Package: AMDconfigurations (via r-universe)

May 18, 2026

Type Package

Title Geometric Analysis of Configurations in High-Dimensional Spaces

Version 0.1.0

Description Tools for analysing the geometry of configurations in high-dimensional spaces using the Average Membership Degree (AMD) framework and synthetic configuration generation. The package supports a domain-agnostic approach to studying the shape, dispersion, and internal structure of point clouds, with applications across biological and ecological datasets, including those derived from deep-time records. The AMD framework builds on the idea that strongly coupled systems may occupy a limited set of recurrent regimes in state space, producing high-occupancy regions separated by sparsely populated transitional configurations. The package focuses on detecting these concentration patterns and quantifying their geometric definition without assuming any underlying dynamical model. It provides AMD curve computation, cluster assignment, and sigma-equivalent estimation, together with S3 methods for plotting, printing, and summarising AMD and sigma-equivalent objects. Mendoza (2025) <https://mmendoza1967.github.io/AMDconfigurations/>.

License MIT + file LICENSE

URL https://github.com/mmendoza1967/AMDconfigurations, https://mmendoza1967.github.io/AMDconfigurations/

BugReports https://github.com/mmendoza1967/AMDconfigurations/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.1)

Imports e1071, stats

Suggests testthat (>= 3.0.0), knitr, rmarkdown, pkgdown

VignetteBuilder knitr

Config/testthat/edition 3

Repository https://mmendoza1967.r-universe.dev

Date/Publication 2026-02-19 04:37:45 UTC

RemoteUrl https://github.com/mmendoza1967/amdconfigurations

RemoteRef HEAD

RemoteSha 746447861061e28f68b7a17719a7c02b2e985a49

Contents

`assign_clusters_best` *Select the best fuzzy c-means partition across repeated initialisations*

Description

This function runs fuzzy c-means clustering (`e1071::cmeans`) repeatedly with different random seeds and selects the partition that maximises an AMD-like objective:

Usage

```
assign_clusters_best(
  data,
  opt_cluster,
  nreps = 10,
  m = 2,
  iter.max = 20,
  scale_data = FALSE,
  seeds = NULL,
  preselect_top_sd = NULL
)
```

Arguments

`data`	A numeric matrix or data frame of samples × features.
`opt_cluster`	Integer; number of clusters to fit.
`nreps`	Number of repeated initialisations.
`m`	Fuzziness parameter for fuzzy c-means (default 2).

iter.max	Maximum number of iterations for fuzzy c-means.
scale_data	Logical; if TRUE, standardise features before clustering.
seeds	Optional numeric vector of seeds for deterministic behaviour. Must have length nreps. If NULL, random seeds are drawn.
preselect_top_sd	Optional integer; if provided, only the top-SD features are retained before clustering (useful for very high-dimensional data).

Details

$$\mathrm{Mpm} = \mathrm{mean}(\max_i u_i) - 1/k$$

where u_i is the membership vector of sample i. The best partition is returned, with cluster labels aligned to the original row order of the input data (rows with missing values receive NA).

Value

A list with components:

cluster Integer vector of cluster labels aligned to the original data. Rows with missing values receive NA.

membership Membership matrix from the best fuzzy c-means run.

centers Cluster centroids from the best run.

Mpm Best AMD-like objective value.

Examples

```
## Not run:
set.seed(1)
X <- matrix(rnorm(1000), nrow = 100, ncol = 10)
out <- assign_clusters_best(X, opt_cluster = 3, nreps = 20)
table(out$cluster)

## End(Not run)
```

---

compute_amd_curve *Compute the AMD curve across a range of cluster numbers*

Description

This function computes the Average Membership Deviation (AMD) curve for fuzzy c-means clustering across a sequence of cluster numbers k. For each k, multiple random initialisations are performed and the AMD value is computed as:

Usage

```
compute_amd_curve(
  data,
  its,
  nin,
  nsp,
  seeds = NULL,
  verbose = TRUE,
  plot_curve = FALSE,
  open_device = TRUE,
  scale_data = FALSE,
  iter_max = 100,
  m = 2,
  preselect_top_sd = NULL
)
```

Arguments

`data`	A numeric matrix or data frame of samples (rows) × features (columns).
`its`	Number of random initialisations per value of `k`.
`nin`	Minimum number of clusters to evaluate.
`nsp`	Maximum number of clusters to evaluate.
`seeds`	Optional numeric vector of seeds for deterministic behaviour. Must have length `its * (nsp - nin + 1)`. If `NULL`, random seeds are drawn.
`verbose`	Logical; print progress messages.
`plot_curve`	Logical; if `TRUE`, plot the AMD curve.
`open_device`	Logical; if `TRUE`, open a new graphics device for the plot.
`scale_data`	Logical; if `TRUE`, standardise features before clustering.
`iter_max`	Maximum number of iterations for fuzzy c-means.
`m`	Fuzziness parameter for fuzzy c-means (default 2).
`preselect_top_sd`	Optional integer; if provided, only the top-SD features are retained before clustering (useful for very high-dimensional data).

Details

$$\mathrm{AMD}(k) = \mathrm{mean}(\max_i u_i) - 1/k$$

where u_i is the membership vector of sample i. The optimal number of clusters is selected as the `k` that maximises the AMD peak across repetitions.

Value

A list with components:

k_opt The optimal number of clusters (maximising AMD peak).

max Vector of AMD peak values for each k.

mean Vector of mean AMD values across repetitions.

raw Matrix of AMD values (rows = repetitions, columns = k).

Examples

```
## Not run:
set.seed(1)
X <- matrix(rnorm(2000), nrow = 100, ncol = 20)
res <- compute_amd_curve(X, its = 10, nin = 2, nsp = 6)
res$k_opt

## End(Not run)
```

---

create_synthetic_samples

Generate synthetic clustered samples with isotropic Gaussian noise

---

Description

This function generates synthetic datasets composed of n_clusters Gaussian clusters in n_dim-dimensional space. Cluster centroids are placed uniformly inside a hypercube of side cube_size, and samples are drawn with isotropic Gaussian noise of standard deviation std_dev.

Usage

```
create_synthetic_samples(
  n_samples,
  n_clusters,
  std_dev,
  n_dim,
  cube_size = 100,
  standardize = FALSE,
  center = TRUE,
  scale. = TRUE
)
```

Arguments

n_samples	Total number of samples to generate.
n_clusters	Number of clusters to simulate.
std_dev	Standard deviation of the Gaussian noise around each centroid.
n_dim	Number of dimensions (features).
cube_size	Side length of the hypercube where centroids are placed.
standardize	Logical; if TRUE, standardise the final dataset (mean 0, sd 1 per feature).
center, scale.	Logical arguments passed to scale() if standardize = TRUE.

Details

The function is used internally to calibrate the compactness of real data by matching its AMD peak against synthetic datasets with varying noise levels.

Value

A data frame of size n_samples × n_dim containing the synthetic samples.

Examples

```
## Not run:
set.seed(1)
syn <- create_synthetic_samples(
  n_samples = 200,
  n_clusters = 4,
  std_dev = 5,
  n_dim = 10
)
head(syn)

## End(Not run)
```

---

estimate_sigma_equivalent

Estimate the sigma-equivalent compactness of a dataset

Description

This function calibrates the observed AMD peak of real data against synthetic datasets generated with varying levels of isotropic Gaussian noise (σ). For each candidate σ, synthetic data are generated with the same number of samples, dimensionality, and number of clusters as the real data. The AMD peak of each synthetic dataset is computed, and the *sigma-equivalent* value is defined as the σ whose synthetic AMD peak best matches the real AMD peak (either by interpolation or nearest match).

Usage

```
estimate_sigma_equivalent(
  real_data,
  its,
  nin,
  nsp,
  k_opt = NULL,
  sigmas,
  iter_max = 20,
  make_plot = FALSE,
  return_plot = TRUE,
  quiet = TRUE,
  open_device_each = FALSE,
  device_width = 7,
  device_height = 5,
  cube_size = 100,
  method = c("interpolate", "nearest"),
  standardize = FALSE,
  seed_base = 7,
  plot_sigma_curves = FALSE
)
```

Arguments

`real_data`	A numeric matrix or data frame of samples × features.
`its`	Number of random initialisations per AMD computation.
`nin`	Minimum number of clusters to evaluate.
`nsp`	Maximum number of clusters to evaluate.
`k_opt`	Optional; the optimal number of clusters for the real data. If `NULL`, it is estimated internally.
`sigmas`	Numeric vector of candidate σ values to evaluate.
`iter_max`	Maximum number of iterations for fuzzy c-means.
`make_plot`	Logical; if `TRUE`, produce a comparative plot of σ vs synthetic AMD peaks.
`return_plot`	Logical; if `TRUE`, return the comparative plot object.
`quiet`	Logical; suppress console output from synthetic data generation.
`open_device_each`	Logical; if `TRUE`, open a new graphics device for each sigma-curve plot (when `plot_sigma_curves = TRUE`).
`device_width, device_height`	Size of graphics device for sigma-curve plots.
`cube_size`	Side length of the hypercube used to place synthetic centroids.
`method`	Method for estimating sigma-equivalent: `"interpolate"` or `"nearest"`.
`standardize`	Logical; if `TRUE`, standardise synthetic data.
`seed_base`	Base seed for reproducibility.
`plot_sigma_curves`	Logical; if `TRUE`, plot the AMD curve for each candidate σ.

Value

A list containing:

amd_real_peak AMD peak of the real dataset.

k_opt Optimal number of clusters for the real data.

table_sigma_amd Data frame of σ vs synthetic AMD peaks.

sigma_equivalent Interpolated sigma-equivalent value.

sigma_eq Nearest-match sigma on the explored grid.

extrapolated Logical; whether interpolation required extrapolation.

plot_comparative Comparative plot object (if requested).

best_i Index of best-matching sigma.

best_sigma Best-matching sigma value.

best_res_syn Full AMD results for the best synthetic dataset.

best_df_curve Data frame of the AMD curve for the best sigma.

Examples

```
## Not run:
set.seed(1)
X <- matrix(rnorm(1000), nrow = 100, ncol = 10)
out <- estimate_sigma_equivalent(
  real_data = X,
  its = 5,
  nin = 2,
  nsp = 6,
  sigmas = seq(1, 10, by = 2)
)
out$sigma_equivalent

## End(Not run)
```

Index