simulating from the model needs a factor of the spatial covariance matrix of the part's spots, computed exactly or with the Vecchia approximation (`spatial_method`).
6. **The decision.** The p-value (`pvalue_method`) is compared with the level of the test, `alpha * (m - 1) / (n - 1)` for a part of m of the n spots. If it is at most the level, the split is kept and both sides are tested in turn; otherwise the part is a domain. With these levels, the probability that any kept split is false (the family-wise error rate) is at most `alpha` (the scheme of Kimes et al., 2017).
7. **Parts that are not tested.** A part with fewer than `min_node_size` spots, a part whose split would leave fewer than `min_child_size` spots on a side, and a single candidate domain are domains without a test. A part where the spatial parameters cannot be estimated (too few pairs of spots in the same candidate domain) is left untested together with everything below it, and its spots get no domain (`NA`); `sigdots()` warns. Where the spots of a part fall into separate pieces on the nearest-neighbour graph, the pieces are joined by their closest spots before the trees are built (see `summary.sigdots()`).
8. **Simulated datasets with a too-small split.** The data's split is tested only when both of its sides have at least `min_child_size` spots, so the same is asked of every simulated dataset: one whose top split leaves fewer spots on a side is replaced by the next one in seed order and is not counted, and the test still uses its number of simulated datasets (`summary()` lists how many were replaced). If the replaced ones outnumber the valid ones, the part is left untested together with everything below it, and its spots get no domain (`NA`); `sigdots()` warns.

Choosing K0

`K0` must be larger than the number of domains you expect, and fine enough that every candidate domain lies inside one true domain, because a candidate domain that straddles two true domains can never be split. About twice the expected number of domains is a reasonable start. A very large `K0` makes the lower parts of the tree small: the spatial parameters need about 80 or more spots in a part (on a regular grid of spots), and small parts have small levels, so a Monte Carlo test there needs many simulated datasets to keep a split.

Run time and memory

Measured on simulated data with 2,000 genes and three domains (`K0 = 5`, exact computation, 10 cores; peak memory of the whole run): 900 spots, 32 seconds and 3.5 GiB; 2,500 spots, 4.3 minutes and 6.1 GiB; 10,000 spots, 18 minutes and 19.8 GiB. With the default Vecchia approximation, 2,500 spots took a median of 4.0 minutes (three domains) and 1.4 minutes (one domain) on 10 cores. Beyond about 20,000 spots use the Vecchia approximation: the exact computation at 50,176 spots took 12 hours and 206 GiB. The estimation of the spatial parameters computes the distances between all pairs of spots of the part being tested, so its memory grows with the square of the number of spots whatever `spatial_method` is: one estimation with 200 genes on a part of 10,000 spots used 2.1 GiB, of 19,881 spots 7.4 GiB, and of 40,000 spots 29 GiB.

Warnings at the end of a run

- *The spatial parameters could not be estimated at ... part(s) of the tree*: a part has too few spots for the estimation; it is marked `[!] NOT TESTED` and its spots get no domain (`NA`). A smaller `K0` makes the parts larger.
- *The simulated datasets whose top split left fewer than min_child_size spots on a side outnumbered the valid ones*: the part is marked `[!] NOT TESTED` and its spots get no domain (`NA`). This happens where the spots of a part lie in many small separate pieces; see `vignette("faq")`.
- *The estimated spatial range ell is at the upper end of its allowed range*: the true range may be longer, which can make splits easier to keep; rerun with a wider `ell_range` in `sigdots_control()`.
- *The test would need more than B_cap simulated datasets*: raise `B_cap` in `sigdots_control()`, or accept that part as a domain.

Before computing anything, `sigdots()` checks its input and stops with a message that names the argument at fault (for example, when `x` has as many columns as `coords` has rows, it suggests `t(x)`). A simulated dataset that fails also stops the run, rather than being dropped (one whose top split is too small has not failed: it is replaced, as described above). Every message is explained in `vignette("faq")`.

References

Wang H, Cao F, Shen H, Liu Y, Yan G (2026). Statistical significance testing for spatial domain detection in spatial transcriptomics. Manuscript.

Kimes PK, Liu Y, Hayes DN, Marron JS (2017). Statistical significance for hierarchical clustering. *Biometrics* 73(3), 811-821.

Vecchia AV (1988). Estimation and model identification for continuous spatial processes. *Journal of the Royal Statistical Society Series B* 50(2), 297-312.

Katzfuss M, Guinness J (2021). A general framework for Vecchia approximations of Gaussian processes. *Statistical Science* 36(1), 124-141.

Singhal V, Chou N, Lee J, et al. (2024). BANKSY unifies cell typing and tissue domain segmentation for scalable spatial omics data analysis. *Nature Genetics* 56(3), 431-441.

Zhao E, Stone MR, Ren X, et al. (2021). Spatial transcriptomics at subspot resolution with BayesSpace. *Nature Biotechnology* 39(11), 1375-1384.

See Also

`sigdots_control()` for the other settings, `summary.sigdots()` for reading the result, `vignette("SigDoTS")` for a worked example, `vignette("own-data")` for Seurat and SpatialExperiment objects and the choice of settings, `vignette("how-it-works")` for the method with figures.

Examples

```
# about 20 seconds on one core
fit <- sigdots(sigdots_example$x, sigdots_example$coords, K0 = 5)
fit
summary(fit)
table(domain = domains(fit)$domain, true_region = sigdots_example$region)
plot(fit)
plot(fit, type = "tree")

# more simulated datasets per test (finer p-values), on 2 cores
fit2 <- sigdots(sigdots_example$x, sigdots_example$coords, K0 = 5,
                B_min = 50, B_max = 50, n_cores = 2)

# the normal approximation of the p-value
fit3 <- sigdots(sigdots_example$x, sigdots_example$coords, K0 = 5,
                pvalue_method = "normal_adaptive")
summary(fit3)
```

---

sigdots_control *Settings of sigdots() that rarely need changing*

Description

`sigdots_control()` collects the settings of `sigdots()` that rarely need changing. Its defaults are the values used in the SigDoTS paper. Pass its result as the `control` argument of `sigdots()`, for example `sigdots(x, coords, K0 = 10, control = sigdots_control(ell_range = c(0.3, 80)))`; settings not named keep their defaults.

Usage

```
sigdots_control(
  k_nn = 6,
  n_comp_min = 0.01,
  min_node_size = 20,
  min_child_size = 8,
  B_cap = 5000,
  eta_min = 0.01,
  ell_range = c(0.3, 40),
```

```
  ell_candidates = 160,
  pairs_per_bin = 4000,
  min_pairs_per_bin = 200,
  r_cap = 30,
  ridge = 1e-06,
  nugget = 1e-06,
  max_exact_memory_gib = 16,
  candidate_seed = NULL,
  normal_band = 10
)
```

Arguments

`k_nn`	The number of nearest spots each spot is linked to (a whole number, 6 by default; on the hexagonal Visium array each spot has 6 nearest neighbours). Two candidate domains are neighbours, and can be merged into one part of the tree, when some of their spots are linked. The same links are used for every simulated dataset.
`n_comp_min`	A number greater than 0 and at most 1 (0.01 by default). When the spots of a candidate domain fall into separate pieces on the nearest-neighbour links, pieces with fewer spots than this share of the candidate domain are ignored when deciding which candidate domains are neighbours, so that a few stray spots do not make distant candidate domains neighbours. If ignoring them would leave the candidate domains unconnected, all pieces are used.
`min_node_size`	A whole number (20 by default). A part of the tree with fewer spots is not tested and is reported as a domain.
`min_child_size`	A whole number (8 by default). A split that would leave fewer spots than this on either side is not tested, and the part is reported as a domain. A simulated dataset whose top split leaves fewer spots on a side is replaced by a new one, so that the test keeps its number of simulated datasets (`summary()` lists how many were replaced). If such simulated datasets outnumber the valid ones, the part is not tested, its spots get no domain, and `sigdots()` warns.
`B_cap`	The largest number of simulated datasets a single Monte Carlo test may use (a whole number, 5000 by default, at least `B_max`). A test on m of n spots needs about `(n - 1) / (alpha * (m - 1))` simulated datasets to keep a split, so a test on a small part of a large sample may need more than `B_cap`; that part is then reported as a domain without a test, and `sigdots()` warns. Not used by `pvalue_method = "normal_adaptive"`.
`eta_min`	The smallest value allowed for `eta`, the share of the noise variance that is spatially correlated (a number greater than 0 and smaller than 1, 0.01 by default; the largest value is 1).
`ell_range`	The range searched for `ell`, the distance over which the noise is spatially correlated: two increasing positive numbers, in spot spacings (the median distance from a spot to its nearest other spot in the part being tested), `c(0.3, 40)` by default. `ell` is also kept below twice the largest distance between two spots of the same candidate domain. When `sigdots()` warns that `ell` is at the upper end of this range, the true range may be longer, which can make splits easier to keep; rerun with a wider range, for example `c(0.3, 80)`.

`ell_candidates`
: The number of values of `ell` tried in `ell_range`, evenly spaced on a log scale (a whole number from 2, 160 by default); the best of them is then refined between its two neighbouring values.

`pairs_per_bin, min_pairs_per_bin`
: `eta` and `ell` are estimated from pairs of spots in the same candidate domain, grouped by their distance into classes (one per spot spacing from 0.5 to 30.5 spacings, then 30.5-35.5, 35.5-42.5, 42.5-55.5 and 55.5-75 spacings). At most `pairs_per_bin` pairs (a whole number, 4000 by default) are drawn at random from each class, and a class with fewer than `min_pairs_per_bin` pairs (200 by default) is not used. At least 4 usable classes are needed; a part with fewer is not tested, its spots get no domain, and `sigdots()` warns.

`r_cap`
: The largest number of strong components kept in the correlation between genes of the simulated data (a whole number, 30 by default). The number actually kept is estimated from the data at every test.

`ridge, nugget`
: Small non-negative numbers added to the diagonals of the gene covariance and of the spatial covariance matrices, for numerical stability (both 1e-6 by default).

`max_exact_memory_gib`
: For `spatial_method = "exact"`: the largest memory, in GiB (1 GiB = 2^30 bytes), a test may need (a positive number, 16 by default). If a test is estimated to need more, the run stops before it starts and suggests the Vecchia approximation or a larger value. 16 GiB is reached at about 19,800 spots.

`candidate_seed`
: The random seed used to make the candidate domains (k-means, BANKSY or BayesSpace), one number. `NULL` (the default) uses the `seed` of `sigdots()`. Setting it separately keeps the candidate domains fixed while `seed` changes the simulated datasets.

`normal_band`
: For `pvalue_method = "normal_adaptive"`: a number larger than 1 (10 by default). A test whose first p-value lies between its level divided by `normal_band` and its level multiplied by `normal_band` simulates more datasets, up to `normal_n_refine`, before deciding.

Details

Most analyses need none of these changed. The ones you may need: `ell_range` and `B_cap`, when `sigdots()` warns about them; `max_exact_memory_gib`, to use the exact computation on a large sample; `candidate_seed`, to reproduce the candidate domains of an earlier run. Values other than the defaults have not been evaluated.

Value

A list of class `"sigdots_control"` holding the settings, for the `control` argument of `sigdots()`. Every value is checked when the list is made; an invalid one stops with a message naming the setting.

See Also

`sigdots()`.

Examples

```
sigdots_control()
sigdots_control(ell_range = c(0.3, 80), B_cap = 20000)
```

---

sigdots_example *A small simulated dataset with three spatial domains*

Description

900 spots on a 30 x 30 grid, 200 genes, three true domains of 300 spots each (vertical bands). The first 40 genes ("marker" genes) have different means in the three domains; the other 160 do not. The noise is spatially correlated: half of its variance (`eta = 0.5`) is correlated across spots, over a range of about 3 spot spacings (`ell = 3`, Matérn 3/2 correlation), and the genes are correlated with each other. Expression values are log-normalised counts simulated from this model, as in the simulation study of the SigDoTS paper.

Usage

```
sigdots_example
```

Format

A list with three elements:

x Numeric matrix, 900 spots x 200 genes, log-normalised expression. Row names are spot identifiers; column names mark the 40 marker genes.

coords Numeric matrix, 900 x 2, the spot coordinates (columns `x`, `y`).

region Integer vector of length 900, the true domain of each spot (1, 2 or 3). SigDoTS does not use it; it is there to compare with the result.

Details

`sigdots(sigdots_example$x, sigdots_example$coords, K0 = 5)` runs in about 20 seconds on one core and recovers the three domains; `vignette("SigDoTS")` walks through it.

Source

Simulated for the package with the data generator of the SigDoTS paper's simulation study: a 30 x 30 grid of spots divided into three vertical bands of 10 columns; for each gene, a latent log-expression with a band-specific mean for the 40 marker genes, spatially correlated noise (`eta = 0.5`, `ell = 3` spot spacings) and correlation between genes; counts drawn from a Poisson distribution given the latent values and a spot-specific sequencing depth; then log(1 + 10,000 x count / total count of the spot). Random seed 20261008.

Examples

```
str(sigdots_example, max.level = 1)
table(sigdots_example$region)
```

`sigdots_prepare` *Expression matrix for sigdots() from a counts matrix*

Description

`sigdots_prepare()` makes the expression matrix `x` that `sigdots()` takes from a matrix of raw counts. It is the preparation used in the SigDoTS paper's simulations; any other normalisation and gene selection (Seurat's, scran's, ...) can be used instead.

Usage

```
sigdots_prepare(counts, n_genes = 2000, scale_factor = 10000)
```

Arguments

`counts`	The raw counts: one row per gene and one column per spot, as most tools return them, with the gene names as row names and the spot identifiers as column names. A matrix or a sparse `Matrix`.
`n_genes`	The number of genes to keep (a whole number, 2000 by default). All genes are kept when there are fewer.
`scale_factor`	Each spot's counts are scaled to this total before the log transform (10,000 by default).

Details

Three steps:

1. Normalisation: the counts of each spot are divided by the spot's total count and multiplied by `scale_factor`.
2. Log transform: log(1 + value), as `Seurat::NormalizeData()` does by default.
3. Gene selection: the variance of every gene's log-normalised values across the spots is computed, and the `n_genes` genes with the largest variance are kept, in decreasing order of variance.

The values are not scaled to variance 1. A sparse `dgCMatrix` stays sparse until the selected genes are turned into the dense result. Spots with a total count of 0 stop the function; remove them first.

Value

A numeric matrix with one row per spot and one column per selected gene (the spot identifiers as row names, the gene names as column names), ready for `sigdots()`.

Examples

```
counts <- t(round(expm1(sigdots_example$x) * 50))   # a toy counts matrix, genes x spots
x <- sigdots_prepare(counts, n_genes = 100)
dim(x)
```

`summary.sigdots` *Print and summarise a SigDoTS result*

Description

`print(fit)` (or typing `fit`) shows the number of domains and the split tree of a `sigdots()` result. `summary(fit)` shows, in addition, the settings of the run and a table with one row per test; the table is also returned, as `summary(fit)$tests`, for use in your own code.

Usage

```
## S3 method for class 'sigdots'
print(x, ...)

## S3 method for class 'sigdots'
summary(object, ...)

## S3 method for class 'summary.sigdots'
print(x, ...)
```

Arguments

`x, object`	A result of `sigdots()` (for `print.summary.sigdots()`, a result of `summary()`).
`...`	Not used.

Details

The printed summary has four parts: the settings (header lines), the number of domains, the split tree, and the tests table, followed, where they apply, by the tests in which simulated datasets were replaced, the parts that could not be tested, the parts whose spots fall into separate pieces, and a note when the candidate domains were supplied by you.

Value

`print()` returns its argument invisibly. `summary()` returns a list of class `"summary.sigdots"` with two elements: `fit`, the result, and `tests`, the table described above (a data frame with the columns `node`, `spots`, `candidate_domains`, `statistic`, `null_used`, `p_value`, `level` and `decision`). Printing it shows the summary.

The header lines

- `Data`: the number of spots and genes, and, for an object, where the expression came from.
- `Start`: how the candidate domains were made (`cluster_method`, or "labels supplied by the user") and how many there are.
- `Spatial`: the computation of the spatial covariance (Vecchia approximation with its number of neighbours, or exact).
- `Level`: `alpha`, the significance level for the whole tree.
- `P-values`: "Monte Carlo", or "normal approximation" with its numbers of simulated datasets (see `pvalue_method` in `sigdots()`).

The split tree

Every line that starts with a number in brackets, `[9]` say, is a part of the tree (a *node*): the number of spots it holds and the candidate domains it is made of, then the decision of the test at that node. SigDoTS tests from the top: the whole sample first, then the two sides of every split. `split` means the two sides are significantly different and each side is tested in turn; `no split` means they are not, and the part is a domain. `=> DOMAIN k` marks a part that was not split further: it is the k-th domain, and the lines under it list the candidate domains it contains with their numbers of spots. A single candidate domain is never tested; when its parent is split, it is a domain by itself. `[!] NOT TESTED` marks a part that could not be tested, because its spatial parameters could not be estimated or because too many of its simulated datasets had a top split with a side smaller than `min_child_size`; its spots get no domain. The tree is drawn with box-drawing characters, or with `|--` and `` `-- `` where the console cannot show them.

The tests table

One row per part of the tree that the testing reached, from the top. The printed table shows the tested parts; the returned table, `summary(fit)$tests`, has every reached part, with `NA` numbers for those not tested. The columns:

- `node` (printed as `test`): the node number used in the tree;
- `spots`: the number of spots in that part;
- `candidate_domains`: the candidate domains in that part (in the returned table only);
- `statistic`: the split statistic of the data, between 0 and 1: the within-group sum of squares of the two sides of the split divided by the total sum of squares, over all genes. A smaller value means a clearer split;
- `null_used`: the number of simulated single-domain datasets the test used. A simulated dataset whose top split leaves fewer than `min_child_size` spots on a side is replaced by a new one and is not counted here; the summary lists the tests where this happened, with the number replaced;
- `p_value` (printed as `p`): the p-value of the test. With the Monte Carlo p-value it is (1 + the number of simulated datasets whose statistic was at most the observed one) / (1 + `null_used`), so with few simulated datasets it can take only a few values; the decision is what the test establishes. With the normal approximation it is the normal probability of a statistic at most the observed one, given the mean and standard deviation of the simulated statistics; it is printed in scientific notation when below 0.0001;
- `level`: the significance level of this test, `alpha * (spots - 1) / (n - 1)` for a sample of n spots, which keeps the probability of any false split in the whole tree at most `alpha`;
- `decision`: `split` when `p_value <= level`, `no split` otherwise, or why the part was not tested: `not tested: spatial fit` (the spatial parameters could not be estimated because the part has too few spots; its spots get no domain), `not tested: simulated splits too smal` (the simulated datasets whose top split left fewer than `min_child_size` spots on a side outnumbered the valid ones; its spots get no domain), `not tested: too few spots` or `not tested: a side too small` (smaller than `min_node_size` or `min_child_size` of `sigdots_control()`; reported as a domain), `not tested: B_cap too small` (the level is so small that more than `B_cap` simulated datasets would be needed; reported as a domain).

Separate pieces

The tree needs the spots of each tested part to be connected through their nearest-neighbour links (`k_nn` of `sigdots_control()`). Where they fall into separate pieces (a layer cut by a fold

of the tissue, or a few spots whose neighbours all belong to other candidate domains), SigDoTS joins each smaller piece to the rest by a link between their closest spots, repeatedly until all are connected, before it builds the trees of the data and of the simulated datasets. Nothing changes for parts whose spots are connected. `summary()` lists the parts where this happened, with the sizes of the pieces (for example "242, 49, 14 and 3 single spots").

See Also

`domains()` for the domain of every spot, `plot.sigdots()` for the maps and the tree, `sigdots()` for the components of the result.

Examples

```
fit <- sigdots(sigdots_example$x, sigdots_example$coords, K0 = 5, verbose = FALSE)
fit
summary(fit)
summary(fit)$tests
```

Index

∗ **datasets**
sigdots_example, 16

add_domains, 2
add_domains(), *3*, *9*

domains, 3
domains(), *3*, *5*, *9*, *20*

plot.sigdots, 4
plot.sigdots(), *4*, *9*, *20*
print.sigdots *(summary.sigdots)*, 18
print.summary.sigdots *(summary.sigdots)*, 18

sigdots, 6
sigdots(), *2–4*, *13–15*, *17*, *18*, *20*
sigdots_control, 13
sigdots_control(), *8*, *9*, *11–13*, *19*
sigdots_example, 16
sigdots_prepare, 17
sigdots_prepare(), *6*
summary.sigdots, 18
summary.sigdots(), *4*, *5*, *9–11*, *13*