Package 'netOP'

September 27, 2026

Title Network Data Operations and Overlapping Partitions Based Methods for Large Networks

Version 0.1.2

Description Implements methods for generating, embedding, and clustering random networks and for estimating and selecting statistical network models.
Provides SONNET (Subsampling ON NETwork), a scalable subsampling-based divide-and-conquer method for community detection described by Chakrabarty, Sengupta and Chen (2025) <doi:10.5705/ss.202022.0108>, and NETCROP (NETwork CRoss-validation using Overlapping Partitions), an overlapping-partition framework for network cross-validation, model selection, and regularization tuning described by Chakrabarty, Sengupta and Chen (2026) <doi:10.48550/arXiv.2504.06903>. Also includes spectral and latent-space methods, loss functions, and helper functions for statistical analysis of network data.

License GPL (>= 2)

URL https://github.com/sayan-ch/netOP, https://sayan-ch.github.io

BugReports https://github.com/sayan-ch/netOP/issues

Encoding UTF-8

Depends R (>= 4.1.0)

Imports cluster, irlba, Matrix, methods, Rcpp, RSpectra, tibble

LinkingTo Rcpp, RcppEigen

Suggests entropy, future, future.apply, ggplot2, knitr, peakRAM, pkgdown, ps, rmarkdown, testthat (>= 3.0.0), withr

VignetteBuilder knitr

RoxygenNote 8.0.0

Config/testthat/edition 3

Copyright 2026 Sayan Chakrabarty

NeedsCompilation yes

Author Sayan Chakrabarty [aut, cre] (ORCID:
<https://orcid.org/0000-0003-2372-2076>)

Maintainer Sayan Chakrabarty <sayanc@umich.edu>

Repository CRAN

Date/Publication 2026-09-27 16:40:08 UTC

Contents

`absolute_error_losses` *Compute absolute-error losses*

Description

`sae()` returns the sum of absolute errors; `mae()` returns their mean.

Usage

```
sae(x, y, na_rm = FALSE, validate_inputs = TRUE)

mae(x, y, na_rm = FALSE, validate_inputs = TRUE)
```

Arguments

x, y	Equal-length numeric vectors.
na_rm	Remove missing comparisons.
validate_inputs	Validate input types, lengths, and na_rm.

Value

A nonnegative numeric scalar.

See Also

sse(), mse()

Examples

```
sae(c(1, 2), c(1, 3))
mae(c(1, 2), c(1, 3))
```

---

adjacency_neighbors *Find adjacent nodes*

Description

Builds adjacency lists without densifying sparse inputs. Nonzero finite entries represent edges. adjacency_neighbors() returns node indices; adjacency_weighted_neighbors() also returns edge weights. For undirected reciprocal weighted edges, the smaller nonzero weight is used.

Usage

```
adjacency_neighbors(A, directed = FALSE, self_loops = c("ignore", "include"))

adjacency_weighted_neighbors(
  A,
  directed = FALSE,
  self_loops = c("ignore", "include")
)
```

Arguments

A	A finite square dense or sparse adjacency matrix.
directed	Preserve edge direction; otherwise an edge in either direction connects both nodes.
self_loops	Include or ignore diagonal edges.

Value

adjacency_neighbors() returns one integer neighbor vector per node. adjacency_weighted_neighbors() returns parallel nodes and weights lists.

See Also

connected_components(), shortest_path_distances()

Examples

```
A <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
adjacency_neighbors(A)
adjacency_weighted_neighbors(A)
```

---

apply_average_degree_scaling

Apply the selected average-degree scaling method

Description

Dispatches to the calibrated or historical probability-scaling rule.

Usage

```
apply_average_degree_scaling(
  P,
  average_degree,
  average_degree_method = "naive",
  self_loops,
  lower_clip,
  upper_clip
)
```

Arguments

P	Dense or sparse edge-probability matrix.
average_degree	Target expected average row degree.
average_degree_method	Scaling method, either "naive" or "calibrated".
self_loops	Whether diagonal probabilities count toward degree.
lower_clip, upper_clip	Finite probability clipping bounds.

Value

A list containing the scaled probability matrix P and multiplier.

See Also

scale_to_average_degree(), scale_lsm_to_average_degree()

Examples

```
P <- matrix(0.1, 200, 200)
diag(P) <- 0
apply_average_degree_scaling(
  P, average_degree = 5, average_degree_method = "naive",
  self_loops = FALSE, lower_clip = 0, upper_clip = 1
)$multiplier
```

---

ase *Compute an adjacency spectral embedding*

Description

Embeds a dense or sparse adjacency matrix in d dimensions.

Usage

```
ase(
  A,
  d,
  directed = FALSE,
  reconstruct = FALSE,
  align_with = NULL,
  ram_check = FALSE
)
```

Arguments

A	A finite square dense or sparse adjacency matrix.
d	Positive embedding dimension.
directed	Whether to treat A as directed.
reconstruct	Whether to return the rank-d matrix reconstruction.
align_with	Optional nrow(A)-by-d reference embedding used for orthogonal Procrustes alignment.
ram_check	Whether to report a conservative RAM preflight estimate.

Details

For an undirected A, the embedding uses the d eigenvalues of largest magnitude and forms $\hat{Z} = U|\Lambda|^{1/2}$. Signed eigenvalues are retained so reconstruction uses $U\Lambda U^T$. Directed input uses a rank-d SVD to produce left and right embeddings. A supplied align_with rotates both embeddings together, preserving their reconstructed matrix.

Value

An embedding object containing coordinates and decomposition details.

See Also

spectral_cluster(), generate_rdpg(), netcrop_rdpg()

Examples

```
A <- generate_rdpg(n = 200, d = 3, seed = 8, ncores = 1)
fit <- ase(A, d = 3)
dim(fit$Z_hat)
```

---

auc_losses *Compute AUC scores and losses*

Description

auc() computes binary ROC AUC using average ranks for tied scores and a compiled implementation when available. auc_as_loss() returns one minus AUC so smaller values are better.

Usage

```
auc(A, P, use_cpp = TRUE, validate_inputs = TRUE)

auc_as_loss(A, P, use_cpp = TRUE, validate_inputs = TRUE)
```

Arguments

A	Binary response vector.
P	Numeric prediction-score vector.
use_cpp	Try the registered compiled implementation before the R implementation.
validate_inputs	Validate binary responses and finite scores.

Value

A numeric scalar containing AUC or one minus AUC.

See Also

bin_dev(), bin_dev_mean()

Examples

```
auc(c(0, 1, 1), c(0.1, 0.7, 0.8), use_cpp = FALSE)
auc_as_loss(c(0, 1, 1), c(0.1, 0.7, 0.8), use_cpp = FALSE)
```

`binary_deviance_losses` *Compute binary deviance losses*

Description

`bin_dev()` returns summed binary cross-entropy after clipping predictions; despite its historical name it does not multiply by two. `bin_dev_mean()` averages over retained response-probability pairs.

Usage

```
bin_dev(x, y, epsilon = 1e-05, na_rm = TRUE, validate_inputs = TRUE)

bin_dev_mean(x, y, epsilon = 1e-05, na_rm = TRUE, validate_inputs = TRUE)
```

Arguments

`x`	Binary response vector.
`y`	Numeric predicted-probability vector.
`epsilon`	Finite clipping constant strictly between zero and one half.
`na_rm`	Remove comparisons containing missing values.
`validate_inputs`	Validate types, lengths, responses, and predictions.

Value

A nonnegative sum from `bin_dev()` or mean from `bin_dev_mean()`.

See Also

`auc()`, `clip_probabilities()`

Examples

```
bin_dev(c(0, 1), c(0.1, 0.8))
bin_dev_mean(c(0, 1), c(0.1, 0.8))
```

clip_probabilities *Clip probability values safely*

Description

Restricts probabilities to `[eps, 1 - eps]` to protect logarithms and inverse-link calculations.

Usage

```
clip_probabilities(P, eps = 1e-06)
```

Arguments

`P`	Numeric probability vector or matrix.
`eps`	Finite clipping constant strictly between zero and one half.

Value

An object shaped like `P` with clipped probabilities.

See Also

`clip_values()`, `bin_dev()`

Examples

```
clip_probabilities(c(0, 0.5, 1), eps = 0.01)
```

clip_values *Clip numeric values to an interval*

Description

Restricts each value to inclusive bounds while preserving missing values, names, dimensions, and dimension names.

Usage

```
clip_values(x, lower_clip = -Inf, upper_clip = Inf)
```

Arguments

`x`	Numeric vector or matrix.
`lower_clip`, `upper_clip`	Inclusive clipping bounds.

Value

An object shaped like x with clipped values.

See Also

clip_probabilities()

Examples

```
clip_values(c(-1, 0.5, 2), 0, 1)
```

---

connected_components *Find connected components*

Description

Finds weak/undirected components: an edge in either direction connects two nodes, and weights do not affect membership. Components and their nodes are returned in breadth-first discovery order. largest_connected_component() selects the first largest component and extracts its induced matrix.

Usage

```
connected_components(A, self_loops = c("ignore", "include"), use_cpp = TRUE)

largest_connected_component(
  A,
  self_loops = c("ignore", "include"),
  use_cpp = TRUE,
  sort_nodes = TRUE
)
```

Arguments

A	A finite square dense or sparse adjacency matrix.
self_loops	Include or ignore diagonal edges.
use_cpp	Try a registered compiled implementation before the R implementation.
sort_nodes	Sort selected indices in largest_connected_component().

Value

connected_components() returns a named list of one-based node vectors. largest_connected_component() returns nodes, size, and the induced submatrix.

See Also

adjacency_neighbors(), shortest_path_distances()

Examples

```
A <- matrix(c(0, 1, 0, 1, 0, 0, 0, 0, 0), 3, 3)
connected_components(A, use_cpp = FALSE)
largest_connected_component(A, use_cpp = FALSE)
```

`dkest_tune_regularizer` *Tune a degree-regularized spectral model*

Description

Selects a spectral regularizer using the DKEST eigenspectral-ratio criterion.

Usage

```
dkest_tune_regularizer(
  A,
  K,
  tau_candidates,
  use_laplacian = TRUE,
  use_dcbm = TRUE,
  dcbm_est_method = c("plugin", "spectral"),
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
  seed = NULL,
  verbose = TRUE,
  force_windows = FALSE,
  ram_check = FALSE,
  failure_handling = c("stop", "omit"),
  retain_intermediates = c("all", "minimal")
)
```

Arguments

`A`	Finite symmetric, loop-free adjacency matrix.
`K`	Fixed positive number of communities.
`tau_candidates`	Unique finite nonnegative regularization candidates.
`use_laplacian`	Use a normalized graph Laplacian.
`use_dcbm`	Fit a degree-corrected rather than ordinary block model.
`dcbm_est_method`	DCBM estimator, either `"plugin"` or `"spectral"`.
`ncores`	Positive worker count.
`seed`	Optional nonnegative reproducibility seed.
`verbose`	Print progress messages.
`force_windows`	Use the Windows-compatible parallel backend.

`ram_check` Report conservative RAM demand.

`failure_handling`
Stop or omit failed candidates.

`retain_intermediates`
Retain all or minimal candidate fits.

Details

Each candidate regularizes the observed matrix, optionally constructs its normalized Laplacian, clusters the fixed-K embedding, estimates SBM or DCBM probabilities, and computes the ratio between selected residual and fitted eigenvalues. Candidate failures are audited explicitly.

Value

A `netcrop_regularizer` object containing `tau_hat`, the selected DK statistic, per-candidate numerator, denominator, and ratio, diagnostics, seeds, worker counts, timing, RAM information, and optional raw fits.

See Also

`netcrop_tune_regularizer()`, `mult_reg_spectral_cluster()`

Examples

```
A <- generate_sbm(n = 200, K = 3, alpha = 0.5, beta = 0.1,
                  seed = 35, ncores = 1)
dkest_tune_regularizer(A, K = 3, tau_candidates = c(0, 0.1),
                       ncores = 1, seed = 36, verbose = FALSE)
```

`ecv_stability` *Edge cross-validation stability selection*

Description

Selects a block-model community count or RDPG dimension by repeated edge sampling. netOP provides self-contained wrappers around an ECV implementation derived from the CRAN package `randnet`; `randnet` is not a package dependency and its ECV-specific helpers remain internal.

References

Li, T., Levina, E., and Zhu, J. (2020). Network cross-validation by edge sampling. *Biometrika*, 107(2), 257-276. doi:10.1093/biomet/asaa006

`ecv_stability_blockmodel` *Select block-model size with edge cross-validation*

Description

Runs repeated edge-sampling cross-validation over every community count from one through `max_K` using the preserved ECV block-model implementation.

Usage

```
ecv_stability_blockmodel(
  A,
  max_K,
  train_proportion = 0.9,
  cv = 3L,
  nrep = 20L,
  tau = 0,
  losses = "sse",
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
  seed = NULL,
  verbose = TRUE,
  force_windows = FALSE,
  ram_check = FALSE,
  failure_handling = c("stop", "omit"),
  retain_intermediates = c("all", "minimal")
)
```

Arguments

`A`	Binary, symmetric, loop-free adjacency matrix.
`max_K`	Maximum candidate community count; candidates are `1:max_K`.
`train_proportion`	Proportion of dyads retained for training.
`cv`	Positive number of edge-sampling folds.
`nrep`	Positive number of repeated ECV runs.
`tau`	Nonnegative block-model regularization value.
`losses`	Canonical loss names to evaluate.
`ncores`	Positive outer worker count; folds run sequentially.
`seed`	Optional nonnegative reproducibility seed.
`verbose`	Print progress messages.
`force_windows`	Use the Windows-compatible parallel backend.
`ram_check`	Report conservative RAM demand.

`failure_handling`
: Stop or omit failed repetitions.

`retain_intermediates`
: Retain all or minimal legacy results.

Details

The wrapper validates binary undirected loop-free input, candidate and sampling feasibility, dependencies, seeds, workers, and losses. Repetitions run through `uni_mclapply()` while folds within each repetition remain sequential. Raw ECV results are audited and converted to the shared tidy loss schema; failed repetitions either stop or are explicitly omitted.

Value

A `netcrop_blockmodel` object with `algorithm = "ECV"`, tidy losses, selections, candidate metadata, realized seeds, failures, worker counts, timing, RAM diagnostics, and optional raw output.

References

Li, T., Levina, E., and Zhu, J. (2020). Network cross-validation by edge sampling. *Biometrika*, 107(2), 257-276. doi:10.1093/biomet/asaa006

See Also

`ecv_stability_rdpg()`, `ncv_stability_blockmodel()`, `netcrop_blockmodel()`

Examples

```
A <- generate_sbm(n = 200, K = 3, alpha = 0.5, beta = 0.1,
                  seed = 6, ncores = 1)
ecv_stability_blockmodel(A, max_K = 5, cv = 2, nrep = 1,
                         ncores = 1, seed = 7, verbose = FALSE)
```

---

ecv_stability_rdpg *Select RDPG dimension with edge cross-validation*

Description

Runs repeated edge-sampling cross-validation over every symmetric-RDPG dimension from one through `max_d`.

Usage

```
ecv_stability_rdpg(
  A,
  max_d,
  cv = 3L,
  nrep = 1L,
  train_proportion = 0.9,
  losses = c("mse", "bin_dev", "auc_as_loss"),
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
  seed = NULL,
  verbose = TRUE,
  force_windows = FALSE,
  ram_check = FALSE,
  failure_handling = c("stop", "omit"),
  retain_intermediates = c("all", "minimal")
)
```

Arguments

`A`	Binary, symmetric, loop-free adjacency matrix.
`max_d`	Maximum candidate dimension; candidates are `1:max_d`.
`cv`	Positive number of edge-sampling folds.
`nrep`	Positive number of repeated ECV runs.
`train_proportion`	Proportion of dyads retained for training.
`losses`	Canonical loss names to evaluate: `"mse"`, `"sse"`, `"bin_dev"`, and `"auc_as_loss"`.
`ncores`	Positive outer worker count; folds run sequentially.
`seed`	Optional nonnegative reproducibility seed.
`verbose`	Print progress messages.
`force_windows`	Use the Windows-compatible parallel backend.
`ram_check`	Report conservative RAM demand.
`failure_handling`	Stop or omit failed repetitions.
`retain_intermediates`	Retain all or minimal legacy results.

Details

The preserved RDPG ECV reconstruction routines run inside an isolated internal environment. The wrapper validates graph and holdout feasibility, supplies namespace-safe dependencies, audits all results, converts them to tidy losses, and retains independently realized repetition seeds.

Value

A `netcrop_rdpg` object with `algorithm = "ECV"`, tidy loss curves, selections, holdout metadata, realized seeds, failures, worker counts, timings, RAM diagnostics, and optional legacy output.

References

Li, T., Levina, E., and Zhu, J. (2020). Network cross-validation by edge sampling. *Biometrika*, 107(2), 257-276. doi:10.1093/biomet/asaa006

See Also

`ecv_stability_blockmodel()`, `netcrop_rdpg()`

Examples

```
A <- as.matrix(generate_rdpg(n = 200, d = 3, average_degree = 8,
                             seed = 33, ncores = 1))
ecv_stability_rdpg(A, max_d = 5, cv = 2, nrep = 1,
                   ncores = 1, seed = 34, verbose = FALSE)
```

`eig_decomp` *Compute an eigendecomposition*

Description

Computes selected eigenpairs of a finite symmetric matrix. Partial decomposition falls back to `base::eigen()` unless `force_engine = TRUE`. The `"irlba"` option is accepted for compatibility but redirected because an SVD is not a signed eigendecomposition.

Usage

```
eig_decomp(
  A,
  d,
  only_values = FALSE,
  scale_by = c("none", "dimension", "sparsity"),
  use_laplacian = FALSE,
  engine = c("rspectra", "base", "irlba"),
  force_engine = FALSE,
  order_by = c("magnitude", "value"),
  safe_d_multiplier = 1,
  validate_inputs = TRUE
)
```

Arguments

`A`	A finite symmetric square dense or sparse matrix.
`d`	Positive number of eigencomponents to return.
`only_values`	Return eigenvalues without eigenvectors.
`scale_by`	Scale by `"none"`, matrix `"dimension"`, or entry `"sparsity"`.

use_laplacian	Transform A with graph_laplacian() first.
engine	Decomposition backend.
force_engine	Stop instead of falling back when the selected partial backend is unavailable or fails.
order_by	Order by signed "value" or absolute "magnitude".
safe_d_multiplier	Multiplier for extra partial components computed before selecting the requested components.
validate_inputs	Validate matrix and option inputs.

Value

A list with values and, unless only_values, vectors.

See Also

singular_decomp(), ase()

Examples

```
eig_decomp(diag(c(3, 2, 1)), d = 2, engine = "base")
```

---

embedding_estimating *Network embedding, clustering, estimation, and label alignment*

Description

Spectral and latent-space embedders, SBM/DCBM probability estimators, and deterministic or exact label-alignment helpers.

---

estimate_sbm *Estimate stochastic block models*

Description

Estimates SBM or DCBM parameters from dense, sparse, full, or supported NCV adjacency layouts.

Usage

```
estimate_sbm(
  A,
  g,
  K = max(g),
  fold_nodes = NULL,
  directed = FALSE,
  self_loops = FALSE,
  validate_inputs = TRUE,
  stabilizer = 0.01
)

estimate_dcbm(
  A,
  g,
  K = max(g),
  method = c("plugin", "spectral"),
  fold_nodes = NULL,
  row_norm = NULL,
  psi_omit = 0L,
  stabilizer = 0.01,
  spectral_engine = c("rspectra", "irlba", "base"),
  spectral_options = list(),
  validate_inputs = TRUE
)
```

Arguments

A	A finite full network matrix or supported rectangular NCV layout.
g	Positive integer community labels for the full network.
K	Positive number of communities.
fold_nodes	Optional original indices of held-out NCV nodes.
directed	Whether edges are directed. Used by `estimate_sbm()`.
self_loops	Whether diagonal edges are included. Used by `estimate_sbm()`.
validate_inputs	Whether to validate inputs; only audited internal callers should disable this.
stabilizer	Nonnegative density-scaled pseudocount used by the SBM and DCBM plug-in estimators.
method	DCBM estimator, either `"plugin"` or `"spectral"`.
row_norm	Optional supplied DCBM spectral row norms.
psi_omit	Number of trailing nodes omitted from returned DCBM degree parameters.
spectral_engine	Spectral-decomposition backend.

`spectral_options`
: Named list of spectral-decomposition options; see the `spectral_options` argument of `spectral_cluster()` for the accepted options and their decomposition backends.

Details

With `fold_nodes = NULL`, `A` is the full network. With fold nodes, `A` may be the full matrix, non-fold rows by all columns, all rows by fold columns, or non-fold rows by fold columns. The rectangular SBM estimator counts represented dyads, pools both available block orientations for undirected input, and identifies genuine self-pairs from original node indices.

`estimate_dcbm()` uses either the historical stabilized degree/block-sum plug-in estimator or spectral row norms. Full square networks use `eig_decomp()`; rectangular NCV layouts use `singular_decomp()`. `row_norm` may be a full-network vector, a vector for the columns of `A` when all row nodes occur there, or a named list with `row` and `col` components matching the dimensions of `A`.

Value

A fitted block-model object.

Functions

- `estimate_dcbm()`: Estimate degree-corrected block and node parameters.

See Also

`estimate_sbm_P_hat()`, `estimate_dcbm_P_hat()`, `spectral_cluster()`

Examples

```
A <- generate_sbm(n = 200, K = 3, alpha = 0.4, beta = 0.1,
                  seed = 11, ncores = 1)
g <- get_generator_parameters(A)$g_true
B_hat <- estimate_sbm(A, g, K = 3)
dim(B_hat)
```

---

estimate_sbm_P_hat *Reconstruct block-model probability matrices*

Description

Reconstructs fitted SBM or DCBM edge probabilities.

Usage

```
estimate_sbm_P_hat(
  A,
  g,
  K = max(g),
  fold_nodes = NULL,
  directed = FALSE,
  self_loops = FALSE,
  lower_clip = 0,
  upper_clip = 1
)

estimate_dcbm_P_hat(
  A,
  g,
  K = max(g),
  method = c("plugin", "spectral"),
  fold_nodes = NULL,
  row_norm = NULL,
  psi_omit = 0L,
  stabilizer = 0.01,
  spectral_engine = c("rspectra", "irlba", "base"),
  spectral_options = list(),
  self_loops = FALSE,
  lower_clip = 0,
  upper_clip = 1
)
```

Arguments

`A`	A finite full network matrix or supported rectangular NCV layout.
`g`	Positive integer community labels for the full network.
`K`	Positive number of communities.
`fold_nodes`	Optional original indices of held-out NCV nodes.
`directed`	Whether edges are directed. Used by `estimate_sbm_P_hat()`.
`self_loops`	Whether diagonal probabilities are retained.
`lower_clip, upper_clip`	Finite lower and upper bounds applied to fitted probabilities.
`method`	DCBM estimator, either `"plugin"` or `"spectral"`.
`row_norm`	Optional supplied DCBM spectral row norms.
`psi_omit`	Number of trailing nodes omitted from returned DCBM degree parameters.
`stabilizer`	Nonnegative plug-in denominator stabilizer.
`spectral_engine`	Spectral-decomposition backend.

spectral_options
: Named list passed to estimate_dcbm(); see that function and spectral_cluster() for the accepted spectral-decomposition options.

Details

In NCV mode, estimate_sbm_P_hat() returns the fold-by-fold probability matrix implied by the block estimate; otherwise it returns probabilities for all nodes.

estimate_dcbm_P_hat() computes $\hat{P}_{ij} = \hat{\psi}_i \hat{\psi}_j \hat{B}_{g_i,g_j}$. In NCV mode, degree parameters are estimated for fold nodes and the result is fold-by-fold. With psi_omit, the result covers retained trailing nodes.

Value

A fitted edge-probability matrix.

Functions

- estimate_dcbm_P_hat(): Reconstruct degree-corrected block-model probabilities.

See Also

estimate_sbm(), estimate_dcbm(), clip_probabilities()

Examples

```
A <- generate_sbm(n = 200, K = 3, alpha = 0.4, beta = 0.1,
                  seed = 12, ncores = 1)
g <- get_generator_parameters(A)$g_true
P_hat <- estimate_sbm_P_hat(A, g, K = 3)
dim(P_hat)
```

---

extrema_selectors *Compute hard and soft extrema*

Description

softmax() and softmin() return normalized exponential weights. hardmax() and hardmin() return zero-one selectors marking all ties. The which_soft*() functions sample one index from the soft weights; the which_hard*() functions return every tied hard-extremum index.

Usage

```
softmax(x, temperature = 1, na_rm = FALSE)

softmin(x, temperature = 1, na_rm = FALSE)

hardmax(x, na_rm = FALSE)
```

```
hardmin(x, na_rm = FALSE)

which_softmax(x, temperature = 1, na_rm = FALSE)

which_softmin(x, temperature = 1, na_rm = FALSE)

which_hardmax(x, na_rm = FALSE)

which_hardmin(x, na_rm = FALSE)
```

Arguments

x	Numeric vector whose extrema are selected.
temperature	Positive soft-selection temperature.
na_rm	Remove missing values before selection.

Value

A numeric weight vector or selected index vector, according to the function called.

See Also

softplus(), sigmoid()

Examples

```
softmax(c(1, 2, 3))
hardmin(c(2, 1, 1))
which_hardmax(c(1, 3, 3))
```

---

generate_adjacency *Sample a network from edge probabilities*

Description

Samples a dense or sparse adjacency matrix from P. Explicit row-specific seeds make results reproducible across worker counts.

Usage

```
generate_adjacency(
  P,
  representation = c("sparse", "dense"),
  directed = FALSE,
  self_loops = FALSE,
  seed = NULL,
```

```
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
  validate_inputs = TRUE,
  n = NULL
)
```

Arguments

P	Dense or sparse edge-probability matrix.
representation	Whether to return a sparse or dense adjacency matrix.
directed	Whether to sample directed edges independently.
self_loops	Whether diagonal edges may be sampled.
seed	Optional nonnegative reproducibility seed.
ncores	Positive worker count.
validate_inputs	Whether to validate P before sampling.
n	Optional network size inferred from P when omitted.

Value

A base matrix or sparse dgCMatrix, according to representation.

See Also

generate_er(), generate_rdpg(), generate_sbm()

Examples

```
P <- matrix(0.03, 200, 200)
diag(P) <- 0
A <- generate_adjacency(P, seed = 3, ncores = 1)
dim(A)
```

---

generate_community_labels

Generate or validate community labels

Description

Produces community memberships for an SBM, DCBM, or latent-space model.

Usage

```
generate_community_labels(
  n = NULL,
  K = NULL,
  g_true = NULL,
  community_probabilities = NULL,
  seed = NULL
)
```

Arguments

n	Positive number of nodes.
K	Positive number of communities.
g_true	Optional supplied community-label vector.
community_probabilities	Optional length-K community probabilities.
seed	Optional nonnegative reproducibility seed.

Value

An integer vector of length n with values from 1 through K.

See Also

generate_sbm(), generate_dcbm(), generate_lsm_positions()

Examples

```
g <- generate_community_labels(n = 200, K = 3, seed = 5)
table(g)
```

---

generate_dcbm *Generate block-model networks*

Description

generate_dcbm() generates a degree-corrected stochastic block model; generate_sbm() is its ordinary-block-model twin with unit degree effects. Both return dense or sparse adjacency matrices.

generate_sbm() fixes every degree parameter to one.

Usage

```
generate_dcbm(
  n = NULL,
  K = NULL,
  g_true = NULL,
  community_probabilities = NULL,
  P_block = NULL,
  alpha = 0.2,
  beta = 0.2,
  psi = NULL,
  degree_distribution = generate_inverse_beta_degree_parameters,
  degree_args = NULL,
  degree_scale = c("none", "max_by_community", "mean_one_by_community"),
  sparsity_multiplier = 1,
  average_degree = NULL,
```

```
  average_degree_method = c("naive", "calibrated"),
  lower_clip = 0,
  upper_clip = 1,
  representation = c("sparse", "dense"),
  directed = FALSE,
  self_loops = FALSE,
  seed = NULL,
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE)
)

generate_sbm(
  n = NULL,
  K = NULL,
  g_true = NULL,
  community_probabilities = NULL,
  P_block = NULL,
  alpha = 0.2,
  beta = 0.2,
  sparsity_multiplier = 1,
  average_degree = NULL,
  average_degree_method = c("naive", "calibrated"),
  lower_clip = 0,
  upper_clip = 1,
  representation = c("sparse", "dense"),
  directed = FALSE,
  self_loops = FALSE,
  seed = NULL,
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE)
)
```

Arguments

`n`	Positive number of nodes.
`K`	Positive number of communities.
`g_true`	Optional supplied community-label vector.
`community_probabilities`	Optional length-`K` community probabilities.
`P_block`	Optional `K`-by-`K` block-probability matrix.
`alpha`, `beta`	Within- and between-community probabilities used when `P_block` is omitted.
`psi`	Optional supplied DCBM degree parameters.
`degree_distribution`	Function used to generate DCBM degree parameters.
`degree_args`	Optional named list passed to `degree_distribution`.
`degree_scale`	Community-wise degree-normalization method.
`sparsity_multiplier`	Nonnegative multiplier applied to probabilities.

average_degree	Optional target expected average degree.
average_degree_method	Scaling method, either "naive" or "calibrated".
lower_clip, upper_clip	Finite probability clipping bounds.
representation	Whether to return a sparse or dense adjacency matrix.
directed	Whether to generate a directed network.
self_loops	Whether diagonal edges may be generated.
seed	Optional nonnegative reproducibility seed.
ncores	Positive worker count.

Value

A generated adjacency matrix; see get_generator_parameters().

See Also

generate_community_labels(), estimate_sbm(), estimate_dcbm()

Examples

```
A <- generate_sbm(n = 200, K = 3, alpha = 0.4, beta = 0.1,
                  seed = 3, ncores = 1)
get_generator_parameters(A)
```

---

generate_degree_parameters

Generate or validate DCBM degree parameters

Description

Returns nonnegative node degree parameters, optionally normalized within communities.

Usage

```
generate_degree_parameters(
  n = NULL,
  psi = NULL,
  degree_distribution = generate_inverse_beta_degree_parameters,
  degree_args = NULL,
  degree_scale = c("none", "max_by_community", "mean_one_by_community"),
  g_true = NULL,
  seed = NULL
)
```

Arguments

n	Positive number of nodes.
psi	Optional supplied nonnegative degree parameters.
degree_distribution	Function used to generate degree parameters.
degree_args	Optional named list passed to degree_distribution.
degree_scale	Community-wise degree-normalization method.
g_true	Optional community-label vector.
seed	Optional nonnegative reproducibility seed.

Value

A nonnegative numeric vector of length n.

See Also

generate_inverse_beta_degree_parameters(), generate_dcbm()

Examples

```
g <- generate_community_labels(n = 200, K = 3, seed = 6)
psi <- generate_degree_parameters(n = 200, g_true = g, seed = 7)
length(psi)
```

---

generate_er *Generate an Erdos-Renyi network*

Description

Generates a dense or sparse Erdos-Renyi adjacency matrix and stores its generating parameters as lightweight metadata.

Usage

```
generate_er(
  n,
  p = NULL,
  average_degree = NULL,
  representation = c("sparse", "dense"),
  directed = FALSE,
  self_loops = FALSE,
  seed = NULL,
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE)
)
```

Arguments

n	Positive number of nodes.
p	Optional edge probability.
average_degree	Optional target expected average degree.
representation	Whether to return a sparse or dense adjacency matrix.
directed	Whether to generate a directed network.
self_loops	Whether diagonal edges may be generated.
seed	Optional nonnegative reproducibility seed.
ncores	Positive worker count.

Value

A generated adjacency matrix; see `get_generator_parameters()`.

See Also

`generate_sbm()`, `generate_rdpg()`

Examples

```
A <- generate_er(n = 200, average_degree = 5, seed = 1, ncores = 1)
get_generator_parameters(A)
```

---

generate_inverse_beta_degree_parameters

Generate inverse-beta degree parameters

Description

Draws the historical default degree parameters used by the DCBM generator.

Usage

```
generate_inverse_beta_degree_parameters(n, shape_1 = 4, shape_2 = 1)
```

Arguments

n	Nonnegative number of degree parameters.
shape_1, shape_2	Positive beta-distribution shape parameters.

Value

A nonnegative numeric vector of length n.

See Also

generate_degree_parameters(), generate_dcbm()

Examples

```
psi <- generate_inverse_beta_degree_parameters(200)
length(psi)
```

`generate_latent_positions` *Generate or validate latent positions*

Description

Returns a finite n-by-d latent-position matrix, either supplied by the caller or sampled from `latent_distribution`.

Usage

```
generate_latent_positions(
  n = NULL,
  d = NULL,
  Z = NULL,
  latent_distribution = stats::runif,
  latent_args = NULL,
  seed = NULL
)
```

Arguments

n	Positive number of nodes.
d	Positive latent dimension.
Z	Optional supplied latent-position matrix.
latent_distribution	Function used to sample latent coordinates.
latent_args	Optional named list passed to `latent_distribution`.
seed	Optional nonnegative reproducibility seed.

Value

A numeric matrix with n rows and d columns.

See Also

generate_rdpg(), generate_lsm_positions()

Examples

```
Z <- generate_latent_positions(n = 200, d = 3, seed = 4)
dim(Z)
```

`generate_lsm` *Generate a latent-space-model network*

Description

Generates a dense or sparse Hoff/Ma-style latent-space-model adjacency matrix from supplied or generated positions, memberships, and intercepts.

Usage

```
generate_lsm(
  n = NULL,
  d = NULL,
  K = 1L,
  g_true = NULL,
  community_probabilities = NULL,
  alpha = NULL,
  Z = NULL,
  Z_left = NULL,
  Z_right = NULL,
  normalize_Z = TRUE,
  distance_adjustment = TRUE,
  mean_lower = -1,
  mean_upper = 1,
  noise_lower = -2,
  noise_upper = 2,
  average_degree = NULL,
  average_degree_method = c("naive", "calibrated"),
  naive_iterations = 10L,
  lower_clip = 0,
  upper_clip = 1,
  representation = c("sparse", "dense"),
  directed = FALSE,
  self_loops = FALSE,
  seed = NULL,
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE)
)
```

Arguments

`n`	Positive number of nodes.
`d`	Positive latent dimension.

K	Positive number of communities.
g_true	Optional supplied community-label vector.
community_probabilities	Optional length-K community probabilities.
alpha	Optional scalar or node-specific intercepts.
Z	Optional undirected latent-position matrix.
Z_left, Z_right	Optional left and right directed latent-position matrices.
normalize_Z	Whether to center and scale latent positions.
distance_adjustment	Whether to apply the historical distance scaling.
mean_lower, mean_upper	Finite bounds for generated community means.
noise_lower, noise_upper	Finite truncation bounds for generated latent noise.
average_degree	Optional target expected average degree.
average_degree_method	Scaling method, either "naive" or "calibrated".
naive_iterations	Positive iteration count for historical scaling.
lower_clip, upper_clip	Finite probability clipping bounds.
representation	Whether to return a sparse or dense adjacency matrix.
directed	Whether to generate a directed network.
self_loops	Whether diagonal edges may be generated.
seed	Optional nonnegative reproducibility seed.
ncores	Positive worker count.

Value

A generated adjacency matrix; see get_generator_parameters().

See Also

generate_lsm_positions(), generate_lsm_alpha(), lsm_pgd(), netcrop_lsm()

Examples

```
A <- generate_lsm(n = 200, d = 3, K = 3, seed = 7, ncores = 1)
get_generator_parameters(A)
```

generate_lsm_alpha *Generate latent-space-model intercepts*

Description

Validates or generates the node intercept used by the latent-space model.

Usage

```
generate_lsm_alpha(n, alpha = NULL, seed = NULL)
```

Arguments

n	Positive number of nodes.
alpha	Optional scalar or length-n intercept vector.
seed	Optional nonnegative reproducibility seed.

Value

A finite numeric vector of length n.

See Also

generate_lsm(), generate_lsm_positions()

Examples

```
alpha <- generate_lsm_alpha(n = 200, seed = 6)
length(alpha)
```

generate_lsm_positions *Generate latent-space-model positions*

Description

Generates an n-by-d latent-position matrix from a finite Gaussian mixture with K communities.

Usage

```
generate_lsm_positions(
  n,
  d,
  K,
  g_true,
  mean_lower = -1,
  mean_upper = 1,
  noise_lower = -2,
  noise_upper = 2,
  seed = NULL
)
```

Arguments

n	Positive number of nodes.
d	Positive latent dimension.
K	Positive number of communities.
g_true	Community-label vector of length n.
mean_lower, mean_upper	Finite bounds for community means.
noise_lower, noise_upper	Finite truncation bounds for latent noise.
seed	Optional nonnegative reproducibility seed.

Value

A numeric matrix with n rows and d columns.

See Also

generate_lsm(), generate_lsm_alpha(), normalize_lsm_positions()

Examples

```
g <- generate_community_labels(n = 200, K = 3, seed = 4)
Z <- generate_lsm_positions(n = 200, d = 3, K = 3, g_true = g, seed = 5)
dim(Z)
```

`generate_rdpg` *Generate a random dot product graph*

Description

Generates a dense or sparse RDPG adjacency matrix. Undirected models use `Z`; directed models use `Z_left` and `Z_right`.

Usage

```
generate_rdpg(
  n = NULL,
  d = NULL,
  Z = NULL,
  Z_left = NULL,
  Z_right = NULL,
  latent_distribution = stats::runif,
  latent_args = NULL,
  sparsity_multiplier = 1,
  scale_P = TRUE,
  average_degree = NULL,
  average_degree_method = c("naive", "calibrated"),
  lower_clip = 0,
  upper_clip = 1,
  representation = c("sparse", "dense"),
  directed = FALSE,
  self_loops = FALSE,
  seed = NULL,
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE)
)
```

Arguments

`n`	Positive number of nodes.
`d`	Positive latent dimension.
`Z`	Optional undirected latent-position matrix.
`Z_left`, `Z_right`	Optional left and right directed latent-position matrices.
`latent_distribution`	Function used to sample latent coordinates.
`latent_args`	Optional named list passed to `latent_distribution`.
`sparsity_multiplier`	Nonnegative multiplier applied to dot-product probabilities.
`scale_P`	Whether to calibrate dot products before sampling.
`average_degree`	Optional target expected average degree.

`average_degree_method`
: Scaling method, either `"naive"` or `"calibrated"`.

`lower_clip, upper_clip`
: Finite probability clipping bounds.

`representation`
: Whether to return a sparse or dense adjacency matrix.

`directed`
: Whether to generate a directed network.

`self_loops`
: Whether diagonal edges may be generated.

`seed`
: Optional nonnegative reproducibility seed.

`ncores`
: Positive worker count.

Value

A generated adjacency matrix; see `get_generator_parameters()`.

See Also

`generate_latent_positions()`, `ase()`, `netcrop_rdpg()`

Examples

```
A <- generate_rdpg(n = 200, d = 3, seed = 2, ncores = 1)
get_generator_parameters(A)
```

---

generate_truncated_normal

Generate truncated standard-normal values

Description

Draws standard-normal noise on a finite interval by inverse-CDF sampling.

Usage

```
generate_truncated_normal(n, lower_bound = -2, upper_bound = 2)
```

Arguments

`n`
: Positive number of values to generate.

`lower_bound, upper_bound`
: Finite truncation bounds with `lower_bound` < `upper_bound`.

Value

A numeric vector of length `n`.

See Also

generate_lsm_positions()

Examples

```
x <- generate_truncated_normal(200, lower_bound = -2, upper_bound = 2)
range(x)
```

---

get_generator_parameters

Retrieve parameters attached by a network generator

Description

Returns the compact generating parameters stored on a network produced by a netOP generator. Both dense and sparse generated matrices are supported.

Usage

```
get_generator_parameters(A)
```

Arguments

A	A network returned by a netOP generator.

Value

A named list of generating parameters, or NULL when none is attached.

See Also

generate_er(), generate_sbm(), generate_rdpg(), generate_lsm()

Examples

```
A <- generate_er(n = 200, average_degree = 5, seed = 1, ncores = 1)
get_generator_parameters(A)
```

graph_laplacian *Construct a graph Laplacian*

Description

Constructs an unnormalized or symmetric normalized graph Laplacian. For positive `tau`, the adjacency is first regularized by adding `tau * mean(degree) / n` to every entry; `tau = 0` gives the usual matrix.

Usage

```
graph_laplacian(A, normalized = TRUE, tau = 0)
```

Arguments

`A`	A finite symmetric square dense or sparse adjacency matrix.
`normalized`	Construct the symmetric normalized Laplacian.
`tau`	Nonnegative dimensionless regularization strength.

Value

A dense or sparse Laplacian matrix.

See Also

`spectral_cluster()`, `ase()`

Examples

```
A <- matrix(c(0, 1, 1, 0), 2, 2)
graph_laplacian(A, normalized = FALSE)
```

is_symmetric_network_matrix *Check whether a network matrix is symmetric*

Description

Tests symmetry for either a base matrix or a sparse `Matrix` object without requiring users to attach Matrix.

Usage

```
is_symmetric_network_matrix(A, tolerance = 1e-10)
```

Arguments

A	A finite dense or sparse network matrix.
tolerance	Nonnegative numerical symmetry tolerance.

Value

A single logical value.

See Also

is_symmetric_matrix()

---

label_match_greedy *Match community labels*

Description

Aligns labels using greedy assignment or exact brute-force matching.

Usage

```
label_match_greedy(
  match_this,
  standard,
  K = max(c(match_this, standard)),
  algorithm = c("greedy", "hungarian"),
  return_mapping = FALSE
)

label_match_brute_force(
  match_this,
  standard,
  K = max(c(match_this, standard)),
  return_mapping = FALSE,
  confirm_large = NULL
)
```

Arguments

match_this	Positive integer labels to relabel.
standard	Positive integer target labels of the same length.
K	Positive number of label classes.
algorithm	Assignment method for label_match_greedy(): historical greedy matching or globally optimal Hungarian matching.
return_mapping	Whether to return mapping diagnostics with the labels.
confirm_large	For exact matching with more than eight classes, explicitly confirm factorial computation in noninteractive use.

Details

The greedy function uses exact shortcuts for identical and two-community inputs, followed by historical maximum-overlap assignment. Its `"hungarian"` option uses a dependency-free cubic-time assignment and maximizes total agreement globally.

`label_match_brute_force()` generates permutations recursively and scores them one at a time. Runtime remains factorial. For `K > 8`, an interactive call requires confirmation and noninteractive callers must set `confirm_large = TRUE`.

Value

A relabeled vector, optionally with mapping diagnostics.

Functions

- `label_match_brute_force()`: Match labels by enumerating every permutation.

See Also

`spectral_cluster()`, `sonnet()`

Examples

```
labels <- c(2, 2, 3, 3, 1, 1)
standard <- c(1, 1, 2, 2, 3, 3)
label_match_greedy(labels, standard, K = 3)$matched_labels
```

---

lsm_pgd *Fit a latent-space model by projected gradient descent*

Description

Estimates latent positions and intercepts for a dense or sparse network.

Usage

```
lsm_pgd(
  A,
  d,
  step_size = 0.3,
  niter = 100L,
  trace = FALSE,
  Z_init = NULL,
  alpha_init = NULL,
  epsilon = 1e-06,
  use_cpp = TRUE,
  ram_check = FALSE
)
```

Arguments

A	A finite square dense or sparse adjacency matrix.
d	Positive latent dimension.
step_size	Positive projected-gradient step size.
niter	Positive number of projected-gradient iterations.
trace	Whether to print optimizer progress.
Z_init	Optional nrow(A)-by-d initial latent-position matrix.
alpha_init	Optional scalar or length-nrow(A) initial intercept.
epsilon	Positive probability-clipping constant.
use_cpp	Whether to use the compiled optimizer when available.
ram_check	Whether to report a conservative RAM preflight estimate.

Details

Initialization uses USVT, a stable logit transform, the solution of $(nI + 11^T)\alpha = \mathrm{rowSums}(\theta)$, and double-centering without constructing a dense centering matrix. The diagonal is excluded from the objective and gradient. Direct initial values may replace either spectral initializer.

Value

A fitted latent-space-model object.

See Also

generate_lsm(), netcrop_lsm()

Examples

```
A <- generate_lsm(n = 200, d = 3, K = 3, seed = 10, ncores = 1)
fit <- lsm_pgd(A, d = 3, niter = 2, use_cpp = FALSE)
dim(fit$Z_hat)
```

---

matrix-generics *Matrix summary generics*

Description

The functions mean(), diag(), rowMeans(), rowSums(), colMeans(), and colSums() are re-exported from Matrix. The sum() wrapper delegates to base::sum(), which dispatches to Matrix methods for sparse inputs. These functions support common summaries of sparse network matrices.

Usage

```
sum(..., na.rm = FALSE)
```

Arguments

`...`	Objects passed to the selected Matrix-aware generic.
`na.rm`	Whether missing values should be removed by `sum()`.

Value

`sum()` and `mean()` return a numeric scalar. `diag()` returns the diagonal vector of a matrix. `rowMeans()` and `rowSums()` return one value per row; `colMeans()` and `colSums()` return one per column. See the corresponding Matrix help for other supported inputs.

Examples

```
A <- Matrix::Matrix(matrix(c(0, 1, 1, 0), nrow = 2), sparse = TRUE)
sum(A)
mean(A)
diag(A)
rowSums(A)
colMeans(A)
```

`matrix_density` *Compute matrix density*

Description

Returns the fraction of matrix entries that are nonzero. This is the entry density called `"sparsity"` by the decomposition APIs.

Usage

```
matrix_density(A)
```

Arguments

`A`	A finite, nonempty dense or sparse matrix.

Value

A numeric scalar between zero and one.

See Also

`generate_adjacency()`

Examples

```
matrix_density(diag(3))
```

measure_peak_ram *Measure peak memory use*

Description

Runs an expression or function through `peakRAM::peakRAM()` while retaining both the evaluated result and the measurement table. The process is evaluated exactly once.

Usage

```
measure_peak_ram(process, ...)
```

Arguments

`process`	An unevaluated R expression or a function to execute and measure.
`...`	Additional arguments forwarded when `process` evaluates to a function.

Value

A list containing `result`, the process value, and `metrics`, a one-row data frame with elapsed seconds, total RAM used in MiB, and peak RAM used in MiB.

See Also

`available_ram()`, `report_ram_preflight()`

Examples

```
if (requireNamespace("peakRAM", quietly = TRUE)) {
  measure_peak_ram(sum, 1:10)
}
```

modal *Compute the statistical mode*

Description

Returns the most frequent value in `x`, using the first value to break ties. If `na_rm = FALSE`, `NA` is a possible modal value; if removal leaves no observations, a typed missing value is returned.

Usage

```
modal(x, na_rm = FALSE)
```

Arguments

x	Input vector.
na_rm	Remove missing values before finding the mode.

Value

A scalar with the same basic type as x.

See Also

nmi()

Examples

```
modal(c(1, 2, 2, 3))
```

---

mult_reg_sonnet *Fit SONNET over multiple regularizers*

Description

Fits SONNET across a requested regularization grid for one network or a matched list.

Usage

```
mult_reg_sonnet(
  A,
  K,
  tau_candidates,
  num_subnetworks = NULL,
  overlap_size = NULL,
  extra_nrep = 0L,
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
  seed = NULL,
  matching_method = c("greedy", "hungarian", "brute_force"),
  confirm_large = TRUE,
  verbose = TRUE,
  force_windows = FALSE,
  ram_check = FALSE,
  share_overlap = FALSE,
  parameter_select_options = list(),
  failure_handling = c("stop", "omit"),
  retain_fits = TRUE,
  ...
)
```

Arguments

A	One adjacency matrix or a nonempty list of equal-sized matrices.
K	Fixed positive number of communities.
tau_candidates	Unique finite nonnegative regularization candidates.
num_subnetworks	Number of overlapping subnetworks; NULL selects it automatically.
overlap_size	Number of overlap nodes; NULL selects it automatically.
extra_nrep	Number of additional SONNET repetitions.
ncores	Positive SONNET worker count.
seed	Optional nonnegative reproducibility seed.
matching_method	Label-alignment method.
confirm_large	Confirm potentially factorial brute-force matching.
verbose	Print progress messages.
force_windows	Use the Windows-compatible parallel backend.
ram_check	Report conservative RAM demand.
share_overlap	Reuse one overlap across repetitions.
parameter_select_options	Named list of automatic-partition options passed to sonnet_param_select() through sonnet().
failure_handling	Stop or omit failed candidate fits.
retain_fits	Retain fitted sonnet objects.
...	Named spectral-clustering arguments forwarded to sonnet().

Details

Candidate fits run sequentially so ncores remains available to SONNET's internal subnetwork computations. The same realized seed is reused across regularization candidates within each network.

Value

A mult_reg_clustering object containing SONNET labels and optional fits by network and regularizer, parameters, tasks, failures, seeds, and timing.

See Also

sonnet(), mult_reg_spectral_cluster()

Examples

```
A <- generate_sbm(n = 200, K = 3, alpha = 0.5, beta = 0.1,
                  seed = 39, ncores = 1)
mult_reg_sonnet(
  A, K = 3, tau_candidates = c(0, 0.1),
  num_subnetworks = 2, overlap_size = 20, ncores = 1,
  seed = 40, verbose = FALSE, spectral_engine = "base"
)
```

---

mult_reg_spectral_cluster

Fit spectral clusterings over multiple regularizers

Description

Fits spectral clustering across a requested regularization grid for one network or a matched list of networks.

Usage

```
mult_reg_spectral_cluster(
  A,
  K,
  tau_candidates,
  laplacian = FALSE,
  normalize_laplacian = TRUE,
  handle_zero_degree_nodes = c("none", "random_label", "remove"),
  row_normalize = FALSE,
  spectral_method = c("eigen", "svd"),
  spectral_engine = c("RSpectra", "irlba", "base"),
  spectral_options = list(),
  cluster_engine = c("clara", "kmeans", "pam"),
  cluster_options = list(),
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
  seed = NULL,
  verbose = TRUE,
  force_windows = FALSE,
  ram_check = FALSE,
  failure_handling = c("stop", "omit"),
  retain_fits = TRUE
)
```

Arguments

A	One adjacency matrix or a nonempty list of equal-sized matrices.
K	Fixed positive number of communities.

`tau_candidates` Unique finite nonnegative regularization candidates.

`laplacian` Convert adjacency matrices to graph Laplacians.

`normalize_laplacian`
Use symmetric normalized Laplacians.

`handle_zero_degree_nodes`
Zero-degree node policy.

`row_normalize` Normalize embedding rows before clustering.

`spectral_method`
Use an eigen- or singular-vector representation.

`spectral_engine`
Decomposition backend.

`spectral_options`
Named list of decomposition options passed to the `spectral_options` argument of `spectral_cluster()`.

`cluster_engine` Clustering backend.

`cluster_options`
Named list of clustering options passed to the `cluster_options` argument of `spectral_cluster()`.

`ncores` Positive task-worker count.

`seed` Optional nonnegative reproducibility seed.

`verbose` Print progress messages.

`force_windows` Use the Windows-compatible parallel backend.

`ram_check` Report conservative RAM demand.

`failure_handling`
Stop or omit failed candidate fits.

`retain_fits` Retain fitted `spectral_cluster` objects.

Details

Network-by-candidate tasks are parallelized. The same realized seed is used across candidates within a network so stochastic clustering comparisons are fair. Results can retain complete fitted objects or labels only.

Value

A `mult_reg_clustering` object containing labels and optional fits by network and candidate, resolved parameters, task metadata, failures, seeds, workers, timing, and RAM diagnostics.

See Also

`dkest_tune_regularizer()`, `netcrop_tune_regularizer()`

Examples

```
A <- generate_sbm(n = 200, K = 3, alpha = 0.5, beta = 0.1,
                  seed = 37, ncores = 1)
fits <- mult_reg_spectral_cluster(
  A, K = 3, tau_candidates = c(0, 0.1), ncores = 1,
  seed = 38, verbose = FALSE, spectral_engine = "base",
  cluster_engine = "kmeans"
)
```

---

ncv_stability *Node cross-validation stability selection*

Description

Repeated node cross-validation for selecting the number of block-model communities. This is the netOP authors' implementation of Chen and Lei (2018), with numerical-stability checks and explicit failure handling.

References

Chen, K. and Lei, J. (2018). Network Cross-Validation for Determining the Number of Communities in Network Data. *Journal of the American Statistical Association*, 113(521), 241-251. doi:10.1080/01621459.2016.1246365

---

ncv_stability_blockmodel

Select block-model size with node cross-validation

Description

Repeats node cross-validation over every community count from one through `max_K` for both SBM and DCBM candidates.

Usage

```
ncv_stability_blockmodel(
  A,
  max_K,
  cv = 3L,
  nrep = 20L,
  dc_est = c("spectral", "plugin"),
  tau = 0,
  use_laplacian = FALSE,
  losses = "sse",
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
```

```
  seed = NULL,
  verbose = TRUE,
  force_windows = FALSE,
  ram_check = FALSE,
  failure_handling = c("stop", "omit"),
  retain_intermediates = c("all", "minimal")
)
```

Arguments

A	Binary, symmetric, loop-free adjacency matrix.
max_K	Maximum community count; candidates are 1:max_K.
cv, nrep	Positive numbers of folds and repetitions.
dc_est	DCBM estimator, either "spectral" or "plugin".
tau	Nonnegative spectral regularization value.
use_laplacian	Use Laplacian spectral representations.
losses	Canonical loss names to evaluate.
ncores	Positive outer worker count.
seed	Optional nonnegative reproducibility seed.
verbose	Print progress messages.
force_windows	Use the Windows-compatible parallel backend.
ram_check	Report conservative RAM demand.
failure_handling	Stop or omit failed repetitions.
retain_intermediates	Retain all or minimal repetition output.

Details

Each repetition partitions nodes into folds, fits candidates on rectangular training blocks, and evaluates held-out dyads. Repetitions are parallelized, fold stages remain sequential, and probabilities are clipped consistently for logarithmic losses. Failed repetitions can stop or be omitted.

Value

A netcrop_blockmodel object with algorithm = "NCV", tidy fold and repetition losses, selections, fold metadata, realized seeds, failures, worker counts, timing, and optional raw output.

References

Chen, K. and Lei, J. (2018). Network Cross-Validation for Determining the Number of Communities in Network Data. *Journal of the American Statistical Association*, 113(521), 241-251. doi:10.1080/01621459.2016.1246365

See Also

ecv_stability_blockmodel(), netcrop_blockmodel()

Examples

```
A <- generate_sbm(n = 200, K = 3, alpha = 0.55, beta = 0.08,
                  seed = 24, ncores = 1)
ncv_stability_blockmodel(A, max_K = 5, cv = 2, nrep = 1,
                         ncores = 1, seed = 26, verbose = FALSE)
```

---

netcrop_blockmodel *Select a block model with NETCROP*

Description

Selects the number of communities and either the stochastic block model (SBM), degree-corrected block model (DCBM), or both by overlapping-subnetwork cross-validation. The procedure obtains spectral labels on each subnetwork, estimates block-model parameters, aligns labels through the common overlap, and evaluates predictions between every pair of non-overlap pieces.

Usage

```
netcrop_blockmodel(
  A,
  K_candidates,
  num_subnetworks = NULL,
  overlap_size = NULL,
  nrep = 1L,
  losses = "sse",
  model_candidates = c("SBM", "DCBM"),
  sbm_est_options = list(),
  dcbm_est_options = list(),
  matching_method = c("greedy", "hungarian", "brute_force"),
  confirm_large = NULL,
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
  seed = NULL,
  verbose = TRUE,
  force_windows = FALSE,
  ram_check = FALSE,
  parameter_select_options = list(),
  retain_intermediates = c("all", "minimal"),
  laplacian = FALSE,
  regularize_tau = 0
)

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

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

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

## S3 method for class 'netcrop_blockmodel'
plot(x, aggregate = TRUE, ...)
```

Arguments

A	Finite square symmetric binary adjacency matrix with zero diagonal, including a supported sparse Matrix object.
K_candidates	Non-empty vector of positive integer community counts; duplicates are discarded. Conventionally this is 1:5.
num_subnetworks	Optional integer number of subnetworks, at least 2. If it or overlap_size is NULL, missing values are selected with netcrop_param_select().
overlap_size	Optional positive integer overlap size. It must leave enough non-overlap nodes for the requested subnetworks and candidates.
nrep	Positive integer number of independent NETCROP repetitions.
losses	Character vector naming prediction-loss functions available in the calling environment, such as "sse".
model_candidates	Character vector selecting "SBM", "DCBM", or both candidate model families.
sbm_est_options, dcbm_est_options	Named lists with optional spectral_cluster and estimate components, each itself a named list: list(spectral_cluster = list(...), estimate = list(...)). The spectral_cluster component is passed to spectral_cluster(). The estimate component is passed to estimate_sbm() for sbm_est_options and to estimate_dcbm() for dcbm_est_options.
matching_method	Label-alignment method: "greedy", "hungarian", or "brute_force".
confirm_large	Optional logical passed to label matching to confirm an unusually large brute-force matching problem.
ncores	Positive integer worker count used across decomposition, estimation, alignment, and loss tasks.
seed	Optional nonnegative integer-like reproducibility seed.
verbose	Whether to print progress messages.
force_windows	Whether to force the Windows-compatible parallel backend, including for backend testing on other platforms.
ram_check	Whether to run RAM preflight checks before major operations.
parameter_select_options	Named list of additional options passed to netcrop_param_select() when partition parameters are selected automatically.

retain_intermediates
: Either "all" or "minimal", controlling how many fitted models and intermediate arrays are retained.

laplacian
: Whether to convert subnetwork adjacency matrices to graph Laplacians before spectral clustering.

regularize_tau
: Nonnegative Laplacian regularization strength shared by SBM and DCBM candidates.

x
: A netcrop_blockmodel or summary.netcrop_blockmodel object. For plotting, it is the first result to display or compare.

...
: Additional plotting arguments or further netcrop_blockmodel objects for comparison; ignored by print and summary methods.

object
: A fitted netcrop_blockmodel object to summarize.

aggregate
: Whether a plot should aggregate loss curves across repetitions. A netcrop_blockmodel object in this position instead starts a comparison plot.

Value

netcrop_blockmodel() returns an object of class netcrop_blockmodel containing candidate-wise cross-validation losses, repetition and overall selections, split and alignment diagnostics, retained intermediates, resource information, and timing. summary.netcrop_blockmodel() returns a compact summary object; print methods return their input invisibly, and the plot method returns a ggplot object.

See Also

ecv_stability_blockmodel(), ncv_stability_blockmodel()

Examples

```
A <- generate_sbm(n = 200, K = 3, alpha = 0.5, beta = 0.1,
                  seed = 5, ncores = 1)
fit <- netcrop_blockmodel(
  A, K_candidates = 1:5, num_subnetworks = 2, overlap_size = 30,
  model_candidates = "SBM", ncores = 1, seed = 6, verbose = FALSE
)
fit
summary(fit)
```

---

netcrop_lsm *Select latent-space dimension with NETCROP*

Description

Selects a latent-space-model (LSM) dimension by overlapping-subnetwork cross-validation. The procedure fits every candidate on every subnetwork, exactly reparameterizes each fitted inner-product model as a squared-distance model, rigidly aligns fits through the overlap, and evaluates every unordered pair of non-overlap pieces.

Usage

```
netcrop_lsm(
  A,
  d_candidates,
  num_subnetworks = NULL,
  overlap_size = NULL,
  nrep = 1L,
  losses = "sse",
  lsm_options = list(),
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
  seed = NULL,
  verbose = TRUE,
  force_windows = FALSE,
  ram_check = FALSE,
  parameter_select_options = list(),
  retain_intermediates = c("all", "minimal")
)

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

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

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

## S3 method for class 'netcrop_lsm'
plot(x, aggregate = TRUE, ...)
```

Arguments

A	Finite square symmetric binary adjacency matrix with zero diagonal, including a supported sparse Matrix object.
d_candidates	Non-empty vector of positive integer latent dimensions; duplicates are discarded. Dimensions must be smaller than the effective subgraph size; conventionally the candidate vector is 1:5.
num_subnetworks	Optional integer number of subnetworks, at least 2. If it or overlap_size is NULL, missing values are selected with netcrop_param_select().
overlap_size	Optional positive integer overlap size. It must leave enough non-overlap nodes for the requested subnetworks and dimensions.
nrep	Positive integer number of independent NETCROP repetitions.
losses	Character vector naming prediction-loss functions available in the calling environment, such as "sse".
lsm_options	Named list of additional options passed to lsm_pgd().

ncores	Positive integer worker count used across fitting, alignment, and loss tasks.
seed	Optional nonnegative integer-like reproducibility seed.
verbose	Whether to print progress messages.
force_windows	Whether to force the Windows-compatible parallel backend, including for backend testing on other platforms.
ram_check	Whether to run RAM preflight checks before major operations.
parameter_select_options	Named list of additional options passed to netcrop_param_select() when partition parameters are selected automatically.
retain_intermediates	Either "all" or "minimal", controlling how many fitted models and intermediate arrays are retained.
x	A netcrop_lsm or summary.netcrop_lsm object to print or plot.
...	Additional arguments, currently ignored by the print, summary, and plot methods.
object	A fitted netcrop_lsm object to summarize.
aggregate	Whether a plot should aggregate loss curves across repetitions.

Value

netcrop_lsm() returns an object of class netcrop_lsm containing candidate-wise cross-validation losses, repetition and overall dimension selections, split diagnostics, fitted LSM options, retained intermediates, resource information, and timing. summary.netcrop_lsm() returns a compact summary object; print methods return their input invisibly, and the plot method returns a ggplot object.

See Also

generate_lsm(), lsm_pgd()

Examples

```
A <- generate_lsm(n = 200, d = 3, K = 3, seed = 9, ncores = 1)
fit <- netcrop_lsm(
  A, d_candidates = 1:5, num_subnetworks = 2, overlap_size = 30,
  ncores = 1, seed = 10, verbose = FALSE,
  lsm_options = list(niter = 20)
)
fit
summary(fit)
```

netcrop_param_select *Select NETCROP partition parameters*

Description

Converts a requested test proportion and one or more overlap-range positions into NETCROP subnetwork counts and overlap proportions. An `o_range` value of exactly one is replaced by `0.8` to avoid the singular upper endpoint. When `n` is supplied, remainder nodes augment the overlap so the remaining nodes divide evenly among the selected subnetworks.

Usage

```
netcrop_param_select(test_prop = 0.02, n = NULL, o_range = c(0, 0.8))
```

Arguments

`test_prop`	One finite number in `(0, 0.5]` giving the target fraction of unordered node pairs held out by a NETCROP split.
`n`	Optional integer network size from 3 through R's maximum supported integer. When supplied, integer subnetwork and overlap sizes are returned.
`o_range`	Numeric vector with values in `[0, 1]` locating candidate overlaps between the lower and upper proportions implied by `test_prop`.

Value

A named list describing candidate subnetwork counts and overlap proportions. When `n` is supplied, it also contains integer overlap and piece sizes, remainder adjustments, and realized test proportions.

See Also

`netcrop_blockmodel()`, `netcrop_rdpg()`, `netcrop_lsm()`

Examples

```
netcrop_param_select(test_prop = 0.05, n = 200, o_range = c(0, 0.8))
```

netcrop_rdpg *Select RDPG dimension with NETCROP*

Description

Selects a symmetric random-dot-product-graph (RDPG) dimension by overlapping-subnetwork cross-validation. The procedure decomposes every subnetwork, constructs each candidate embedding, aligns non-reference embeddings through the overlap, and evaluates every unordered pair of non-overlap pieces.

Usage

```
netcrop_rdpg(
  A,
  d_candidates,
  num_subnetworks = NULL,
  overlap_size = NULL,
  nrep = 1L,
  losses = "sse",
  eig_options = list(),
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
  seed = NULL,
  verbose = TRUE,
  force_windows = FALSE,
  ram_check = FALSE,
  parameter_select_options = list(),
  retain_intermediates = c("all", "minimal")
)

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

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

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

## S3 method for class 'netcrop_rdpg'
plot(x, aggregate = TRUE, ...)
```

Arguments

A	Finite square symmetric numeric adjacency matrix with zero diagonal, including a supported sparse Matrix object.
d_candidates	Non-empty vector of positive integer latent dimensions; duplicates are discarded. Dimensions must be smaller than the effective subgraph size; conventionally the candidate vector is 1:5.
num_subnetworks	Optional integer number of subnetworks, at least 2. If it or overlap_size is NULL, missing values are selected with netcrop_param_select().
overlap_size	Optional positive integer overlap size. It must leave enough non-overlap nodes for the requested subnetworks and dimensions.
nrep	Positive integer number of independent NETCROP repetitions.
losses	Character vector naming prediction-loss functions available in the calling environment, such as "sse".
eig_options	Named list of additional options passed to eig_decomp().

ncores	Positive integer worker count used across decomposition, embedding, alignment, and loss tasks.
seed	Optional nonnegative integer-like reproducibility seed.
verbose	Whether to print progress messages.
force_windows	Whether to force the Windows-compatible parallel backend, including for backend testing on other platforms.
ram_check	Whether to run RAM preflight checks before major operations.
parameter_select_options	Named list of additional options passed to netcrop_param_select() when partition parameters are selected automatically.
retain_intermediates	Either "all" or "minimal", controlling how many embeddings and intermediate arrays are retained.
x	A netcrop_rdpg or summary.netcrop_rdpg object. For plotting, it is the first result to display or compare.
...	Additional plotting arguments or further netcrop_rdpg objects for comparison; ignored by print and summary methods.
object	A fitted netcrop_rdpg object to summarize.
aggregate	Whether a plot should aggregate loss curves across repetitions. A netcrop_rdpg object in this position instead starts a comparison plot.

Value

netcrop_rdpg() returns an object of class netcrop_rdpg containing candidate-wise cross-validation losses, repetition and overall dimension selections, split and eigenvalue diagnostics, retained intermediates, resource information, and timing. summary.netcrop_rdpg() returns a compact summary object; print methods return their input invisibly, and the plot method returns a ggplot object.

See Also

ecv_stability_rdpg(), ase()

Examples

```
A <- generate_rdpg(n = 200, d = 3, seed = 7, ncores = 1)
fit <- netcrop_rdpg(
  A, d_candidates = 1:5, num_subnetworks = 2, overlap_size = 30,
  ncores = 1, seed = 8, verbose = FALSE
)
fit
summary(fit)
```

netcrop_tune_regularizer *Tune a network regularizer with NETCROP*

Description

Selects a spectral regularization parameter by overlapping-subnetwork cross-validation for an SBM or DCBM.

Usage

```
netcrop_tune_regularizer(
  A,
  K,
  tau_candidates,
  use_dcbm = FALSE,
  num_subnetworks = NULL,
  overlap_size = NULL,
  nrep = 1L,
  use_laplacian = FALSE,
  dcbm_est_method = c("plugin", "spectral"),
  losses = "sse",
  loss_types = NULL,
  label_reference = c("full_network", "leave_pair_out"),
  spectral_options = list(),
  cluster_engine = c("clara", "kmeans", "pam"),
  cluster_options = list(),
  estimator_options = list(),
  matching_method = c("greedy", "hungarian", "brute_force"),
  confirm_large = NULL,
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
  seed = NULL,
  verbose = TRUE,
  force_windows = FALSE,
  ram_check = FALSE,
  parameter_select_options = list(),
  retain_intermediates = c("all", "minimal")
)

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

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

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

```
## S3 method for class 'netcrop_regularizer'
plot(x, aggregate = TRUE, ...)
```

Arguments

A	Finite symmetric, loop-free adjacency matrix.
K	Fixed positive number of communities.
tau_candidates	Unique finite nonnegative regularization candidates.
use_dcbm	Use a degree-corrected rather than ordinary block model.
num_subnetworks	Number of overlapping subnetworks; NULL selects it automatically.
overlap_size	Number of overlap nodes; NULL selects it automatically.
nrep	Positive number of independently sampled partitions.
use_laplacian	Cluster a regularized graph Laplacian.
dcbm_est_method	DCBM estimator, either "plugin" or "spectral".
losses	Canonical validation losses to evaluate.
loss_types	Optional named mapping for compatible legacy loss labels.
label_reference	Whether label losses use a full-network reference or a leave-pair-out reference.
spectral_options	Named list of additional options passed to spectral_cluster().
cluster_engine	Clustering backend used by spectral_cluster(): "clara" (the default), "kmeans", or "pam".
cluster_options	Named list of additional options passed to the function selected by cluster_engine: cluster::clara(), stats::kmeans(), or cluster::pam(). Defaults are chosen for the selected backend and model, then overridden by this list.
estimator_options	Named list of additional options passed to estimate_sbm() for SBM fitting or estimate_dcbm() for DCBM fitting.
matching_method	Label-alignment method.
confirm_large	Confirm potentially factorial brute-force matching.
ncores	Positive worker count.
seed	Optional nonnegative reproducibility seed.
verbose	Print progress messages.
force_windows	Use the Windows-compatible parallel backend.
ram_check	Report conservative RAM demand.
parameter_select_options	Named list of additional options passed to netcrop_param_select() for automatic partition selection.

`retain_intermediates`
: Retain all or minimal intermediate results.

`x, object`
: A fitted `netcrop_regularizer` object handled by an S3 method.

`...`
: Additional arguments for S3 method compatibility.

`aggregate`
: Aggregate repetitions when plotting.

Details

The function clusters overlapping subnetworks for every candidate, aligns their labels, estimates block-model probabilities, and evaluates held-out edges between non-overlap pieces. Missing partition parameters are selected with `netcrop_param_select()`. `K = 1` returns invisibly because no clustering-based regularizer selection is required. The spectral and clustering defaults match `netcrop_blockmodel()`: eigenvectors from `RSpectra`, CLARA clustering, Euclidean distance for SBM, Manhattan distance for DCBM, and row normalization for DCBM. User option lists override the corresponding defaults without changing tuner-controlled inputs.

Value

A `netcrop_regularizer` object containing candidate loss curves, per-repetition and overall selections, resolved options, partition and matching diagnostics, worker counts, timings, and optional intermediates.

See Also

`dkest_tune_regularizer()`, `mult_reg_spectral_cluster()`

Examples

```
A <- generate_sbm(n = 200, K = 3, alpha = 0.5, beta = 0.1,
                  seed = 31, ncores = 1)
fit <- netcrop_tune_regularizer(
  A, K = 3, tau_candidates = c(0, 0.1),
  num_subnetworks = 2, overlap_size = 20,
  nrep = 1, ncores = 1, seed = 32, verbose = FALSE
)
fit
```

---

network_generators *Network input, generation, and probability calibration*

Description

Read and write edge lists and generate ER, SBM, DCBM, RDPG, and latent-space networks. Generators return an adjacency matrix with compact truth metadata available through `get_generator_parameters()`.

nmi *Compute normalized mutual information*

Description

Computes empirical mutual information divided by joint entropy. This normalization differs from definitions based on marginal entropies. A constant joint labeling has zero entropy and returns NaN.

Usage

```
nmi(g_1, g_2)
```

Arguments

g_1, g_2	Equal-length, non-missing atomic label vectors.

Value

A numeric agreement score.

See Also

label_match_greedy(), pair_nmi_loss()

Examples

```
nmi(c(1, 1, 2, 2), c(2, 2, 1, 1))
```

normalize_lsm_positions *Normalize latent-space positions*

Description

Centers and scales a latent-position matrix using the LSM convention.

Usage

```
normalize_lsm_positions(Z)
```

Arguments

Z	A finite numeric latent-position matrix.

Value

A centered and normalized matrix with the same dimensions as Z.

See Also

generate_lsm_positions(), generate_lsm()

Examples

```
Z <- matrix(stats::rnorm(200 * 3), 200, 3)
dim(normalize_lsm_positions(Z))
```

`oracle_plotter` *Compare clustering methods with known labels*

Description

Evaluates SONNET, spectral clustering, or both against known community labels over candidate regularizers.

Usage

```
oracle_plotter(
  A,
  g_true,
  K = NULL,
  netcrop_outcomes = NULL,
  dkest_outcomes = NULL,
  include_netcrop_mean = TRUE,
  include_netcrop_mode = TRUE,
  losses = NULL,
  engines = c("sonnet", "spectral_cluster"),
  include_sonnet_tau_zero = TRUE,
  include_spectral_tau_zero = TRUE,
  matching_method = c("greedy", "hungarian", "brute_force"),
  confirm_large = NULL,
  sonnet_options = list(),
  spectral_cluster_options = list(),
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
  seed = NULL,
  verbose = TRUE,
  force_windows = FALSE,
  ram_check = FALSE
)
```

Arguments

`A`	One adjacency matrix or a list matching `g_true`.
`g_true`	Ground-truth labels or a list matching `A`.
`K`	Optional fixed number of communities; inferred from `g_true` when omitted.

`netcrop_outcomes`, `dkest_outcomes`
: Matching tuner results. At least one must be supplied; lists must contain one outcome per network.

`include_netcrop_mean`, `include_netcrop_mode`
: Include NETCROP repetition summaries in addition to its first repetition.

`losses`
: Optional NETCROP loss names to include.

`engines`
: Fit `"sonnet"`, `"spectral_cluster"`, or both.

`include_sonnet_tau_zero`, `include_spectral_tau_zero`
: Add a separate unregularized baseline for the corresponding engine. These evaluations default to `TRUE` and do not affect the tuner-derived oracle grid.

`matching_method`
: Label-alignment method used for accuracy.

`confirm_large`
: Confirm potentially factorial brute-force matching.

`sonnet_options`
: Named list of additional options passed to `sonnet()`.

`spectral_cluster_options`
: Named list of additional options passed to `spectral_cluster()`.

`ncores`
: Positive worker count.

`seed`
: Optional nonnegative reproducibility seed.

`verbose`
: Print progress messages.

`force_windows`
: Use the Windows-compatible parallel backend.

`ram_check`
: Report conservative RAM demand.

Details

The oracle candidate grid is obtained from the supplied NETCROP and DKEST outcomes. If their grids differ, the function warns and uses their floating-point-tolerant intersection; an empty intersection is an error. Selected off-grid values are still evaluated. Optional engine-specific `tau = 0` baselines do not expand the grid used to define the oracle. Accuracy is one minus the validated label-matching mismatch rate; multiple networks are summarized with means and standard deviations.

Value

Invisibly returns the rendered `ggplot`. Raw accuracy data, aggregated plotting data, effective fit metadata, and diagnostics are attached as attributes.

See Also

`spectral_cluster()`, `sonnet()`

Examples

```
A <- generate_sbm(n = 200, K = 3, alpha = 0.5, beta = 0.1,
                  seed = 41, ncores = 1)
truth <- get_generator_parameters(A)$g_true
netcrop_fit <- netcrop_tune_regularizer(
  A, K = 3, tau_candidates = c(0, 0.1),
```

```
  num_subnetworks = 2, overlap_size = 20,
  nrep = 1, ncores = 1, seed = 42, verbose = FALSE
)
oracle_plotter(
  A, g_true = truth, K = 3,
  netcrop_outcomes = netcrop_fit,
  engines = "spectral_cluster", ncores = 1, seed = 43,
  verbose = FALSE,
  spectral_cluster_options = list(spectral_engine = "base")
)
```

`outer_add` *Add vectors by outer expansion*

Description

Forms all pairwise sums, using a registered compiled implementation when requested and available and otherwise `base::outer()`.

Usage

```
outer_add(x, y, operator = "+", use_cpp = TRUE)
```

Arguments

`x, y`	Numeric vectors.
`operator`	Outer operation; currently only `"+"` is supported.
`use_cpp`	Try the compiled implementation before the R implementation.

Value

A numeric matrix with `length(x)` rows and `length(y)` columns.

See Also

`procrustes()`

Examples

```
outer_add(1:2, 3:4, use_cpp = FALSE)
```

`pair_nmi_loss` *Compare pairs of clusterings*

Description

Forms the Cartesian-product clustering induced by each pair of community label vectors, then compares the two induced clusterings.

Usage

```
pair_nmi_loss(g_1, g_2, g_3, g_4)

pair_hamming_loss(g_1, g_2, g_3, g_4)
```

Arguments

`g_1, g_2`	Finite positive-integer label vectors defining the first Cartesian-product clustering.
`g_3, g_4`	Finite positive-integer label vectors defining the second Cartesian-product clustering. Its product size must match the first.

Details

`pair_nmi_loss()` returns negative normalized mutual information. If both product clusterings contain only one class, it returns `-1`. `pair_hamming_loss()` aligns the product labels with the dependency-free Hungarian implementation before returning their mismatch proportion.

Value

`pair_nmi_loss()` returns a numeric negative-NMI loss. `pair_hamming_loss()` returns a numeric mismatch proportion.

See Also

`nmi()`, `label_match_greedy()`

Examples

```
pair_nmi_loss(c(1, 1, 2), c(1, 2), c(1, 1, 2), c(1, 2))
pair_hamming_loss(c(1, 1, 2), c(1, 2), c(2, 2, 1), c(2, 1))
```

`procrustes` *Align point configurations by Procrustes transformation*

Description

Aligns `X` to `X_star` by an orthogonal transformation with optional centering and dilation. Inputs may have different column counts; the narrower configuration is padded with zero columns. A compiled translated, non-dilated path is used when requested and available.

Usage

```
procrustes(
  X,
  X_star,
  translate = FALSE,
  dilate = FALSE,
  sumsq = FALSE,
  use_cpp = TRUE,
  validate_inputs = TRUE
)
```

Arguments

`X`	Numeric point-configuration matrix to align.
`X_star`	Numeric reference point-configuration matrix with the same number of rows as `X`.
`translate`	Center configurations and estimate translation.
`dilate`	Estimate and apply a dilation factor.
`sumsq`	Include residual sum of squares.
`use_cpp`	Try the registered compiled implementation.
`validate_inputs`	Validate matrices and logical controls.

Value

A list containing aligned `X_new` and `rotation`, with optional `translation`, `scale`, and `sse` components.

See Also

`ase()`, `generate_latent_positions()`

Examples

```
X <- matrix(c(0, 0, 1, 0, 0, 1), ncol = 2, byrow = TRUE)
procrustes(X, X, use_cpp = FALSE)$X_new
```

`run_simulations` *Run reproducible simulation jobs*

Description

Executes a simulation function repeatedly with optional outer-loop parallelism, reproducible per-simulation seeds, durable RDS results, resuming, and resource measurements. `one_simulation` must accept a named `simulation` argument. Values supplied through `...` are forwarded unchanged.

Usage

```
run_simulations(
  one_simulation,
  nsim,
  ...,
  use_parallel_simulations = FALSE,
  ncores_outer = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
  seed = NULL,
  seeds = NULL,
  results_file = NULL,
  action = c("replace", "resume", "archive"),
  retry_failed = TRUE,
  continue_on_error = TRUE,
  measure_resources = FALSE,
  show_progress = TRUE,
  force_windows = FALSE
)
```

Arguments

`one_simulation`	Function implementing one simulation. It must accept a named `simulation` argument containing the one-based simulation number.
`nsim`	Positive number of simulations.
`...`	Additional arguments forwarded to `one_simulation`.
`use_parallel_simulations`	Parallelize the outer simulation loop. The default avoids accidental nested parallelism.
`ncores_outer`	Positive number of outer-loop workers.
`seed`	Optional nonnegative base seed. Simulation `i` uses `seed + i - 1`, reduced to R's supported integer range.
`seeds`	Optional vector of `nsim` explicit nonnegative seeds. Supply either `seed` or `seeds`, not both.
`results_file`	Optional trusted RDS path used for durable results.
`action`	How to handle an existing results file: `"replace"`, `"resume"`, or `"archive"`.

`retry_failed` When resuming, rerun records whose success field is `FALSE`.

`continue_on_error`
: Return all records when simulations fail. If `FALSE`, save available records and stop with a summary error.

`measure_resources`
: Measure elapsed time and peak additional R-managed RAM for each simulation.

`show_progress` Print a compact status line for each newly run simulation when supported by the backend.

`force_windows` Use the Windows-compatible future backend.

Details

When `results_file` is supplied, each simulation is stored as a record with its number, seed, success status, elapsed time, optional peak RAM, returned value, and error message. Existing results can be replaced, resumed, or archived.

Value

A list of `nsim` simulation records, invisibly when a resumed run has no work remaining.

See Also

`uni_mclapply()`, `get_generator_parameters()`

Examples

```
run_simulations(
  one_simulation = function(simulation, offset) simulation + offset,
  nsim = 2, offset = 10, seed = 1, show_progress = FALSE
)
```

---

scale_lsm_to_average_degree

Scale latent-space logits to a target average degree

Description

Shifts an LSM logit matrix using the historical iterative method or calibrated bisection.

Usage

```
scale_lsm_to_average_degree(
  theta,
  average_degree,
  average_degree_method = c("naive", "calibrated"),
  self_loops = FALSE,
```

```
  lower_clip = 0,
  upper_clip = 1,
  naive_iterations = 10L,
  tolerance = 1e-08,
  max_iterations = 100L
)
```

Arguments

`theta`	Finite square latent-space logit matrix.
`average_degree`	Target expected average degree.
`average_degree_method`	Scaling method, either `"naive"` or `"calibrated"`.
`self_loops`	Whether diagonal probabilities count toward degree.
`lower_clip, upper_clip`	Finite probability clipping bounds.
`naive_iterations`	Positive iteration count for historical scaling.
`tolerance`	Positive bisection convergence tolerance.
`max_iterations`	Positive maximum number of bisection iterations.

Value

A list containing the probability matrix `P` and `intercept_shift`.

See Also

`generate_lsm()`, `scale_to_average_degree()`

Examples

```
Z <- matrix(stats::rnorm(200 * 3), 200, 3)
theta <- -tcrossprod(Z)
scaled <- scale_lsm_to_average_degree(theta, average_degree = 5)
mean(rowSums(scaled$P))
```

---

scale_to_average_degree

Scale probabilities to a target average degree

Description

These twin functions apply calibrated bisection or the historical one-step scaling rule to a dense or sparse probability matrix.

`scale_to_average_degree_naive()` applies the historical one-step ratio before clipping and loop removal.

Usage

```
scale_to_average_degree(
  P,
  average_degree,
  self_loops,
  lower_clip,
  upper_clip,
  tolerance = 1e-08,
  max_iterations = 100L
)

scale_to_average_degree_naive(
  P,
  average_degree,
  self_loops,
  lower_clip,
  upper_clip
)
```

Arguments

P	Dense or sparse edge-probability matrix.
average_degree	Target expected average row degree.
self_loops	Whether diagonal probabilities count toward degree.
lower_clip, upper_clip	Finite probability clipping bounds.
tolerance	Positive bisection convergence tolerance.
max_iterations	Positive maximum number of bisection iterations.

Value

A list containing the scaled probability matrix P and multiplier.

See Also

apply_average_degree_scaling(), scale_lsm_to_average_degree()

Examples

```
P <- matrix(0.1, 200, 200)
diag(P) <- 0
scaled <- scale_to_average_degree(
  P, average_degree = 5, self_loops = FALSE,
  lower_clip = 0, upper_clip = 1
)
mean(rowSums(scaled$P))
```

`shortest_path_distances` *Compute shortest-path distances*

Description

Computes all-pairs shortest-path distances without requiring igraph. Rows are sources, columns are destinations, diagonal entries are zero, and unreachable pairs are `Inf`. Undirected weighted paths use the smaller nonzero weight when reciprocal weights differ.

Usage

```
shortest_path_distances(
  A,
  directed = FALSE,
  weighted = FALSE,
  self_loops = c("ignore", "include"),
  use_cpp = TRUE
)
```

Arguments

`A`	A finite square dense or sparse adjacency matrix.
`directed`	Preserve edge direction.
`weighted`	Treat nonzero entries as nonnegative path lengths rather than unit-length edges.
`self_loops`	Include or ignore diagonal edges.
`use_cpp`	Try a registered compiled implementation before the R implementation.

Value

A numeric all-pairs distance matrix.

See Also

`adjacency_neighbors()`, `connected_components()`

Examples

```
A <- matrix(c(0, 1, 0, 1, 0, 1, 0, 1, 0), 3, 3)
shortest_path_distances(A, use_cpp = FALSE)
```

sigmoid *Compute the sigmoid transform*

Description

Evaluates the numerically stable logistic sigmoid elementwise.

Usage

```
sigmoid(x)
```

Arguments

x	Numeric vector or matrix.

Value

A numeric object matching the shape of x.

See Also

softplus()

Examples

```
sigmoid(c(-1, 0, 1))
```

singular_decomp *Compute a singular-value decomposition*

Description

Computes selected singular values and optional left and right vectors. Partial backends fall back to base::svd() unless force_engine = TRUE.

Usage

```
singular_decomp(
  A,
  d,
  only_values = FALSE,
  nu = if (only_values) 0L else d,
  nv = if (only_values) 0L else d,
  scale_by = c("none", "dimension", "sparsity"),
  use_laplacian = FALSE,
  engine = c("rspectra", "base", "irlba"),
```

```
  force_engine = FALSE,
  order_by = c("value", "magnitude"),
  safe_d_multiplier = 1,
  validate_inputs = TRUE
)
```

Arguments

A	A finite dense or sparse matrix.
d	Positive number of singular components to return.
only_values	Return singular values without singular vectors.
nu, nv	Numbers of left and right singular vectors to return.
scale_by	Scale by "none", matrix "dimension", or entry "sparsity".
use_laplacian	Transform A with graph_laplacian() first.
engine	Decomposition backend: "rspectra", "base", or "irlba".
force_engine	Stop instead of falling back when the selected partial backend is unavailable or fails.
order_by	Component ordering convention; singular values make "value" and "magnitude" equivalent.
safe_d_multiplier	Multiplier for extra partial components computed before selecting the requested components.
validate_inputs	Validate matrix and option inputs.

Value

A list with values and requested u and v matrices.

See Also

eig_decomp(), truncated_svd_reconstruct()

Examples

```
singular_decomp(diag(c(3, 2, 1)), d = 2, engine = "base")
```

softplus *Compute the softplus transform*

Description

Evaluates the numerically stable softplus transform elementwise.

Usage

```
softplus(x)
```

Arguments

x	Numeric vector or matrix.

Value

A numeric object matching the shape of x.

See Also

sigmoid()

Examples

```
softplus(c(-1, 0, 1))
```

sonnet *Fit a SONNET network clustering*

Description

Fits scalable network clustering by partitioning a dense or sparse network into overlapping subnetworks, clustering each subnetwork, aligning labels through their overlaps, and aggregating repetition-level memberships. The overlap may be shared across repetitions or sampled independently.

Usage

```
sonnet(
  A,
  K,
  num_subnetworks = NULL,
  overlap_size = NULL,
  extra_nrep = 0L,
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
  seed = NULL,
```

```
  matching_method = c("greedy", "hungarian", "brute_force"),
  confirm_large = TRUE,
  verbose = TRUE,
  force_windows = FALSE,
  ram_check = FALSE,
  share_overlap = FALSE,
  parameter_select_options = list(),
  laplacian = FALSE,
  regularize_tau = 0,
  regularize_subnetworks = TRUE,
  ...
)

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

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

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

Arguments

A	Non-empty finite square numeric adjacency matrix, including a supported sparse Matrix object.
K	Positive integer number of communities, no larger than nrow(A).
num_subnetworks	Optional positive integer number of subnetworks. If it or overlap_size is NULL, missing partition values are selected with sonnet_param_select().
overlap_size	Optional positive integer number of overlap nodes, no larger than nrow(A).
extra_nrep	Nonnegative integer number of additional partition and clustering repetitions beyond the initial run.
ncores	Positive integer worker count used across fitting and label alignment tasks.
seed	Optional nonnegative integer-like reproducibility seed.
matching_method	Label-alignment method: "greedy", "hungarian", or "brute_force".
confirm_large	Whether to require confirmation before an unusually large brute-force label-matching problem.
verbose	Whether to print progress and timing messages.
force_windows	Whether to force the Windows-compatible parallel backend, including for backend testing on other platforms.
ram_check	Whether to run RAM preflight checks before major operations.
share_overlap	Whether every repetition uses the same overlap nodes. If FALSE, each repetition samples its overlap independently.

parameter_select_options
: Named list of additional options passed to sonnet_param_select() when partition parameters are selected automatically.

laplacian
: Whether to convert adjacency matrices to graph Laplacians before spectral clustering.

regularize_tau
: Nonnegative Laplacian regularization strength.

regularize_subnetworks
: Whether Laplacian conversion and regularization are performed separately within each subnetwork. If FALSE, the full network is transformed before subnetworks are extracted.

...
: Additional named arguments passed to spectral_cluster(). A, U, and K are reserved and cannot be supplied here. For the print and summary methods, additional arguments are currently ignored.

x
: A fitted sonnet or summary.sonnet object to print.

object
: A fitted sonnet object to summarize.

Value

sonnet() returns a fitted object of class sonnet containing final labels, aligned labels and memberships, subnetwork fits and node sets, parameters, core allocation, and timing. summary.sonnet() returns a summary.sonnet object. Print methods return their input invisibly.

See Also

sonnet_shared_overlap(), sonnet_independent_overlap(), spectral_cluster()

Examples

```
A <- generate_sbm(n = 200, K = 3, alpha = 0.5, beta = 0.1,
                  seed = 3, ncores = 1)
fit <- sonnet(
  A, K = 3, num_subnetworks = 2, overlap_size = 30,
  ncores = 1, seed = 4, verbose = FALSE, spectral_engine = "base"
)
fit
summary(fit)
```

---

sonnet_param_select *Select SONNET partition parameters*

Description

Selects the number of subnetworks and their common overlap by a constrained integer grid search. Computational cost is evaluated as s * ((o + (n - o) / s) / n)^theta without forming n^theta. Feasible candidates have cost no greater than q * ncores, and (n - o) must be divisible by s. If none meets the requested budget, the least-cost admissible candidate is returned and the effective q is increased.

Usage

```
sonnet_param_select(
  n,
  K = 5L,
  theta = 3,
  q = 0.005,
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
  s_lower = 2,
  s_upper = 50,
  o_lower = NULL,
  o_upper = NULL
)
```

Arguments

`n`	Positive integer network size, at least 3.
`K`	Positive integer number of communities, no larger than `n`; used to choose the default lower overlap bound.
`theta`	Positive computational-complexity exponent.
`q`	Number in `(0, 1]` giving the target sequential computational fraction per available core.
`ncores`	Positive integer number of cores included in the computational budget.
`s_lower`, `s_upper`	Numeric lower and upper bounds for the number of subnetworks. Bounds are normalized to admissible integers.
`o_lower`, `o_upper`	Optional numeric lower and upper bounds for overlap size. `NULL` uses `min(K^3, n - s_lower)` for the lower bound and the largest admissible overlap for the upper bound.

Value

A named list containing the selected `num_subnetworks` and `overlap_size`, piece size, objective and computational fractions, requested and effective tuning values, search counts, and normalized bounds.

See Also

`sonnet()`, `netcrop_param_select()`

Examples

```
sonnet_param_select(n = 200, K = 3, ncores = 1)
```

sonnet_shared_overlap *Fit SONNET overlap variants*

Description

Fit SONNET using either one overlap set shared by every repetition or an independently sampled overlap set in each repetition. Both wrappers run the partition, spectral-clustering, label-alignment, and modal-aggregation workflow used by `sonnet()`.

Usage

```
sonnet_shared_overlap(...)

sonnet_independent_overlap(...)
```

Arguments

`...`	Named arguments passed to the SONNET fitting engine: `A`, `K`, `num_subnetworks`, `overlap_size`, `extra_nrep`, `ncores`, `seed`, `matching_method`, `confirm_large`, `verbose`, `force_windows`, `ram_check`, `parameter_select_options`, `laplacian`, `regularize_tau`, and `regularize_subnetworks`, plus supported `spectral_cluster()` arguments. Do not supply `share_overlap`; each wrapper sets it.

Value

A fitted object of class `sonnet`, containing final labels, repetition-level labels and memberships, split information, parameters, core allocation, timing, and the matched call.

See Also

`sonnet()`, `sonnet_param_select()`

Examples

```
A <- generate_sbm(n = 200, K = 3, alpha = 0.5, beta = 0.1,
                  seed = 1, ncores = 1)
shared <- sonnet_shared_overlap(
  A = A, K = 3, num_subnetworks = 2, overlap_size = 30,
  extra_nrep = 1, ncores = 1, seed = 2, verbose = FALSE,
  spectral_engine = "base"
)
independent <- sonnet_independent_overlap(
  A = A, K = 3, num_subnetworks = 2, overlap_size = 30,
  extra_nrep = 1, ncores = 1, seed = 2, verbose = FALSE,
  spectral_engine = "base"
)
```

spectral_cluster *Cluster a network spectrally*

Description

Clusters a dense or sparse network into K communities using a configurable spectral representation and clustering backend.

Usage

```
spectral_cluster(
  A = NULL,
  U = NULL,
  K,
  laplacian = FALSE,
  normalize_laplacian = TRUE,
  regularize_tau = 0,
  handle_zero_degree_nodes = c("none", "random_label", "remove"),
  row_normalize = FALSE,
  spectral_method = c("eigen", "svd"),
  spectral_engine = c("RSpectra", "irlba", "base"),
  spectral_options = list(),
  cluster_engine = c("clara", "kmeans", "pam"),
  cluster_options = list(),
  ram_check = FALSE,
  validate_inputs = TRUE
)
```

Arguments

A	Optional finite square dense or sparse adjacency matrix. Supply either A or U.
U	Optional finite matrix whose rows are clustered directly. Supply either U or A.
K	Positive number of communities.
laplacian	Whether to cluster a graph-Laplacian representation.
normalize_laplacian	Whether a requested Laplacian is normalized.
regularize_tau	Nonnegative adjacency regularization parameter.
handle_zero_degree_nodes	Policy for zero-degree nodes: leave them in the fit, assign random labels, or remove them during clustering.
row_normalize	Whether to normalize spectral-representation rows.
spectral_method	Whether to use an eigendecomposition or SVD.
spectral_engine	Spectral backend: "RSpectra", "irlba", or base R.

spectral_options
: Named list of additional options passed to eig_decomp() when spectral_method = "eigen" or singular_decomp() when spectral_method = "svd".

cluster_engine
: Clustering backend: "clara", "kmeans", or "pam".

cluster_options
: Named list of additional options passed to cluster::clara(), stats::kmeans(), or cluster::pam(), according to cluster_engine.

ram_check
: Whether to report a conservative RAM preflight estimate.

validate_inputs
: Whether to validate inputs; only audited internal callers should disable this.

Details

If U is supplied, it is clustered directly and A, including its degree information, is ignored. Otherwise asymmetric A is converted to A + t(A). Regularization uses $A_\tau = A + \tau\,\mathrm{mean}(degree)/n$ before direct decomposition or Laplacian construction.

Value

A fitted spectral-clustering object with labels and diagnostics.

See Also

ase(), estimate_sbm(), sonnet()

Examples

```
A <- generate_sbm(n = 200, K = 3, alpha = 0.4, beta = 0.1,
                  seed = 9, ncores = 1)
fit <- spectral_cluster(A, K = 3, spectral_engine = "base",
                        cluster_engine = "kmeans")
table(fit$g_hat)
```

---

squared_error_losses *Compute squared-error losses*

Description

sse() returns the sum of squared errors; mse() returns their mean.

Usage

```
sse(x, y, na_rm = FALSE, validate_inputs = TRUE)

mse(x, y, na_rm = FALSE, validate_inputs = TRUE)
```

Arguments

x, y	Equal-length numeric vectors.
na_rm	Remove missing comparisons.
validate_inputs	Validate input types, lengths, and na_rm.

Value

A nonnegative numeric scalar.

See Also

sae(), mae()

Examples

```
sse(c(1, 2), c(1, 3))
mse(c(1, 2), c(1, 3))
```

---

truncated_svd_reconstruct

Reconstruct a truncated singular-value approximation

Description

Reconstructs U_d diag(s_d) V_d' without materializing the diagonal matrix, then clips entries to the requested interval. The default bounds produce a probability-matrix estimate; infinite bounds leave the reconstruction unmodified.

Usage

```
truncated_svd_reconstruct(
  A,
  d,
  lower_clip = 0,
  upper_clip = 1,
  engine = c("rspectra", "base", "irlba"),
  force_engine = FALSE,
  safe_d_multiplier = 1
)
```

Arguments

A	A finite dense or sparse matrix.
d	Positive reconstruction rank.
lower_clip, upper_clip	Inclusive clipping bounds.
engine	Singular-decomposition backend.
force_engine	Stop instead of falling back when the backend fails.
safe_d_multiplier	Multiplier for extra partial components.

Value

A numeric rank-d matrix approximation.

See Also

singular_decomp(), usvt()

Examples

```
truncated_svd_reconstruct(diag(3), d = 1, engine = "base")
```

---

uni_mclapply *Apply a function consistently across platforms*

Description

Applies FUN to each element of X. Unix-like systems first attempt parallel::mclapply(). Windows, or force_windows = TRUE, uses future.apply::future_lapply(). A temporary multisession plan is created and restored by default. Backend failures fall back to base::lapply() unless stop_on_error = TRUE.

Usage

```
uni_mclapply(
  X,
  FUN,
  ...,
  ncores = max(floor(parallel::detectCores()/2), 1L, na.rm = TRUE),
  force_windows = FALSE,
  stop_on_error = FALSE,
  manage_future_plan = TRUE,
  future_packages = NULL,
  suppress_worker_renv_sync_check = TRUE
)
```

Arguments

X	A vector, list, or other collection accepted by lapply-style functions.
FUN	Function applied to each element of X.
...	Additional arguments forwarded to FUN.
ncores	Positive number of workers. The default is half the detected logical cores, rounded down, with a minimum of one.
force_windows	Use the Windows-compatible future backend even on another operating system.
stop_on_error	Stop with failed task indices and worker messages instead of falling back or returning worker-level errors.
manage_future_plan	Create and restore a temporary multisession plan. Set FALSE only when the caller manages the future plan.
future_packages	Optional package names loaded by future workers, including packages needed for S4 method registration.
suppress_worker_renv_sync_check	Temporarily suppress repeated renv synchronization messages while multisession workers start.

Value

A list containing one result per element of X.

See Also

run_simulations(), sonnet()

Examples

```
uni_mclapply(1:3, function(x) x^2, ncores = 1)
```

---

usvt *Estimate a matrix with universal singular-value thresholding*

Description

Applies the historical pasted USVT rule d = ceiling(n^(1/3)), reconstructs that truncated SVD, and clips every entry to the supplied interval.

Usage

```
usvt(
  A,
  lower_clip = 0,
  upper_clip = 1,
  engine = c("irlba", "rspectra", "base"),
  force_engine = FALSE,
  safe_d_multiplier = 1
)
```

Arguments

A	A finite nonempty square dense or sparse matrix.
lower_clip, upper_clip	Inclusive clipping bounds.
engine	Singular-decomposition backend.
force_engine	Stop instead of falling back when the backend fails.
safe_d_multiplier	Multiplier for extra partial components.

Value

A dense thresholded matrix estimate.

See Also

singular_decomp(), ase()

Examples

```
usvt(diag(3), engine = "base")
```

---

write_network *Read and write network edge lists*

Description

write_network() writes a dense or sparse network as an edge list with two node columns and an optional weight column. Metadata preserves network size, direction, and weighting, including when isolated nodes are absent.

read_network() reconstructs an edge list as a dense matrix or general sparse dgCMatrix. For undirected input, each off-diagonal edge must occur only once.

Usage

```
write_network(
  A,
  file,
  directed = NULL,
  weighted = NULL,
  triangle = c("upper", "lower"),
  format = c("auto", "csv", "tsv"),
  include_header = TRUE,
  overwrite = FALSE,
  tolerance = 1e-10
)

read_network(
  file,
  representation = c("sparse", "dense"),
  directed = NULL,
  weighted = NULL,
  n = NULL,
  format = c("auto", "csv", "tsv"),
  has_header = TRUE
)
```

Arguments

A	A finite square dense or sparse network matrix to write.
file	Edge-list file path.
directed	Optional direction indicator; inferred from metadata or symmetry when omitted.
weighted	Optional weight indicator. For read_network(), FALSE treats a two-column edge list as binary and ignores a third weight column with a warning; NULL infers weighting from the number of columns. For write_network(), NULL infers weighting from the stored edge values.
triangle	For undirected output, which matrix triangle to write.
format	Delimited format: "auto", "csv", or "tsv".
include_header	Whether write_network() writes column names.
overwrite	Whether write_network() may replace an existing file.
tolerance	Numerical tolerance used when checking symmetry and weights.
representation	Whether read_network() returns a sparse or dense matrix.
n	Optional network size used by read_network() when metadata cannot determine it.
has_header	Whether an input edge list has a header row.

Value

write_network() invisibly returns file; read_network() returns a dense matrix or sparse dgCMatrix, according to representation.

See Also

generate_adjacency()

Examples

```
A <- generate_er(n = 200, average_degree = 5, seed = 2, ncores = 1)
path <- tempfile(fileext = ".csv")
write_network(A, path, format = "csv")
A_roundtrip <- read_network(path, format = "csv")
unlink(path)
dim(A_roundtrip)
```

Index