

Package: KODAMA (via r-universe)

August 20, 2026

Title Knowledge Discovery by Accuracy Maximization

Version 0.99.6

Author Stefano Cacciatore [aut, cre]

Maintainer Stefano Cacciatore <stefano.cacciatore@icgeb.org>

Description Knowledge Discovery by Accuracy Maximization using KNN or SIMPLS followed by latent-space LDA. The portable source package vendors the MIT-licensed CPU implementation from the standalone 'kodama-cpp' library. Optional developer builds can link the same public interface to NVIDIA CUDA or Apple Metal backends. Reusable nearest-neighbor graphs, graph-aware visualization, preprocessing, and adapters for common single-cell containers are provided.

License MIT + file LICENSE

Depends R (>= 4.6.0)

Encoding UTF-8

Roxygen list(markdown = TRUE)

LinkingTo Rcpp

Imports methods, Rcpp, S4Vectors, stats

Suggests BiocStyle, knitr, Matrix, rmarkdown, Seurat, SeuratObject, SingleCellExperiment, SpatialExperiment, SummarizedExperiment, testthat (>= 3.0.0)

VignetteBuilder knitr

biocViews Software, Classification, Clustering, DimensionReduction, GPU, Preprocessing, Visualization, SingleCell, Spatial

Config/testthat/edition 3

SystemRequirements C++17

URL <https://github.com/tkcaccia/KODAMA>

BugReports <https://github.com/tkcaccia/KODAMA/issues>

Config/roxygen2/version 8.0.0

Repository <https://tkcaccia.r-universe.dev>

Date/Publication 2026-08-20 17:20:38 UTC

RemoteUrl <https://github.com/tkcaccia/kodama>

RemoteRef HEAD

RemoteSha c0f203cf6246b789f2a6122c1b1ee538763d9986

Contents

CoreKNN	3
CorePLSLDA	4
dinisurface	5
helicoid	6
KNNCV	6
KODAMA.clustering	7
KODAMA.diagnostics	8
KODAMA.graph	9
KODAMA.graph.materialize	10
KODAMA.timing	11
KODAMA.visualization	11
KODAMA_backend	13
kodama_matrix	13
kodama_matrix_graph	16
kodama_pca	18
lymphoma	20
MetRef	20
normalization	21
passing.message	22
PLSLDACV	23
RunFastPCA	24
RunKODAMAgaph	26
RunKODAMAmatrix	28
RunKODAMAvsualization	30
RunSpatialFeatureSelection	31
scaling	33
spatial_feature_selection	34
spirals	35
swissroll	36
USA	36

Index	38
--------------	-----------

CoreKNN

*KODAMA core optimization with KNN classifier***Description**

KODAMA core optimization with KNN classifier

Usage

```
CoreKNN(
  data,
  labels,
  constrain = NULL,
  fix = NULL,
  cycles = 100L,
  folds = 10L,
  stratified = TRUE,
  seed = 1L,
  k = 30L,
  metric = c("euclidean", "cosine", "inner_product"),
  backend = NULL,
  n.cores = 4L,
  gpu.device = 0L
)
```

Arguments

<code>data</code>	Numeric matrix with samples in rows and variables in columns.
<code>labels</code>	Initial labels for the label-evolution process.
<code>constrain</code>	Optional integer vector assigning samples to indivisible proposal and fold groups.
<code>fix</code>	Optional integer vector marking samples whose labels are fixed.
<code>cycles</code>	Number of proposal/evaluation cycles.
<code>folds</code>	Number of cross-validation folds.
<code>stratified</code>	Whether to stratify folds by class labels.
<code>seed</code>	Integer random seed.
<code>k</code>	Number of neighbors used by the KNN classifier.
<code>metric</code>	Distance or similarity metric.
<code>backend</code>	Execution backend: "cpu", "cuda", or "metal".
<code>n.cores</code>	Number of CPU worker threads requested by the wrapper.
<code>gpu.device</code>	CUDA device id when backend = "cuda".

Value

A list containing the best labels and predictions, accuracy and score traces, plus explicit proposal, acceptance, rejection, coarsening, and absorption counters for optimizer-state diagnostics.

Examples

```
x <- as.matrix(iris[, 1:4])
y <- rep(1:3, each = 50)
fit <- CoreKNN(x, y, cycles = 1, folds = 3, k = 5, n.cores = 1)
fit$accuracy
```

CorePLSLDA

*KODAMA core optimization with PLS-LDA classifier***Description**

KODAMA core optimization with PLS-LDA classifier

Usage

```
CorePLSLDA(
  data,
  labels,
  constrain = NULL,
  fix = NULL,
  cycles = 100L,
  folds = 10L,
  stratified = TRUE,
  seed = 1L,
  ncomp = min(50L, ncol(data)),
  backend = NULL,
  n.cores = 4L,
  gpu.device = 0L
)
```

Arguments

<code>data</code>	Numeric matrix with samples in rows and variables in columns.
<code>labels</code>	Initial labels for the label-evolution process.
<code>constrain</code>	Optional integer vector assigning samples to indivisible proposal and fold groups.
<code>fix</code>	Optional integer vector marking samples whose labels are fixed.
<code>cycles</code>	Number of proposal/evaluation cycles.
<code>folds</code>	Number of cross-validation folds.
<code>stratified</code>	Whether to stratify folds by class labels.
<code>seed</code>	Integer random seed.
<code>ncomp</code>	Number of SIMPLS latent components requested.
<code>backend</code>	Execution backend: "cpu", "cuda", or "metal".
<code>n.cores</code>	Number of CPU worker threads requested by the wrapper.
<code>gpu.device</code>	CUDA device id when backend = "cuda".

Value

A list containing the best labels and predictions, accuracy and score traces, plus explicit proposal, acceptance, rejection, coarsening, and absorption counters for optimizer-state diagnostics.

Examples

```
x <- as.matrix(iris[, 1:4])
y <- rep(1:3, each = 50)
fit <- CorePLSLDA(x, y, cycles = 1, folds = 3, ncomp = 2, n.cores = 1)
fit$accuracy
```

dinisurface	<i>Generate Dini's surface</i>
-------------	--------------------------------

Description

Generates points distributed on a Dini surface using the construction from the KODAMA R package.

Usage

```
dinisurface(N = 1000)
```

Arguments

N Number of data points.

Value

An N by 3 numeric matrix with columns x, y, and z.

See Also

[helicoid\(\)](#), [spirals\(\)](#), [swissroll\(\)](#)

Examples

```
dim(dinisurface(100))
```

helicoid	<i>Generate a helicoid surface</i>
----------	------------------------------------

Description

Generates points distributed on a helicoid using the construction from the KODAMA R package.

Usage

```
helicoid(N = 1000)
```

Arguments

N Number of data points.

Value

An N by 3 numeric matrix with columns x, y, and z.

See Also

[dinisurface\(\)](#), [spirals\(\)](#), [swissroll\(\)](#)

Examples

```
dim(helicoid(100))
```

KNNCV	<i>Cross-validated KNN classification</i>
-------	---

Description

Cross-validated KNN classification

Usage

```
KNNCV(
  data,
  labels,
  constrain = NULL,
  folds = 10L,
  stratified = TRUE,
  seed = 1L,
  k = 10L,
  metric = c("cosine", "inner_product", "euclidean"),
  backend = NULL,
```

```

    n.cores = 1L,
    gpu.device = 0L
  )

```

Arguments

<code>data</code>	Numeric matrix with samples in rows and variables in columns.
<code>labels</code>	Class labels used as the cross-validation truth.
<code>constrain</code>	Optional integer vector assigning samples to indivisible fold groups.
<code>folds</code>	Number of cross-validation folds.
<code>stratified</code>	Whether to stratify folds by class labels.
<code>seed</code>	Integer random seed for fold construction.
<code>k</code>	Number of neighbors used by the KNN classifier.
<code>metric</code>	Distance or similarity metric.
<code>backend</code>	Execution backend: "cpu", "cuda", or "metal".
<code>n.cores</code>	Number of CPU worker threads requested by the wrapper.
<code>gpu.device</code>	CUDA device id when backend = "cuda".

Value

A cross-validation result containing predictions, truth, fold assignments, fold and global accuracy, confusion matrix, timing, and backend diagnostics.

Examples

```

x <- as.matrix(iris[, 1:4])
y <- as.integer(iris$Species)
fit <- KNNCV(x, y, folds = 3, k = 5, backend = "cpu")
fit$accuracy

```

KODAMA.clustering	<i>Cluster a graph or embedding with random walks</i>
-------------------	---

Description

Cluster a graph or embedding with random walks

Usage

```

KODAMA.clustering(
  x,
  n.clusters = 0L,
  weight = c("distance", "snn", "adaptive", "binary"),
  k = 30L,
  metric = c("euclidean", "cosine", "inner_product"),

```

```

graph.backend = NULL,
n.cores = 4L,
n.iterations = 10L,
random.walk.steps = 4L,
gpu.device = 0L
)

```

Arguments

<code>x</code>	Input embedding matrix, KODAMA result, or KNN graph list.
<code>n.clusters</code>	Optional target number of clusters. Random-walk clustering reports an error when it cannot produce the requested count exactly.
<code>weight</code>	Graph edge-weighting rule.
<code>k</code>	Number of neighbors used when <code>x</code> is an embedding matrix.
<code>metric</code>	Distance or similarity metric used when <code>x</code> is a matrix.
<code>graph.backend</code>	Backend used to construct a graph from an embedding: "cpu", "cuda", or "metal". Clustering itself runs on CPU.
<code>n.cores</code>	Number of CPU worker threads requested by the wrapper.
<code>n.iterations</code>	Number of clustering refinement iterations.
<code>random.walk.steps</code>	Number of random-walk steps.
<code>gpu.device</code>	CUDA device id when <code>graph.backend = "cuda"</code> .

Value

A list containing integer cluster membership, cluster count, graph metadata, and runtime diagnostics.

Examples

```

set.seed(1)
x <- rbind(matrix(rnorm(40, -2), 20, 2), matrix(rnorm(40, 2), 20, 2))
fit <- KODAMA.clustering(x, k = 5, n.cores = 1)
table(fit$membership)

```

KODAMA.diagnostics	<i>Diagnose wrapper runtime libraries and environment</i>
--------------------	---

Description

Diagnose wrapper runtime libraries and environment

Usage

```
KODAMA.diagnostics(all = FALSE)
```

Arguments

`all` If TRUE, return all linked shared libraries reported by the platform linker tool. If FALSE, keep only likely runtime dependencies.

Value

A `kodama_diagnostics` list describing package and runtime state.

Examples

```
KODAMA.diagnostics()
```

KODAMA.graph

Build the reusable KODAMA graph and visualization initialization

Description

Build the reusable KODAMA graph and visualization initialization

Usage

```
KODAMA.graph(
  data,
  spatial = NULL,
  samples = NULL,
  k = 100L,
  metric = c("euclidean", "cosine", "inner_product"),
  backend = NULL,
  n.cores = NULL,
  gpu.device = 0L,
  seed = 1234L,
  storage = c("handle", "matrix")
)
```

Arguments

`data` Numeric matrix with samples in rows and variables in columns.

`spatial` Optional spatial coordinates. When supplied, their invariant nearest-neighbor graph and jitter scale are prepared once here and reused by every classifier passed to `KODAMA.matrix`.

`samples` Optional sample or slide identifier. Multiple samples are separated along spatial coordinate 1 using the original KODAMA rule.

`k` Number of nearest neighbors to retain.

`metric` Distance or similarity metric.

`backend` Execution backend: "cpu", "cuda", or "metal".

n.cores	Number of worker threads. NULL uses options(n.cores = ...), then N_CORES, and otherwise 4.
gpu.device	CUDA device id when backend = "cuda".
seed	Integer seed used by the backend-specific PCA initialization.
storage	Graph representation. "matrix" returns conventional R matrices; "handle" keeps one float32 graph behind a process-local external pointer. Materialize a handle before saving it for another R session.

Value

A kodama_graph object containing either graph matrices or a graph handle, plus PCA starts for UMAP and openTSNE. Raw data are not retained.

Examples

```
g <- KODAMA.graph(as.matrix(iris[, 1:4]), k = 5, n.cores = 1)
g$parameters
```

KODAMA.graph.materialize

Materialize a handle-backed KODAMA graph

Description

Materialize a handle-backed KODAMA graph

Usage

```
KODAMA.graph.materialize(graph)
```

Arguments

graph	A graph returned by KODAMA.graph(storage = "handle") or a KODAMA result whose knn member uses handle storage.
-------	---

Value

A graph list containing R index and distance matrices. Materialized graphs can be serialized normally.

Examples

```
g <- KODAMA.graph(as.matrix(iris[, 1:4]), k = 5, storage = "handle")
materialized <- KODAMA.graph.materialize(g)
dim(materialized$indices)
```

KODAMA.timing	<i>Return a compact timing table for a KODAMA result</i>
---------------	--

Description

Return a compact timing table for a KODAMA result

Usage

```
KODAMA.timing(x)
```

Arguments

x	A kodama_matrix result or compatible list with timing fields. Matrix results include separate landmark-selection aggregates and classifier optimization timings.
---	--

Value

A data frame with analysis steps, elapsed seconds, and percentage of the recorded total runtime.

Examples

```
KODAMA.timing(list(timing = list(graph = 0.5, runtime_seconds = 2)))
```

KODAMA.visualization	<i>Visualize a matrix or KODAMA graph with UMAP or openTSNE</i>
----------------------	---

Description

Visualize a matrix or KODAMA graph with UMAP or openTSNE

Usage

```
KODAMA.visualization(
  x,
  method = c("UMAP", "t-SNE", "opentsne"),
  init = NULL,
  raw.data = NULL,
  initialize.from.raw = TRUE,
  k = 30L,
  metric = c("euclidean", "cosine", "inner_product"),
  backend = NULL,
  n.cores = NULL,
  gpu.device = 0L,
  n.epochs = 200L,
```

```

    n.iter = 500L,
    perplexity = 30,
    graph.mode = c("fuzzy", "binary"),
    seed = 4L,
    ...
  )

```

Arguments

<code>x</code>	Input matrix, KODAMA result, or KNN graph list.
<code>method</code>	Embedding method.
<code>init</code>	Optional two-column initialization matrix.
<code>raw.data</code>	Optional raw data used for backend-native PCA initialization. When <code>x</code> is a raw matrix, that matrix is used automatically. A stored initialization from <code>KODAMA.matrix()</code> is reused first when its backend matches backend; <code>raw.data</code> is used only when a compatible stored start is unavailable.
<code>initialize.from.raw</code>	Whether raw-data PCA initialization is the default when no explicit <code>init</code> is supplied.
<code>k</code>	Number of graph neighbors used by the embedding.
<code>metric</code>	Distance or similarity metric used when <code>x</code> is a matrix.
<code>backend</code>	Execution backend: "cpu", "cuda", or "metal" for UMAP and openTSNE.
<code>n.cores</code>	Number of worker threads. NULL uses <code>options(n.cores = ...)</code> , then <code>N_CORES</code> , and otherwise 4.
<code>gpu.device</code>	CUDA device id when backend = "cuda".
<code>n.epochs</code>	Number of UMAP optimization epochs.
<code>n.iter</code>	Number of openTSNE optimization iterations.
<code>perplexity</code>	openTSNE perplexity.
<code>graph.mode</code>	UMAP graph weighting mode. "fuzzy" is the default; "binary" remains available as an explicit compatibility mode.
<code>seed</code>	Integer random seed.
<code>...</code>	Additional embedding parameters forwarded to the C++ wrapper.

Value

A two-column numeric matrix with attributes describing the selected backend, optimizer, initialization source and backend, runtime, and (for UMAP) fuzzy-graph edge count and maximum weight. The graph diagnostics are zero for openTSNE because its sparse probability matrix is not a UMAP fuzzy graph.

Examples

```

fit <- KODAMA.visualization(
  as.matrix(iris[, 1:4]), method = "UMAP", k = 5,
  n.epochs = 5, n.cores = 1
)
dim(fit)

```

KODAMA_backend	<i>Configure the default KODAMA execution backend</i>
----------------	---

Description

KODAMA uses an explicit backend argument when supplied. Otherwise it reads `options(backend = ...)`, then `BACKEND`, and finally uses `"cpu"`. The former KODAMA-specific option and environment variable remain compatibility fallbacks. This function gets or sets the shared option.

Usage

```
KODAMA_backend(backend = NULL)
```

Arguments

backend Optional backend: `"cpu"`, `"cuda"`, or `"metal"`.

Value

The active backend. Setting returns the previous value invisibly.

Examples

```
KODAMA_backend()
old <- KODAMA_backend("cpu")
options(backend = old)
```

kodama_matrix	<i>Run KODAMA matrix optimization</i>
---------------	---------------------------------------

Description

Run KODAMA matrix optimization

Usage

```
kodama_matrix(
  data = NULL,
  graph = NULL,
  spatial = NULL,
  samples = NULL,
  W = NULL,
  constrain = NULL,
  fix = NULL,
  M = 100L,
  Tcycle = 20L,
```

```

ncomp = NULL,
landmarks = 10000L,
splitting = NULL,
n.cores = NULL,
graph.neighbors = NULL,
knn.k = 30L,
spatial.resolution = 0.4,
spatial.graph.mix = FALSE,
spatial.constraint.mode = c("kmeans", "graph", "auto"),
spatial.mode = c("standard", "population"),
metric = "euclidean",
classifier = c("knn", "pls_lda"),
backend = NULL,
seed = 1234L,
folds = 5L,
visual.init = TRUE,
progress = TRUE,
progress.file = NULL,
apply.kodama.dissimilarity = TRUE,
return.graph = FALSE,
...
)

```

Arguments

<code>data</code>	Optional raw numeric matrix. It is required when <code>graph</code> is not supplied. The C++ core stores analysis values as <code>float32</code> .
<code>graph</code>	Optional <code>KODAMA.graph</code> result or a bare list with indices and distances.
<code>spatial</code>	Optional numeric matrix of spatial coordinates.
<code>samples</code>	Optional sample or slide identifier. With more than one unique value, spatial coordinate 1 is separated exactly as in the original KODAMA implementation before spatial graph and constraint construction.
<code>W</code>	Optional integer vector.
<code>constrain</code>	Optional integer vector of sample constraints.
<code>fix</code>	Optional integer vector marking fixed samples.
<code>M</code>	Number of independent KODAMA runs.
<code>Tcycle</code>	Number of optimization cycles per run.
<code>ncomp</code>	Number of PLS components for the PLS-LDA classifier.
<code>landmarks</code>	Maximum number of samples optimized directly in each run.
<code>splitting</code>	Initial number of label classes used for each run.
<code>n.cores</code>	CPU worker count. <code>NULL</code> uses <code>options(n.cores = ...)</code> , then <code>N_CORES</code> , and otherwise 4. For CUDA or Metal matrix optimization, an explicit 0 enables backend-specific automatic independent-run lane selection.
<code>graph.neighbors</code>	Number of neighbors retained in the returned graph.

<code>knn.k</code>	Number of neighbors used by the KNN classifier.
<code>spatial.resolution</code>	Resolution parameter for constrained grouping when optional coordinate constraints are supplied.
<code>spatial.graph.mix</code>	Logical flag passed to the C++ constraint builder.
<code>spatial.constraint.mode</code>	Constraint construction mode.
<code>spatial.mode</code>	Coordinate treatment. "population" applies the classic repeated-coordinate regularization independently in every M run before spatial clustering. The default "standard" is unchanged.
<code>metric</code>	Distance or similarity metric.
<code>classifier</code>	Either "knn" or "pls_lda".
<code>backend</code>	Execution backend: "cpu", "cuda", or "metal".
<code>seed</code>	Integer random seed.
<code>folds</code>	Number of cross-validation folds used by the classifier.
<code>visual.init</code>	Whether the same native matrix call performs one float32 PCA and stores both UMAP and openTSNE initializations for reuse by KODAMA.visualization.
<code>progress</code>	Whether the C++ core prints run/cycle progress.
<code>progress.file</code>	Optional progress-log path. When progress=TRUE, the native workers append M/Tcycle checkpoints here. A temporary path is created and announced when this is NULL.
<code>apply.kodama.dissimilarity</code>	Whether to return the KODAMA-corrected neighbor graph rather than only the base graph.
<code>return.graph</code>	FALSE omits the result graph, TRUE materializes R index and distance matrices, and "handle" retains one float32 graph in an external C++ pointer for reuse without two large R matrices.
<code>...</code>	Reserved internal controls for reproducibility experiments.

Value

A list returned by the kodama-cpp core. The full-data graph is built once before all M runs. The result contains one knn graph; `knn_is_kodama_corrected`, `graph_storage_bytes`, `graph_builds`, and `timing` expose its lifecycle. Per-run landmark-selection times are retained in `landmark_seconds`; their sum, mean, and median are also reported in `timing` so landmark construction is not conflated with the classifier core.

Examples

```
x <- as.matrix(iris[, 1:4])
fit <- KODAMA.matrix(
  data = x, classifier = "knn", M = 1, Tcycle = 1,
  landmarks = 100, splitting = 10, knn.k = 5, n.cores = 1,
  return.graph = FALSE
)
fit$acc
```

kodama_matrix_graph	<i>KODAMA matrix from a precomputed KNN graph</i>
---------------------	---

Description

KODAMA matrix from a precomputed KNN graph

Usage

```
kodama_matrix_graph(
  indices,
  distances = NULL,
  data = NULL,
  spatial = NULL,
  samples = NULL,
  W = NULL,
  constrain = NULL,
  fix = NULL,
  M = 100L,
  Tcycle = 20L,
  ncomp = NULL,
  landmarks = 10000L,
  splitting = NULL,
  n.cores = 4L,
  graph.neighbors = NULL,
  knn.k = 30L,
  spatial.resolution = 0.4,
  spatial.graph.mix = FALSE,
  spatial.constraint.mode = c("kmeans", "graph", "auto"),
  spatial.mode = c("standard", "population"),
  classifier = c("knn", "pls_lda"),
  backend = NULL,
  graph.feature.mode = "laplacian_self_tuning",
  graph.feature.components = 0L,
  graph.feature.steps = 3L,
  seed = 1234L,
  folds = 5L,
  visual.init = TRUE,
  progress = TRUE,
  progress.file = NULL,
  apply.kodama.dissimilarity = TRUE,
  return.graph = FALSE,
  ...
)
```

Arguments

indices	Integer matrix of neighbor indices, or a list with indices and distances.
---------	---

distances	Numeric matrix of neighbor distances.
data	Optional original data matrix. When supplied, KODAMA uses this float32 geometry for landmark selection and initial splitting, while reusing the supplied graph for KNN classification/projection and the final KODAMA graph. When omitted, a self-tuning graph Laplacian geometry is used.
spatial	Optional spatial or external coordinate matrix used only for constrained grouping when supplied.
samples	Optional sample or slide identifier used to separate spatial coordinate 1 before spatial graph and constraint construction.
W	Optional starting labels.
constrain	Optional group vector.
fix	Optional fixed-label mask.
M	Number of independent runs.
Tcycle	Number of optimization cycles per run.
ncomp	Number of graph features/components used by the PLS-LDA path.
landmarks	Maximum landmarks optimized directly.
splitting	Initial number of classes.
n.cores	CPU workers. For CUDA or Metal, 0 enables backend-specific automatic independent-run lane selection.
graph.neighbors	Number of neighbors retained in the returned graph.
knn.k	Number of neighbors used by the KNN classifier.
spatial.resolution	Resolution used when spatial constraints are derived inside the core.
spatial.graph.mix	Whether to combine spatial and feature graphs for grouping.
spatial.constraint.mode	Spatial constraint strategy.
spatial.mode	Coordinate treatment. "population" applies the classic repeated-coordinate regularization independently in every M run before spatial clustering.
classifier	Either "knn" or "pls_lda".
backend	Execution backend: "cpu", "cuda", or "metal".
graph.feature.mode	Graph-to-feature transform for PLS-LDA and graph-only initialization. The standard path is "laplacian_self_tuning".
graph.feature.components	Feature count; 0 uses ncomp.
graph.feature.steps	Power iterations used by graph feature extraction.
seed	Integer random seed.
folds	Number of cross-validation folds used by the classifier.

<code>visual.init</code>	Whether to propagate the PCA starts stored in a <code>KODAMA.graph</code> object.
<code>progress</code>	Whether the C++ core records progress.
<code>progress.file</code>	Optional progress-log path. When <code>progress=TRUE</code> , the native workers append M/Tcycle checkpoints here. A temporary path is created and announced when this is <code>NULL</code> .
<code>apply.kodama.dissimilarity</code>	Whether to return the KODAMA-corrected graph.
<code>return.graph</code>	<code>FALSE</code> omits the graph, <code>TRUE</code> materializes matrices, and "handle" returns a reusable external C++ graph pointer.
<code>...</code>	Reserved internal controls for reproducibility experiments.

Value

A `kodama_matrix` result with evolved labels, accuracy traces, diagnostics, timing, and the corrected graph when requested.

Examples

```
x <- as.matrix(iris[, 1:4])
g <- KODAMA.graph(x, k = 10, storage = "matrix", n.cores = 1)
fit <- KODAMA.matrix.graph(
  g$indices, g$distances, data = x, M = 1, Tcycle = 1,
  landmarks = 100, splitting = 10, knn.k = 5, n.cores = 1,
  return.graph = FALSE
)
fit$acc
```

`kodama_pca`

Backend-native float32 PCA

Description

Backend-native float32 PCA

Usage

```
kodama_pca(
  data,
  ncomp = 2L,
  center = TRUE,
  scale = FALSE,
  backend = NULL,
  n.cores = NULL,
  gpu.device = 0L,
  seed = 4L,
  oversample = NULL,
  power = NULL
```

```

)

KODAMA.pca(
  data,
  ncomp = 2L,
  center = TRUE,
  scale = FALSE,
  backend = NULL,
  n.cores = NULL,
  gpu.device = 0L,
  seed = 4L,
  oversample = NULL,
  power = NULL
)

```

Arguments

<code>data</code>	Numeric matrix with observations in rows.
<code>ncomp</code>	Number of principal components.
<code>center</code>	Whether to center columns.
<code>scale</code>	Whether to scale centered columns to unit sample standard deviation.
<code>backend</code>	Execution backend: "cpu", "cuda", or "metal".
<code>n.cores</code>	Number of worker threads. NULL uses <code>options(n.cores = ...)</code> , then <code>N_CORES</code> , and otherwise 1.
<code>gpu.device</code>	Accelerator device id.
<code>seed</code>	Random seed for the Gaussian subspace sketch.
<code>oversample</code>	Optional randomized-SVD oversampling width. NULL uses the standalone backend's automatic policy.
<code>power</code>	Optional randomized-SVD power count. NULL uses the standalone backend's automatic policy.

Value

A list containing scores, loadings, singular values, explained variance, preprocessing vectors, backend metadata, and runtime.

Examples

```

fit <- KODAMA.pca(as.matrix(iris[, 1:4]), ncomp = 2)
dim(fit$scores)

```

lymphoma	<i>Lymphoma gene-expression data</i>
----------	--------------------------------------

Description

Gene-expression profiles for diffuse large B-cell lymphoma, follicular lymphoma, and B-cell chronic lymphocytic leukemia.

Usage

```
data(lymphoma)
```

Format

A list with two elements:

data Numeric matrix with 62 rows and 4,026 columns.

class Factor with levels B-CLL, DLBCL, and FL.

Source

Distributed with the original KODAMA R package.

References

Alizadeh AA, Eisen MB, Davis RE, et al. (2000). Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling. *Nature* 403:503-511.

MetRef	<i>Nuclear Magnetic Resonance spectra of urine samples</i>
--------	--

Description

A longitudinal metabolomics cohort containing 873 urine spectra from 22 healthy donors. Each spectrum contains 450 bins.

Usage

```
data(MetRef)
```

Format

A list with three elements:

data Numeric matrix with 873 rows and 450 columns.

gender Character vector containing donor gender.

donor Character vector containing donor identifiers.

Source

Distributed with the original KODAMA R package.

References

Assfalg M, Bertini I, Colangiuli D, et al. (2008) Evidence of different metabolic phenotypes in humans. PNAS 105:1420-1424.

normalization	<i>Normalize samples</i>
---------------	--------------------------

Description

Compatibility implementation of the preprocessing utility exposed by the KODAMA R package. Samples are stored in rows and variables in columns.

Usage

```
normalization(
  Xtrain,
  Xtest = NULL,
  method = "pqn",
  ref = NULL,
  backend = NULL,
  n.cores = 1L,
  gpu.device = 0L
)
```

Arguments

Xtrain	Numeric training matrix.
Xtest	Optional numeric test matrix with the same variables as Xtrain.
method	Normalization method. One of "pqn", "sum", "median", "sqrt", or "none"; an unambiguous partial match is accepted.
ref	Optional reference spectrum for probabilistic quotient normalization. When omitted, the variable-wise training median is used.
backend	Native backend: "cpu", "cuda", "metal", or "auto".
n.cores	Number of CPU worker threads.
gpu.device	CUDA device identifier; Metal currently uses device zero.

Value

A list containing newXtrain and coeXtrain, and also newXtest and coeXtest when Xtest is supplied.

References

Dieterle F, Ross A, Schlotterbeck G, Senn H. Probabilistic quotient normalization as robust method to account for dilution of complex biological mixtures. *Analytical Chemistry* 78, 4281-4290 (2006).

See Also

[scaling\(\)](#)

Examples

```
x <- abs(as.matrix(iris[, 1:4])) + 1
fit <- normalization(x, method = "pqn")
dim(fit$newXtrain)
```

passing.message

Spatial message-passing preprocessing

Description

Reproduces the self-inclusive, distance-weighted passing.message preprocessing operation from KODAMAextra using the native exact 2D/3D grid search. The weighted sum is intentionally not normalized by its total weight.

Usage

```
passing.message(
  data,
  spatial,
  number_knn = 15L,
  samples = NULL,
  backend = NULL,
  n.cores = 4L,
  gpu.device = 0L
)
```

Arguments

data	Numeric expression matrix with observations in rows.
spatial	Numeric matrix containing two or three coordinates per row.
number_knn	Number of self-inclusive spatial nearest neighbors.
samples	Optional sample, tissue, or slide vector. Neighbor search and distance scaling are performed independently for every distinct value.
backend	Native backend: "cpu", "cuda", "metal", or "auto".
n.cores	Number of CPU worker threads.
gpu.device	CUDA device identifier; Metal currently uses device zero.

Value

A numeric matrix with the dimensions and dimnames of data. Timing, backend, precision, and sample-group metadata are attached as attributes.

Examples

```
set.seed(1)
x <- matrix(rnorm(40), 20, 2)
xy <- matrix(runif(40), 20, 2)
dim(passing.message(x, xy, number_knn = 5, n.cores = 1))
```

PLSLDACV

*Cross-validated SIMPLS + PLS-LDA classification***Description**

Cross-validated SIMPLS + PLS-LDA classification

Usage

```
PLSLDACV(
  data,
  labels,
  constrain = NULL,
  folds = 10L,
  stratified = TRUE,
  seed = 1L,
  ncomp = min(50L, ncol(data)),
  center = TRUE,
  scale = TRUE,
  backend = NULL,
  n.cores = 1L,
  gpu.device = 0L
)
```

Arguments

data	Numeric matrix with samples in rows and variables in columns.
labels	Class labels used as the cross-validation truth.
constrain	Optional integer vector assigning samples to indivisible fold groups.
folds	Number of cross-validation folds.
stratified	Whether to stratify folds by class labels.
seed	Integer random seed for fold construction.
ncomp	Number of SIMPLS latent components requested.
center	Whether the C++ core centers the analysis matrix.

scale	Whether the C++ core scales the analysis matrix.
backend	Execution backend: "cpu", "cuda", or "metal".
n.cores	Number of CPU worker threads requested by the wrapper.
gpu.device	CUDA device id when backend = "cuda".

Value

A cross-validation result containing predictions, truth, fold assignments, fold and global accuracy, confusion matrix, timing, and backend diagnostics.

Examples

```
x <- as.matrix(iris[, 1:4])
y <- as.integer(iris$Species)
fit <- PLSLDACV(x, y, folds = 3, ncomp = 2, backend = "cpu")
fit$accuracy
```

RunFastPCA

*Fast native PCA for matrices and single-cell containers***Description**

Runs the standalone float32 randomized PCA and stores its scores directly in supported single-cell containers. Expression assays are transposed internally because their rows are features and their columns are cells.

Usage

```
RunFastPCA(object, ...)

## Default S3 method:
RunFastPCA(object, features = NULL, ...)

## S3 method for class 'SingleCellExperiment'
RunFastPCA(
  object,
  assay.type = NULL,
  features = NULL,
  nfeatures = NULL,
  ncomp = 50L,
  reduction.name = "PCA",
  ...
)

## S3 method for class 'SpatialExperiment'
RunFastPCA(
  object,
```

```

    assay.type = NULL,
    features = NULL,
    nfeatures = NULL,
    ncomp = 50L,
    reduction.name = "PCA",
    ...
)

## S3 method for class 'Seurat'
RunFastPCA(
  object,
  assay = NULL,
  layer = "data",
  features = NULL,
  nfeatures = NULL,
  ncomp = 50L,
  reduction.name = "pca",
  ...
)

## S3 method for class 'giotto'
RunFastPCA(
  object,
  values = NULL,
  features = NULL,
  nfeatures = NULL,
  ncomp = 50L,
  reduction.name = "pca",
  ...
)

```

Arguments

object	Numeric matrix, SingleCellExperiment, SpatialExperiment, Seurat object, Giotto object, or list of Seurat objects.
...	Arguments passed to kodama_pca() .
features	Optional feature names, indices, or logical selector.
nfeatures	Number of top features from a preceding SpatialFeatureSelection() call. Cannot be combined with features.
assay.type	Assay to use. NULL prefers logcounts, then normcounts, then counts, and finally the first available assay.
ncomp	Number of principal components.
reduction.name	Name used to store PCA scores.
assay	Seurat assay containing the features.
layer	Seurat assay layer, normally "data".
values	Giotto expression-values selection. NULL uses the object's default expression values.

Value

A native PCA result for a matrix input, otherwise the updated container with a PCA dimensional reduction.

Examples

```
x <- matrix(rnorm(120), 30, 4)
fit <- RunFastPCA(x, ncomp = 2, backend = "cpu")
dim(fit$scores)
```

RunKODAMAggraph	<i>Run KODAMA graph construction on matrices and single-cell containers</i>
-----------------	---

Description

The object methods extract an existing dimensional reduction, construct one reusable KODAMA graph, and store it in the container. The graph does not retain the original expression matrix.

Usage

```
RunKODAMAggraph(object, ...)

## Default S3 method:
RunKODAMAggraph(object, ...)

## S3 method for class 'SingleCellExperiment'
RunKODAMAggraph(
  object,
  reduction = "PCA",
  dims = 50L,
  graph.name = "KODAMA",
  ...
)

## S3 method for class 'SpatialExperiment'
RunKODAMAggraph(
  object,
  reduction = "PCA",
  dims = 50L,
  graph.name = "KODAMA",
  use.spatial = TRUE,
  sample.column = "sample_id",
  ...
)

## S3 method for class 'Seurat'
```

```

RunKODAMAggraph(
  object,
  reduction = "pca",
  dims = 50L,
  use.spatial = TRUE,
  reduction.save = "KODAMA",
  ...
)

## S3 method for class 'giotto'
RunKODAMAggraph(
  object,
  reduction = "pca",
  dims = 50L,
  use.spatial = TRUE,
  reduction.save = "KODAMA",
  ...
)

```

Arguments

<code>object</code>	Numeric matrix, SingleCellExperiment, SpatialExperiment, Seurat object, Giotto object, or list of Seurat objects.
<code>...</code>	Arguments passed to KODAMA.graph() .
<code>reduction</code>	Name of the input dimensional reduction.
<code>dims</code>	Number of leading dimensions, or an integer vector selecting dimensions.
<code>graph.name</code>	Name under which the reusable graph is stored.
<code>use.spatial</code>	Whether spatial coordinates should constrain a spatial container analysis.
<code>sample.column</code>	Column in <code>colData</code> identifying slides or samples.
<code>reduction.save</code>	Name used to store KODAMA state and the final visualization.

Value

A graph for a matrix input, otherwise the updated container.

Examples

```

x <- as.matrix(iris[, 1:4])
g <- RunKODAMAggraph(x, k = 5, n.cores = 1)
g$parameters

```

RunKODAMAmatrix

Run KODAMA optimization on matrices and single-cell containers

Description

Object methods reuse a graph previously stored by [RunKODAMAggraph\(\)](#) unless `reuse.graph = FALSE`. The optimization result is stored as KODAMA state; the actual reduced dimension is created by [RunKODAMAvsualization\(\)](#).

Usage

```
RunKODAMAmatrix(object, ...)

## Default S3 method:
RunKODAMAmatrix(object, graph = NULL, ...)

## S3 method for class 'SingleCellExperiment'
RunKODAMAmatrix(
  object,
  reduction = "PCA",
  dims = 50L,
  graph.name = "KODAMA",
  reuse.graph = TRUE,
  ...
)

## S3 method for class 'SpatialExperiment'
RunKODAMAmatrix(
  object,
  reduction = "PCA",
  dims = 50L,
  graph.name = "KODAMA",
  use.spatial = TRUE,
  sample.column = "sample_id",
  reuse.graph = TRUE,
  ...
)

## S3 method for class 'Seurat'
RunKODAMAmatrix(
  object,
  reduction = "pca",
  dims = 50L,
  use.spatial = TRUE,
  reduction.save = "KODAMA",
  reuse.graph = TRUE,
  ...
)
```

```

)

## S3 method for class 'giotto'
RunKODAMAmatrix(
  object,
  reduction = "pca",
  dims = 50L,
  use.spatial = TRUE,
  reduction.save = "KODAMA",
  reuse.graph = TRUE,
  ...
)

```

Arguments

<code>object</code>	Numeric matrix, SingleCellExperiment, SpatialExperiment, Seurat object, Giotto object, or list of Seurat objects.
<code>...</code>	Arguments passed to <code>KODAMA.matrix()</code> .
<code>graph</code>	Optional graph supplied for a matrix input.
<code>reduction</code>	Name of the input dimensional reduction.
<code>dims</code>	Number of leading dimensions, or an integer vector selecting dimensions.
<code>graph.name</code>	Name under which the reusable graph is stored.
<code>reuse.graph</code>	Reuse a graph stored in the object when available.
<code>use.spatial</code>	Whether spatial coordinates should constrain a spatial container analysis.
<code>sample.column</code>	Column in <code>colData</code> identifying slides or samples.
<code>reduction.save</code>	Name used to store KODAMA state and the final visualization.

Value

A KODAMA result for a matrix input, otherwise the updated container.

Examples

```

x <- as.matrix(iris[, 1:4])
fit <- RunKODAMAmatrix(
  x, M = 1, Tcycle = 1, landmarks = 100,
  splitting = 10, knn.k = 5, n.cores = 1,
  return.graph = FALSE
)
fit$acc

```

`RunKODAMAvisualization`*Visualize KODAMA results stored in single-cell containers*

Description

Visualize KODAMA results stored in single-cell containers

Usage

```
RunKODAMAvisualization(object, ...)

## Default S3 method:
RunKODAMAvisualization(object, ...)

## S3 method for class 'SingleCellExperiment'
RunKODAMAvisualization(
  object,
  reduction = "PCA",
  dims = 50L,
  graph.name = "KODAMA",
  reduction.save = "KODAMA",
  ...
)

## S3 method for class 'SpatialExperiment'
RunKODAMAvisualization(
  object,
  reduction = "PCA",
  dims = 50L,
  graph.name = "KODAMA",
  reduction.save = "KODAMA",
  ...
)

## S3 method for class 'Seurat'
RunKODAMAvisualization(
  object,
  reduction = "pca",
  dims = 50L,
  reduction.save = "KODAMA",
  ...
)

## S3 method for class 'giotto'
RunKODAMAvisualization(
  object,
```

```

    reduction = "pca",
    dims = 50L,
    reduction.save = "KODAMA",
    ...
)

```

Arguments

object	Numeric matrix, SingleCellExperiment, SpatialExperiment, Seurat object, Giotto object, or list of Seurat objects.
...	Arguments passed to <code>KODAMA.visualization()</code> .
reduction	Name of the input dimensional reduction.
dims	Number of leading dimensions, or an integer vector selecting dimensions.
graph.name	Name under which the reusable graph is stored.
reduction.save	Name used to store KODAMA state and the final visualization.

Value

A matrix for a direct KODAMA input, otherwise the updated container with a genuine KODAMA reduced dimension.

Examples

```

x <- as.matrix(iris[, 1:4])
fit <- RunKODAMAvisualization(
  x, method = "UMAP", k = 5, n.epochs = 5, n.cores = 1
)
dim(fit)

```

RunSpatialFeatureSelection

Spatial feature selection for matrices and spatial containers

Description

Extracts expression values, coordinates, and slide identities from supported containers, then calls the multicore CPU `spatial_feature_selection()` implementation. Expression values are passed without normalization or transformation.

Usage

```
RunSpatialFeatureSelection(object, ...)
```

```
## Default S3 method:
```

```
RunSpatialFeatureSelection(object, spatial, samples = NULL, ...)
```

```

## S3 method for class 'SpatialExperiment'
RunSpatialFeatureSelection(
  object,
  assay.type = "logcounts",
  sample.column = "sample_id",
  samples = NULL,
  ...
)

## S3 method for class 'Seurat'
RunSpatialFeatureSelection(
  object,
  assay = NULL,
  layer = "data",
  samples = NULL,
  ...
)

## S3 method for class 'giotto'
RunSpatialFeatureSelection(
  object,
  values = NULL,
  sample.column = NULL,
  samples = NULL,
  ...
)

SpatialFeatureSelection(object, ...)

```

Arguments

object	Numeric matrix, SpatialExperiment, Seurat object, Giotto object, or list of Seurat objects.
...	Additional arguments passed to spatial_feature_selection() .
spatial	Coordinate matrix for a matrix input.
samples	Optional slide or sample identifier for each observation.
assay.type	Name of the SpatialExperiment assay to screen.
sample.column	Column in colData identifying slides or samples.
assay	Seurat assay containing the features to screen.
layer	Seurat assay layer, normally "data".
values	Giotto expression-values selection. NULL uses the object's default expression values.

Value

A `kodama_spatial_features` result for a matrix input. Container methods return the updated object with statistics in feature metadata.

Examples

```
set.seed(1)
x <- matrix(rnorm(120), 30, 4)
xy <- matrix(runif(60), 30, 2)
fit <- RunSpatialFeatureSelection(x, spatial = xy, n.cores = 1)
head(fit$ranking)
```

scaling	<i>Scale variables</i>
---------	------------------------

Description

Compatibility implementation of the scaling utility exposed by the KODAMA R package. Training statistics are reused for an optional test matrix.

Usage

```
scaling(
  Xtrain,
  Xtest = NULL,
  method = "autoscaling",
  backend = NULL,
  n.cores = 1L,
  gpu.device = 0L
)
```

Arguments

Xtrain	Numeric training matrix.
Xtest	Optional numeric test matrix with the same variables as Xtrain.
method	Scaling method. One of "none", "centering", "autoscaling", "rangescaling", or "paretoscaling"; an unambiguous partial match is accepted.
backend	Native backend: "cpu", "cuda", "metal", or "auto".
n.cores	Number of CPU worker threads.
gpu.device	CUDA device identifier; Metal currently uses device zero.

Value

A list containing newXtrain, and also newXtest when Xtest is supplied.

References

van den Berg RA, Hoefsloot HCJ, Westerhuis JA, et al. Centering, scaling, and transformations: improving the biological information content of metabolomics data. *BMC Genomics* 7, 142 (2006).

See Also

`normalization()`

Examples

```
fit <- scaling(as.matrix(iris[, 1:4]), method = "autoscaling")
dim(fit$newXtrain)
```

spatial_feature_selection

Fast multi-slide spatial feature screening

Description

Independently screens variables with low-rank spatial covariance projections spanning linear, smooth, and periodic coordinate bases. Basis-level evidence is combined with the Cauchy rule; slides are processed independently. This implementation was written independently and does not use SPARK-X code.

Usage

```
spatial_feature_selection(
  data,
  spatial,
  samples = NULL,
  n.cores = NULL,
  require.nonzero.each.sample = TRUE
)
```

```
KODAMA.spatial.features(
  data,
  spatial,
  samples = NULL,
  n.cores = NULL,
  require.nonzero.each.sample = TRUE
)
```

Arguments

data	Numeric matrix with observations in rows and variables in columns. Sparse and delayed matrix-like inputs are converted to dense float32 work buffers in bounded feature batches rather than densified in full.
spatial	Two- or three-column spatial-coordinate matrix.
samples	Optional slide/sample vector. Statistics are calculated independently within each value.

`n.cores` Number of CPU threads. NULL uses `options(n.cores = ...)`, then `N_CORES`, and otherwise 4.

`require.nonzero.each.sample`
 Exclude variables that are identically zero in any slide, matching the filtering convention of `multi_SPARKX()`.

Value

A list with rankings, scores, raw and BH-adjusted p-values, per-slide results, basis/statistic timings, resolved core count, and backend metadata.

Examples

```
set.seed(1)
x <- matrix(rnorm(120), 30, 4)
xy <- matrix(runif(60), 30, 2)
fit <- spatial_feature_selection(x, xy, n.cores = 1)
head(fit$ranking)
```

<code>spirals</code>	<i>Generate spiral clusters</i>
----------------------	---------------------------------

Description

Produces two-dimensional spiral clusters using the construction from the KODAMA R package.

Usage

```
spirals(n = c(100, 100, 100), sd = c(0, 0, 0))
```

Arguments

`n` Integer vector giving the number of points in each spiral.

`sd` Numeric vector giving the radial noise standard deviation for each spiral.

Value

A `sum(n)` by 2 numeric matrix with columns `x` and `y`.

See Also

[dinisurface\(\)](#), [helicoid\(\)](#), [swissroll\(\)](#)

Examples

```
dim(spirals(c(50, 50)))
```

swissroll	<i>Generate a Swiss roll</i>
-----------	------------------------------

Description

Generates the three-dimensional Swiss roll used by the KODAMA R package.

Usage

```
swissroll(N = 1000)
```

Arguments

N Number of data points.

Value

An N by 3 numeric matrix with columns x, y, and z.

See Also

[dinisurface\(\)](#), [helicoid\(\)](#), [spirals\(\)](#)

Examples

```
dim(swissroll(100))
```

USA	<i>State of the Union data</i>
-----	--------------------------------

Description

TF-IDF representation of 86 spoken State of the Union addresses from 1900 through 2014, using 834 retained words.

Usage

```
data(USA)
```

Format

A list with three elements:

data Numeric TF-IDF matrix with 86 rows and 834 columns.

year Numeric vector containing speech years.

president Character vector containing president names.

Source

Distributed with the original KODAMA R package.

Index

- * **datasets**
 - lymphoma, [20](#)
 - MetRef, [20](#)
 - USA, [36](#)
- CoreKNN, [3](#)
- CorePLSLDA, [4](#)
- dinisphere, [5](#)
- dinisphere(), [6](#), [35](#), [36](#)
- helicoid, [6](#)
- helicoid(), [5](#), [35](#), [36](#)
- KNNCV, [6](#)
- KODAMA.clustering, [7](#)
- KODAMA.diagnostics, [8](#)
- KODAMA.graph, [9](#)
- KODAMA.graph(), [27](#)
- KODAMA.graph.materialize, [10](#)
- KODAMA.makeSNNGraph (KODAMA.graph), [9](#)
- KODAMA.matrix (kodama_matrix), [13](#)
- KODAMA.matrix(), [29](#)
- KODAMA.matrix.graph
 - (kodama_matrix_graph), [16](#)
- KODAMA.pca (kodama_pca), [18](#)
- KODAMA.spatial.features
 - (spatial_feature_selection), [34](#)
- KODAMA.timing, [11](#)
- KODAMA.visualization, [11](#)
- KODAMA.visualization(), [31](#)
- KODAMA_backend, [13](#)
- kodama_diagnostics
 - (KODAMA.diagnostics), [8](#)
- kodama_matrix, [13](#)
- kodama_matrix_graph, [16](#)
- kodama_pca, [18](#)
- kodama_pca(), [25](#)
- kodama_timing (KODAMA.timing), [11](#)
- lymphoma, [20](#)
- makeSNNGraph (KODAMA.graph), [9](#)
- MetRef, [20](#)
- normalization, [21](#)
- normalization(), [34](#)
- passing.message, [22](#)
- PLSLDACV, [23](#)
- RunFastPCA, [24](#)
- RunKODAMAGraph, [26](#)
- RunKODAMAGraph(), [28](#)
- RunKODAMAmatrix, [28](#)
- RunKODAMAvizualization, [30](#)
- RunKODAMAvizualization(), [28](#)
- RunSpatialFeatureSelection, [31](#)
- scaling, [33](#)
- scaling(), [22](#)
- spatial_feature_selection, [34](#)
- spatial_feature_selection(), [31](#), [32](#)
- SpatialFeatureSelection
 - (RunSpatialFeatureSelection), [31](#)
- spirals, [35](#)
- spirals(), [5](#), [6](#), [36](#)
- swissroll, [36](#)
- swissroll(), [5](#), [6](#), [35](#)
- USA, [36](#)