

Package ‘topolow’

September 14, 2026

Title Force-Directed Euclidean Embedding of Dissimilarity Data

Version 2.1.0

Maintainer Omid Arhami <omid.arhami@uga.edu>

Description A robust implementation of Topolow algorithm. It embeds objects into a low-dimensional Euclidean space from a matrix of pairwise dissimilarities, even when the data do not satisfy metric or Euclidean axioms. The package is particularly well-suited for sparse, incomplete, and censored (thresholded) datasets such as antigenic relationships. The core is a physics-inspired, gradient-free optimization framework that models objects as particles in a physical system, where observed dissimilarities define spring rest lengths and unobserved pairs exert repulsive forces. The package also provides functions specific to antigenic mapping to transform cross-reactivity and binding affinity measurements into accurate spatial representations in a phenotype space.

Key features include:

- * Robust Embedding from Sparse Data: Effectively creates complete and consistent maps (in optimal dimensions) even with high proportions of missing data (e.g., >95%).

- * Physics-Inspired Optimization: Models objects (e.g., antigens, landmarks) as particles connected by springs (for measured dissimilarities) and subject to repulsive forces (for missing dissimilarities), and simulates the physical system using laws of mechanics, reducing the need for complex gradient computations.

- * Automatic Dimensionality Detection: Employs a likelihood-based approach to determine the optimal number of dimensions for the embedding/map, avoiding distortions common in methods with fixed low dimensions.

- * Noise and Bias Reduction: Naturally mitigates experimental noise and bias through its network-based, error-dampening mechanism.

- * Antigenic Velocity Calculation (for antigenic data): Introduces and quantifies “antigenic velocity,” a vector that describes the rate and direction of antigenic drift for each pathogen isolate. This can help identify cluster transitions and potential lineage replacements.

- * Broad Applicability: Analyzes data from various objects that their dissimilarity may be of interest, ranging from complex biological measurements such as continuous and relational phenotypes, antibody-antigen interactions, and protein folding to abstract concepts, such as customer perception of different brands.

Methods are described in the context of bioinformatics applications in Arhami and Rohani (2025a) <[doi:10.1093/bioinformatics/btaf372](https://doi.org/10.1093/bioinformatics/btaf372)>, and mathematical proofs and Euclidean embedding details are in Arhami and Rohani (2025b) <[doi:10.48550/arXiv.2508.01733](https://doi.org/10.48550/arXiv.2508.01733)>.

License BSD_3_clause + file LICENSE

Encoding UTF-8

Imports future, lifecycle, ggplot2 (>= 3.4.0), dplyr (>= 1.1.0),
 data.table (>= 1.14.0), reshape2 (>= 1.4.4), grDevices,
 graphics, grid, stats, tools, utils, parallel (>= 4.1.0),
 filelock, lhs, rlang, gridExtra, igraph, zoo, Rcpp (>= 1.0.0)

LinkingTo Rcpp

Suggests coda (>= 0.19-4), Rtsne, ape, MASS, RANN, smacof, Racmacs (>= 1.1.2),
 vegan, umap, rgl (>= 1.0.0), scales, ggrepel, plotly (>= 4.10.0), viridisLite,
 covr, knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

URL <https://github.com/omid-arhami/topolow>

BugReports <https://github.com/omid-arhami/topolow/issues>

LazyData true

Depends R (>= 4.1.0)

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Omid Arhami [aut, cre, cph] (ORCID:
<https://orcid.org/0009-0005-2681-6598>)

Repository CRAN

Date/Publication 2026-09-14 04:50:02 UTC

Contents

analyze_network_structure	4
calculate_diagnostics	5
calculate_prediction_interval	6
calculate_weighted_marginals	7
check_gaussian_convergence	8
check_matrix_connectivity	9
clean_data	10
color_palettes	11
coordinates_to_matrix	11
create_cv_folds	11
create_diagnostic_report	13
denv_data	14
error_calculator_comparison	15
euclidean_embedding	16
Euclidify	20
example_positions	23
extract_numeric_values	24
ggsave_white_bg	25

h3n2_data	25
hiv_titers	26
hiv_viruses	26
initial_parameter_optimization	27
list_to_matrix	30
log_transform_parameters	32
make_interactive	33
new_aesthetic_config	35
new_annotation_config	36
new_dim_reduction_config	38
new_layout_config	39
parameter_sensitivity_analysis	40
plot.parameter_sensitivity	42
plot.profile_likelihood	43
plot.topolow_convergence	44
plot.topolow_diagnostics	45
plot_3d_mapping	46
plot_cluster_mapping	48
plot_euclidify_diagnostics	51
plot_ll_improvement	53
plot_mcmc_diagnostics	53
plot_network_structure	55
plot_performance_trace	56
plot_temporal_mapping	58
print.parameter_sensitivity	61
print.profile_likelihood	61
print.topolow	62
print.topolow_convergence	63
print.topolow_diagnostics	63
process_antigenic_data	64
profile_likelihood	66
prune_sparse_matrix	68
run_adaptive_sampling	70
sanity_check_subsample	72
save_plot	74
scatterplot_fitted_vs_true	75
subsample_dissimilarity_matrix	77
summary.topolow	79
symmetric_to_nonsymmetric_matrix	79
table_to_matrix	80
titers_list_to_matrix	81
weighted_kde	83

`analyze_network_structure`*Analyze Network Structure*

Description

Analyzes the connectivity of a dissimilarity matrix, returning node degrees and overall completeness.

Usage

```
analyze_network_structure(dissimilarity_matrix)
```

Arguments

`dissimilarity_matrix`
Square symmetric matrix of dissimilarities.

Value

A list containing the network analysis results:

<code>adjacency</code>	A logical matrix where TRUE indicates a measured dissimilarity.
<code>connectivity</code>	A data.frame with node-level metrics, including the completeness (degree) for each point.
<code>summary</code>	A list of overall network statistics, including <code>n_points</code> , <code>n_measurements</code> , and total completeness.

Examples

```
# Create a sample dissimilarity matrix
dist_mat <- matrix(runif(25), 5, 5)
rownames(dist_mat) <- colnames(dist_mat) <- paste0("Point", 1:5)
dist_mat[lower.tri(dist_mat)] <- t(dist_mat)[lower.tri(dist_mat)]
diag(dist_mat) <- 0
dist_mat[1, 3] <- NA; dist_mat[3, 1] <- NA

# Analyze the network structure
metrics <- analyze_network_structure(dist_mat)
print(metrics$summary$completeness)
```

calculate_diagnostics *Calculate MCMC-style Diagnostics for Sampling Chains*

Description

Calculates standard MCMC-style convergence diagnostics for multiple chains from an optimization or sampling run. It computes the R-hat (potential scale reduction factor) and effective sample size (ESS) to help assess if the chains have converged to a stable distribution.

Usage

```
calculate_diagnostics(chain_files, mutual_size = 500)
```

Arguments

chain_files	Character vector. Paths to CSV files, where each file represents a chain of samples.
mutual_size	Integer. Number of samples to use from the end of each chain for calculations.

Value

A list object of class `topolow_diagnostics` containing convergence diagnostics for the MCMC chains.

rhat	A numeric vector of the R-hat (potential scale reduction factor) statistic for each parameter. Values close to 1 indicate convergence.
ess	A numeric vector of the effective sample size for each parameter.
chains	A list of data frames, where each data frame is a cleaned and trimmed MCMC chain.
param_names	A character vector of the parameter names being analyzed.
mutual_size	The integer number of samples used from the end of each chain for calculations.

Examples

```
# This example demonstrates how to use the function with temporary files.
# Create dummy chain files in a temporary directory
temp_dir <- tempdir()
chain_files <- character(3)
par_names <- c("log_N", "log_k0", "log_cooling_rate", "log_c_repulsion")
sample_data <- data.frame(
  log_N = rnorm(100), log_k0 = rnorm(100),
  log_cooling_rate = rnorm(100), log_c_repulsion = rnorm(100),
  NLL = runif(100), Holdout_MAE = runif(100)
)
for (i in 1:3) {
  chain_files[i] <- file.path(temp_dir, paste0("chain", i, ".csv"))
  write.csv(sample_data, chain_files[i], row.names = FALSE)
```

```
}

# Calculate diagnostics
diag_results <- calculate_diagnostics(chain_files, mutual_size = 50)
print(diag_results)

# Clean up the temporary files and directory
unlink(chain_files)
unlink(temp_dir, recursive = TRUE)
```

`calculate_prediction_interval`*Calculate Prediction Interval for Dissimilarity Estimates*

Description

Computes prediction intervals for the estimated dissimilarities based on residual variation between true and predicted values.

Usage

```
calculate_prediction_interval(
  dissimilarity_matrix,
  predicted_dissimilarity_matrix,
  confidence_level = 0.95
)
```

Arguments

`dissimilarity_matrix`
Matrix of true dissimilarities.

`predicted_dissimilarity_matrix`
Matrix of predicted dissimilarities.

`confidence_level`
The confidence level for the interval (default: 0.95).

Value

A single numeric value representing the margin of error for the prediction interval.

`calculate_weighted_marginals`*Calculate Weighted Marginal Distributions*

Description

Calculates the marginal probability distribution for each model parameter. The distributions are weighted by the likelihood of each sample, making this useful for identifying the most probable parameter values from a set of Monte Carlo samples.

Usage

```
calculate_weighted_marginals(samples)
```

Arguments

<code>samples</code>	A data frame containing parameter samples (e.g., <code>log_N</code> , <code>log_k0</code>) and a holdout MAE column named <code>Holdout_MAE</code> .
----------------------	---

Details

This function uses the `weighted_kde` helper to perform kernel density estimation for each parameter, with weights derived from $-\log(\text{MAE})$. Lower MAE (better fit) gives higher weight. Using $\log(\text{MAE})$ instead of NLL ensures weights are comparable across evaluations with different numbers of validation points (e.g., different subsamples or CV configurations). When the number of validation points is constant, this produces identical weights to the previous NLL-based scheme. Using $-\log(\text{MAE})$ produces identical weights to the previous -NLL scheme when the number of validation points (n) is constant across samples, because $\text{NLL} = n \cdot (1 + \log(2 \cdot \text{MAE}))$ and the n factor cancels during normalization. Unlike NLL, $-\log(\text{MAE})$ is invariant to n , making weights comparable across different subsamples and CV configurations.

Value

A named list where each element is a density object (a list with `x` and `y` components) corresponding to a model parameter.

<code>x</code>	Vector of parameter values
<code>y</code>	Vector of density estimates

`check_gaussian_convergence`*Check Multivariate Gaussian Convergence*

Description

Assesses the convergence of multivariate samples by monitoring the stability of the mean vector and covariance matrix over a sliding window. This is useful for checking if a set of parameter samples has stabilized.

Usage

```
check_gaussian_convergence(data, window_size = 300, tolerance = 0.01)
```

Arguments

<code>data</code>	Matrix or Data Frame. A matrix of samples where columns are parameters.
<code>window_size</code>	Integer. The size of the sliding window used to compute statistics.
<code>tolerance</code>	Numeric. The convergence threshold for the relative change in the mean and covariance.

Value

An object of class `topolow_convergence` containing diagnostics about the convergence of the multivariate samples. This list includes logical flags for convergence (`converged`, `mean_converged`, `cov_converged`) and the history of the mean and covariance changes.

Examples

```
# Create sample data for the example
chain_data <- as.data.frame(matrix(rnorm(500 * 4), ncol = 4))
colnames(chain_data) <- c("param1", "param2", "param3", "param4")

# Run the convergence check
conv_results <- check_gaussian_convergence(chain_data)
print(conv_results)

# The plot method for this object can be used to create convergence plots.
# plot(conv_results)
```

`check_matrix_connectivity`*Check Dissimilarity Matrix Connectivity*

Description

Checks whether a dissimilarity matrix forms a connected graph, meaning all points can be reached from any other point through a path of observed dissimilarities. This is critical for ensuring that subsampled data will allow proper embedding optimization.

Usage

```
check_matrix_connectivity(dissimilarity_matrix, min_completeness = 0.1)
```

Arguments

`dissimilarity_matrix`

Square symmetric matrix of dissimilarities. Can contain NA values for missing measurements.

`min_completeness`

Numeric. Minimum network completeness (fraction of possible edges that are observed) to consider acceptable. Default: 0.10. This is used as a warning threshold, not a hard requirement.

Details

A connected graph means there are no isolated groups of points. If the graph has multiple components (islands), optimization will fail because points in different islands have no observed relationships to constrain their relative positions.

The function uses [analyze_network_structure](#) to build an adjacency matrix, then uses igraph to identify connected components.

Value

A list containing connectivity diagnostics:

<code>is_connected</code>	Logical. TRUE if the graph forms a single connected component.
<code>n_components</code>	Integer. Number of separate connected components (islands).
<code>completeness</code>	Numeric. Fraction of possible edges that are observed (0-1).
<code>n_points</code>	Integer. Number of points in the matrix.
<code>n_measurements</code>	Integer. Number of observed dissimilarities.

Examples

```
# Create a connected matrix
connected_mat <- matrix(c(0, 1, 2, 1, 0, 1.5, 2, 1.5, 0), nrow = 3)
result <- check_matrix_connectivity(connected_mat)
print(result$is_connected) # TRUE

# Create a disconnected matrix (two islands)
disconnected_mat <- matrix(NA, nrow = 4, ncol = 4)
diag(disconnected_mat) <- 0
disconnected_mat[1, 2] <- disconnected_mat[2, 1] <- 1
disconnected_mat[3, 4] <- disconnected_mat[4, 3] <- 1
result <- check_matrix_connectivity(disconnected_mat)
print(result$is_connected) # FALSE
print(result$n_components) # 2
```

clean_data

Clean Data by Removing MAD-based Outliers

Description

Removes outliers from numeric data using the Median Absolute Deviation (MAD) method. Outliers are replaced with NA values.

Usage

```
clean_data(x, k = 3, take_log = FALSE)
```

Arguments

x	Numeric vector to clean.
k	Numeric threshold for outlier detection (default: 3).
take_log	Logical. Deprecated parameter. Log transformation should be done before calling this function.

Value

A numeric vector of the same length as x, where detected outliers have been replaced with NA.

See Also

detect_outliers_mad for the underlying outlier detection.

Examples

```
# Clean parameter values
params <- c(0.01, 0.012, 0.011, 0.1, 0.009, 0.011, 0.15)
clean_params <- clean_data(params)
```

color_palettes	<i>Color Palettes</i>
----------------	-----------------------

Description

Predefined color palettes optimized for visualization.

Usage

```
c25
```

coordinates_to_matrix	<i>Convert Coordinates to a Distance Matrix</i>
-----------------------	---

Description

Calculates pairwise Euclidean distances between points in a coordinate space.

Usage

```
coordinates_to_matrix(positions)
```

Arguments

positions	Matrix or Data Frame of coordinates where rows are points and columns are dimensions.
-----------	---

Value

A symmetric matrix of pairwise Euclidean distances between points.

create_cv_folds	<i>Create Cross-Validation Folds for a Dissimilarity Matrix</i>
-----------------	---

Description

Creates k-fold cross-validation splits from a dissimilarity matrix while maintaining symmetry. Each fold in the output consists of a training matrix (with some values masked as NA) and a corresponding ground truth matrix for validation.

Usage

```
create_cv_folds(  
  dissimilarity_matrix,  
  ground_truth_matrix = NULL,  
  n_folds = 10,  
  random_seed = NULL  
)
```

Arguments

dissimilarity_matrix	The input dissimilarity matrix, which may contain noise.
ground_truth_matrix	An optional, noise-free dissimilarity matrix to be used as the ground truth for evaluation. If NULL, the input dissimilarity_matrix is used as the truth.
n_folds	The integer number of folds to create.
random_seed	An optional integer to set the random seed for reproducibility.

Value

A list of length `n_folds`. Each element of the list is itself a list containing two matrices: `truth` (the ground truth for that fold) and `train` (the training matrix with NA values for validation).

Note

This function has breaking changes from previous versions:

- Parameter `truth_matrix` renamed to `dissimilarity_matrix`
- Parameter `no_noise_truth` renamed to `ground_truth_matrix`
- Return structure now uses named elements (`$truth`, `$train`)

Examples

```
# Create a sample dissimilarity matrix  
d_mat <- matrix(runif(100), 10, 10)  
diag(d_mat) <- 0  
  
# Create 5-fold cross-validation splits  
folds <- create_cv_folds(d_mat, n_folds = 5, random_seed = 123)
```

`create_diagnostic_report`*Create Summary Diagnostic Report*

Description

Generates a text summary of the Euclidify optimization process.

Usage

```
create_diagnostic_report(euclidify_result, output_file = NULL)
```

Arguments

`euclidify_result`

A result object from Euclidify with diagnostics.

`output_file`

Character. Optional path to save report as text file.

Value

Character vector with report lines (invisibly).

Examples

```
test_data <- data.frame(
  object = rep(paste0("Obj", 1:4), each = 4),
  reference = rep(paste0("Ref", 1:4), 4),
  score = sample(c(1, 2, 4, 8, 16, 32, 64, "<1", ">12"), 16, replace = TRUE)
)
dist_mat <- list_to_matrix(
  data = test_data, object_col = "object", reference_col = "reference",
  value_col = "score", is_similarity = TRUE
)

result <- Euclidify(
  dissimilarity_matrix = dist_mat,
  output_dir = tempdir(check = TRUE),
  n_initial_samples = 5,
  n_adaptive_samples = 3,
  folds = 3,
  mapping_max_iter = 50,
  ndim_range = c(2, 4),
  max_cores = 1,
  verbose = "off",
  create_diagnostic_plots = TRUE,
  diagnostic_plot_types = "parameter_search"
)

# output_file is optional; supply a path only if you want the report saved.
```

```
report <- create_diagnostic_report(
  result,
  output_file = file.path(tempdir(check = TRUE), "report.txt")
)
cat(report, sep = "\n")
```

denv_data

Dengue Virus (DENV) Titer Data

Description

A dataset containing neutralization titer data for Dengue virus. This data can be used to create antigenic maps and explore the antigenic relationships between different DENV strains.

Usage

```
denv_data
```

Format

A data frame with the following columns:

virus_strain Character, the name of the virus strain.

serum_strain Character, the name of the serum strain.

titer Character, the neutralization titer value. May include values like '<10' or '>1280'.

virusYear Numeric, the year the virus was isolated.

serumYear Numeric, the year the serum was collected.

cluster Factor, the cluster or serotype assignment for the strains.

color Character, a color associated with the cluster for plotting.

Source

Katzelnick, L.C., et al. (2019). An antigenically diverse, representative panel of dengue viruses for neutralizing antibody discovery and vaccine evaluation. *eLife*. doi:10.7554/eLife.42496

`error_calculator_comparison`*Error calculation and validation metrics for topolow Calculate Comprehensive Error Metrics*

Description

Computes a comprehensive set of error metrics (in-sample, out-of-sample, completeness) between predicted and true dissimilarities for model evaluation.

Usage

```
error_calculator_comparison(  
  predicted_dissimilarities,  
  true_dissimilarities,  
  input_dissimilarities = NULL  
)
```

Arguments

`predicted_dissimilarities`
Matrix of predicted dissimilarities from the model.

`true_dissimilarities`
Matrix of true, ground-truth dissimilarities.

`input_dissimilarities`
Matrix of input dissimilarities, which may contain NAs and is used to identify the pattern of missing values for out-of-sample error calculation. Optional - if not provided, defaults to `true_dissimilarities` (no holdout set).

Details

Input requirements and constraints:

- All input matrices must have matching dimensions.
- Row and column names must be consistent across matrices.
- NAs are allowed and handled appropriately.
- Threshold indicators (< or >) in the input matrix are processed correctly.

When `input_dissimilarities` is provided, it represents the training data where some values have been set to NA to create a holdout set. This allows calculation of:

- In-sample errors: for data available during training
- Out-of-sample errors: for data held out during training

When `input_dissimilarities` is NULL (default), all errors are treated as in-sample since no data was held out.

Value

A list containing:

report_df	A data.frame with detailed error metrics for each point-pair, including InSampleError, OutSampleError, and their percentage-based counterparts.
Completeness	A single numeric value representing the completeness statistic, which is the fraction of validation points for which a prediction could be made.

Examples

```
# Example 1: Normal evaluation (no cross-validation)
true_mat <- matrix(c(0, 1, 2, 1, 0, 3, 2, 3, 0), 3, 3)
pred_mat <- true_mat + rnorm(9, 0, 0.1) # Add some noise

# Evaluate all predictions (input_dissimilarities defaults to true_dissimilarities)
errors1 <- error_calculator_comparison(pred_mat, true_mat)

# Example 2: Cross-validation evaluation
input_mat <- true_mat
input_mat[1, 3] <- input_mat[3, 1] <- NA # Create holdout set

# Evaluate with train/test split
errors2 <- error_calculator_comparison(pred_mat, true_mat, input_mat)
```

euclidean_embedding *Main topolow algorithm implementation*

Description**[Stable]**

topolow (topological stochastic pairwise reconstruction for Euclidean embedding) optimizes point positions in an N-dimensional space to match a target dissimilarity matrix. This version uses an **Exact** C++ backend that:

- Iterates over all $N(N-1)/2$ pairs in every iteration ($O(N^2)$ complexity).
- Ensures perfect fidelity to the original algorithm's force physics.
- Uses Gauss-Seidel immediate updates for fast convergence.
- Stochastic pair shuffling for escaping local optima.
- Compressed edge list (COO format) for efficient error calculation.

Usage

```
euclidean_embedding(
  dissimilarity_matrix,
  ndim,
  mapping_max_iter = 1000,
  k0,
  cooling_rate,
  c_repulsion,
  relative_epsilon = 1e-04,
  convergence_counter = 5,
  initial_positions = NULL,
  write_positions_to_csv = FALSE,
  output_dir,
  verbose = FALSE,
  convergence_check_freq = 3,
  preserve_order = FALSE
)
```

Arguments

dissimilarity_matrix	Matrix. A square, symmetric dissimilarity matrix. Can contain NA values for missing measurements and character strings with < or > prefixes for thresholded measurements.
ndim	Integer. Number of dimensions for the embedding space.
mapping_max_iter	Integer. Maximum number of map optimization iterations.
k0	Numeric. Initial spring constant controlling spring forces.
cooling_rate	Numeric. Rate of spring constant decay per iteration ($0 < \text{cooling_rate} < 1$).
c_repulsion	Numeric. Repulsion constant controlling repulsive forces.
relative_epsilon	Numeric. Convergence threshold for relative change in error. Default is $1e-4$.
convergence_counter	Integer. Number of consecutive iterations below threshold before declaring convergence. Default is 5.
initial_positions	Matrix or NULL. Optional starting coordinates. If NULL, random initialization is used. Matrix should have $\text{nrow} = \text{nrow}(\text{dissimilarity_matrix})$ and $\text{ncol} = \text{ndim}$.
write_positions_to_csv	Logical. Whether to save point positions to a CSV file. Default is FALSE.
output_dir	Character. Directory to save the CSV file. Required if <code>write_positions_to_csv</code> is TRUE.
verbose	Logical. Whether to print progress messages. Default is FALSE.
convergence_check_freq	Integer. How often to check for convergence (every N iterations). Lower values give more precise stopping but add overhead. Default is 3.

`preserve_order` Logical. If TRUE, the original row and column order of the dissimilarity matrix is strictly preserved. If FALSE (default), the matrix is reordered internally for optimized convergence (spectral pattern with largest values in corners). Set to TRUE when domain knowledge supports a specific order (e.g., directional evolution) or when downstream analyses depend on specific point ordering matching the input matrix.

Details

The algorithm iteratively updates point positions using:

- Spring forces between points with measured dissimilarities.
- Repulsive forces between ALL points without measurements (or thresholded pairs).
- Conditional forces for thresholded measurements (< or >).
- An adaptive spring constant that decays over iterations.
- Convergence monitoring based on relative error change.
- Automatic matrix reordering to optimize convergence. Consider if downstream analyses depend on specific point ordering: The order of points in the output is adjusted to put high-dissimilarity points in the opposing ends.

This function replaces the deprecated `create_topolow_map()`. The core algorithm is identical, but includes performance improvements and enhanced validation.

Value

A list object of class `topolow`. This list contains the results of the optimization and includes the following components:

- `positions`: A matrix of the optimized point coordinates in the N-dimensional space.
- `est_distances`: A matrix of the Euclidean distances between points in the final optimized configuration.
- `mae`: The final Mean Absolute Error between the target dissimilarities and the estimated distances.
- `iter`: The total number of iterations performed before the algorithm terminated.
- `parameters`: A list containing the input parameters used for the optimization run.
- `convergence`: A list containing the final convergence status, including a logical achieved flag and the final error value.

Reproducibility

Two independent sources of randomness act on a run, and `set.seed()` governs only the first:

- **Starting configuration (seeded)**. Initial point positions are drawn with `stats::runif()` in R, so `set.seed()` before the call fixes them.
- **Sweep order (not seeded)**. The C++ backend re-permutes the order in which the $N(N-1)/2$ pairs are visited on every iteration, using a Mersenne Twister seeded from `std::random_device`. This generator is independent of R's RNG stream and is therefore not affected by `set.seed()`.

The shuffle is deliberate and confined to ordering. It never changes which pairs are considered, their dissimilarities or weights, the spring and repulsion physics, the cooling schedule, or the convergence test – all of which are deterministic. Because positions are updated in place (Gauss-Seidel), a fixed sweep order would systematically favour points early in that order, which are always moved against stale neighbour positions. Reshuffling each iteration removes that bias and supplies the perturbation that lets the configuration escape local optima.

Consequently, repeated runs on the same input give similar but not identical coordinates, and results are not bit-reproducible across runs or platforms. This is expected. When a single canonical map is needed, run the embedding several times and compare the runs with the package diagnostics – for example `error_calculator_comparison()` and `scatterplot_fitted_vs_true()` – then retain the run with the lowest holdout error. Agreement across runs is itself the evidence that the embedding is well determined by the data.

See Also

`create_topolow_map()` for the deprecated predecessor function.

Examples

```
# Create a simple dissimilarity matrix
dist_mat <- matrix(c(0, 2, 3, 2, 0, 4, 3, 4, 0), nrow=3)

# Run topolow in 2D
result <- euclidean_embedding(
  dissimilarity_matrix = dist_mat,
  ndim = 2,
  mapping_max_iter = 100,
  k0 = 1.0,
  cooling_rate = 0.001,
  c_repulsion = 0.01,
  verbose = FALSE
)

# View results
head(result$positions)
print(result$mae)

# Example with thresholded measurements
thresh_mat <- matrix(c(0, ">2", 3, ">2", 0, "<5", 3, "<5", 0), nrow=3)
result_thresh <- euclidean_embedding(
  dissimilarity_matrix = thresh_mat,
  ndim = 2,
  mapping_max_iter = 50,
  k0 = 0.5,
  cooling_rate = 0.01,
  c_repulsion = 0.001
)
```

Description

A user-friendly wrapper function that automatically optimizes parameters and performs Euclidean embedding on a dissimilarity matrix. This function handles the entire workflow from parameter optimization to final embedding, with comprehensive diagnostic tracking and visualization.

Usage

```
Euclidify(
  dissimilarity_matrix,
  output_dir,
  ndim_range = c(2, 10),
  k0_range = c(0.1, 20),
  cooling_rate_range = c(1e-04, 0.1),
  c_repulsion_range = c(1e-04, 1),
  n_initial_samples = 50,
  n_adaptive_samples = 150,
  max_cores = NULL,
  folds = 20,
  mapping_max_iter = 500,
  opt_subsample = NULL,
  clean_intermediate = TRUE,
  verbose = "standard",
  fallback_to_defaults = FALSE,
  save_results = FALSE,
  create_diagnostic_plots = FALSE,
  diagnostic_plot_types = "all",
  preserve_order = FALSE
)
```

Arguments

dissimilarity_matrix	Square symmetric dissimilarity matrix. Can contain NA values for missing measurements and threshold indicators (< or >).
output_dir	Character. Directory for saving optimization files and results. Required - no default.
ndim_range	Integer vector of length 2. Starting range for the number of dimensions (minimum, maximum). Default: c(2, 10). This seeds the search rather than bounding it – see the note on parameter ranges below.
k0_range	Numeric vector of length 2. Starting range for the initial spring constant (minimum, maximum). Default: c(0.1, 15)

cooling_rate_range	Numeric vector of length 2. Starting range for the cooling rate (minimum, maximum). Default: c(0.001, 0.07)
c_repulsion_range	Numeric vector of length 2. Starting range for the repulsion constant (minimum, maximum). Default: c(0.001, 0.4)
n_initial_samples	Integer. Number of samples for initial parameter optimization. Default: 100
n_adaptive_samples	Integer. Number of samples for adaptive refinement. Default: 150
max_cores	Integer. Maximum number of cores to use. Default: NULL (auto-detect)
folds	Integer. Number of cross-validation folds. Default: 20
mapping_max_iter	Integer. Maximum iterations for final embedding. Half this value is used for parameter search. Default: 500
opt_subsample	Integer or NULL. If specified, uses subsampling during parameter optimization (both initial and adaptive) to reduce computational cost. Randomly samples this many points for each parameter evaluation. Final embedding always uses the full dataset. Default: NULL (no subsampling). Recommended for large datasets (>300 points). See initial_parameter_optimization for details.
clean_intermediate	Logical. Whether to remove intermediate files. Default: TRUE
verbose	Character. Verbosity level: "off" (no output), "standard" (progress updates), or "full" (detailed output including from internal functions). Default: "standard"
fallback_to_defaults	Logical. Whether to use default parameters if optimization fails. Default: FALSE
save_results	Logical. Whether to save the final positions as CSV. Default: FALSE
create_diagnostic_plots	Logical. Whether to create diagnostic and trace plots showing the parameter optimization process and embedding quality. Default: FALSE
diagnostic_plot_types	Character vector. Which plot types to create. Options: "all", "parameter_search", "convergence", "quality", "cv_errors". Default: "all"
preserve_order	Logical. If TRUE, the original row and column order of the dissimilarity matrix is preserved in the output coordinates. If FALSE, the data is reordered based on their available distances for a faster convergence. Set to TRUE when domain knowledge supports a specific order (e.g., directional evolution) or when downstream analyses depend on specific point ordering matching the input matrix. Default: FALSE

Value

A list containing:

positions	Matrix of optimized coordinates
est_distances	Matrix of estimated distances

<code>mae</code>	Mean absolute error
<code>optimal_params</code>	List of optimal parameters found, including cross-validation MAE during optimization
<code>optimization_summary</code>	Summary of the optimization process
<code>data_characteristics</code>	Summary of input data characteristics
<code>runtime</code>	Total runtime in seconds
<code>all_samples</code>	Data frame of all parameter evaluations (if <code>create_diagnostic_plots=TRUE</code>)
<code>diagnostic_plots</code>	List of ggplot objects (if <code>create_diagnostic_plots=TRUE</code>)
<code>dissimilarity_matrix</code>	Input dissimilarity matrix (if <code>create_diagnostic_plots=TRUE</code>)

Parameter ranges are a starting point, not a constraint

The four `*_range` arguments define where the search begins. They are not hard bounds: initial sampling draws inside them, but adaptive refinement samples from a kernel density estimate fitted to the best parameter sets found so far, and that estimate can propose values outside the original range. Refinement therefore may return an `ndim`, `k0`, `cooling_rate` or `c_repulsion` beyond the range you supplied, when the data support it.

This is deliberate – it lets the search escape a range that was set too narrowly – but it means `ndim_range = c(3, 6)` does not guarantee a 3- to 6-dimensional result. Only the physical sanity limits are enforced (for example $1 \leq \text{ndim} \leq 50$). Check `result$optimal_params` for the values actually used, and widen or reconsider the range if refinement consistently drifts away from it. If a hard cap is essential, call `euclidean_embedding()` directly with an explicit `ndim`.

Reproducibility

`Euclidify()` calls `euclidean_embedding()`, whose sweep order is randomized by a generator independent of R's RNG. Repeated runs therefore give similar but not identical coordinates even under `set.seed()`. See the Reproducibility section of `euclidean_embedding()` for what is and is not seeded, and for how to obtain a canonical map.

Examples

```
# Build a small dissimilarity matrix from a long-format table. The scores
# include censored values ("<1", ">12"), which are carried through as
# thresholds rather than being discarded.
test_data <- data.frame(
  object = rep(paste0("Obj", 1:4), each = 4),
  reference = rep(paste0("Ref", 1:4), 4),
  score = sample(c(1, 2, 4, 8, 16, 32, 64, "<1", ">12"), 16, replace = TRUE)
)
dist_mat <- list_to_matrix(
  data = test_data,
  object_col = "object",
  reference_col = "reference",
```

```

    value_col = "score",
    is_similarity = TRUE
  )

  # Missing measurements need no imputation; leave them as NA.
  dist_mat[1, 3] <- dist_mat[3, 1] <- NA

  # `output_dir` is required. Examples must not write outside tempdir().
  # `max_cores = 1` keeps this example fast; omit it in real use and
  # Euclidify() will choose a sensible number of cores itself.
  result <- Euclidify(
    dissimilarity_matrix = dist_mat,
    output_dir = tempdir(check = TRUE),
    n_initial_samples = 10,
    n_adaptive_samples = 7,
    max_cores = 1,
    verbose = "off"
  )

  # Embedded coordinates, one row per object
  head(result$positions)

  # Dimensionality and parameters chosen by the search
  result$optimal_params

  # Fit of the embedding to the input dissimilarities
  result$mae

```

example_positions

Example Antigenic Mapping Data

Description

HI titers of Influenza antigens and antisera published in Smith et al., 2004 were used to find the antigenic relationships and coordinates of the antigens. It can be used for mapping. The data captures how different influenza virus strains (antigens) react with antisera from infected individuals.

Usage

```
example_positions
```

Format

A data frame with 285 rows and 11 variables:

V1 First dimension coordinate from 5D mapping

V2 Second dimension coordinate from 5D mapping
V3 Third dimension coordinate from 5D mapping
V4 Fourth dimension coordinate from 5D mapping
V5 Fifth dimension coordinate from 5D mapping
name Strain identifier
antigen Logical; TRUE if point represents an antigen
antiserum Logical; TRUE if point represents an antiserum
cluster Factor indicating antigenic cluster assignment (A/H3N2 1968-2003)
color Color assignment for visualization
year Year of strain isolation

Source

Smith et al., 2004

extract_numeric_values

Utility functions for the topolow package Extract Numeric Values from Mixed Data

Description

Extracts numeric values from data that may contain threshold indicators (e.g., "<10", ">1280") or regular numeric values.

Usage

```
extract_numeric_values(x)
```

Arguments

x A vector that may contain numeric values, character strings with threshold indicators, or a mix of both.

Value

A numeric vector with threshold indicators converted to their numeric equivalents.

Examples

```
# Mixed data with threshold indicators
mixed_data <- c(10, 20, "<5", ">100", 50)
extract_numeric_values(mixed_data)
```

ggsave_white_bg	<i>Save ggplot with white background</i>
-----------------	--

Description

Wrapper around `ggplot2::ggsave` that ensures a white background by default.

Usage

```
ggsave_white_bg(..., bg = "white")
```

Arguments

...	Other arguments passed on to the graphics device function, as specified by device.
bg	Background colour. If NULL, uses the <code>plot.background</code> fill value from the plot theme.

Value

No return value, called for side effects.

h3n2_data	<i>H3N2 Influenza HI Assay Data from Smith et al. 2004</i>
-----------	--

Description

Hemagglutination inhibition (HI) assay data for influenza A/H3N2 viruses spanning 35 years of evolution.

Usage

```
h3n2_data
```

Format

A data frame with the following variables:

virusStrain Character. Virus strain identifier
serumStrain Character. Antiserum strain identifier
titer Numeric. HI assay titer value
virusYear Numeric. Year virus was isolated
serumYear Numeric. Year serum was collected
cluster Factor. Antigenic cluster assignment
color Character. Color code for visualization

Source

Smith et al. (2004) Science, 305(5682), 371-376.

hiv_titers	<i>HIV Neutralization Assay Data</i>
------------	--------------------------------------

Description

IC50 neutralization measurements between HIV viruses and antibodies.

Usage

hiv_titers

Format

A data frame with the following variables:

Antibody Character. Antibody identifier

Virus Character. Virus strain identifier

IC50 Numeric. IC50 neutralization value

Source

Los Alamos HIV Database (<https://www.hiv.lanl.gov/>)

hiv_viruses	<i>HIV Virus Metadata</i>
-------------	---------------------------

Description

Reference information for HIV virus strains used in neutralization assays.

Usage

hiv_viruses

Format

A data frame with the following variables:

Virus.name Character. Virus strain identifier

Country Character. Country of origin

Subtype Character. HIV subtype

Year Numeric. Year of isolation

Source

Los Alamos HIV Database (<https://www.hiv.lanl.gov/>)

initial_parameter_optimization

Parameter Space Sampling and Optimization Functions for topolow

Description

Performs parameter optimization using Latin Hypercube Sampling (LHS) combined with k-fold cross-validation. Parameters are sampled from specified ranges using maximin LHS design to ensure good coverage of parameter space. Each parameter set is evaluated using k-fold cross-validation to assess prediction accuracy. To calculate one NLL per set of parameters, the function uses a pooled errors approach which combine all validation errors into one set, then calculate a single NLL. This approach has two main advantages: 1- It treats all validation errors equally, respecting the underlying error distribution assumption 2- It properly accounts for the total number of validation points

Note: As of version 2.0.0, this function returns log-transformed parameters directly, eliminating the need to call `log_transform_parameters()` separately.

Usage

```
initial_parameter_optimization(  
  dissimilarity_matrix,  
  mapping_max_iter = 1000,  
  relative_epsilon = 0.001,  
  convergence_counter = 3,  
  scenario_name,  
  N_min,  
  N_max,  
  k0_min,  
  k0_max,  
  c_repulsion_min,  
  c_repulsion_max,  
  cooling_rate_min,  
  cooling_rate_max,  
  num_samples = 20,  
  epochs = 1,  
  max_cores = NULL,  
  folds = 20,  
  opt_subsample = NULL,  
  verbose = FALSE,  
  write_files = FALSE,  
  output_dir,  
  preserve_order = FALSE  
)
```

Arguments

dissimilarity_matrix	Matrix. Input dissimilarity matrix. Must be square and symmetric.
mapping_max_iter	Integer. Maximum number of optimization iterations for each map.
relative_epsilon	Numeric. Convergence threshold for relative change in error.
convergence_counter	Integer. Number of iterations below threshold before declaring convergence.
scenario_name	Character. Name for output files and job identification.
N_min, N_max	Integer. Range for the number of dimensions parameter.
k0_min, k0_max	Numeric. Range for the initial spring constant parameter.
c_repulsion_min, c_repulsion_max	Numeric. Range for the repulsion constant parameter.
cooling_rate_min, cooling_rate_max	Numeric. Range for the cooling rate parameter.
num_samples	Integer. Number of LHS samples to generate per epoch . Default: 20.
epochs	Integer. Number of optimization epochs. In each epoch, parameters are sampled, evaluated, and the best 50% are used to refine the search space for the next epoch. Default: 3.
max_cores	Integer. Maximum number of cores for parallel processing. Default: NULL (uses all but one).
folds	Integer. Number of cross-validation folds. Default: 20.
opt_subsample	Integer or NULL. If specified, randomly samples this many points from the dissimilarity matrix for each parameter evaluation to reduce computational cost. The function automatically validates that subsampled data forms a connected graph. Each parameter evaluation uses a different random subsample for robustness. Default: NULL (use full data).
Notes:	
• If connectivity cannot be achieved, another random sample is attempted up to max_attempts times (default: 5). • Minimum recommended: <code>opt_subsample >= max(100, folds)</code> • Each parameter set gets a different subsample, but all CV folds within that parameter set use the same subsample • The actual subsample size used is reported in output columns	
verbose	Logical. Whether to print progress messages. Default: FALSE.
write_files	Logical. Whether to save results to a CSV file. Default: FALSE.
output_dir	Character. Directory for output files. Required if write_files is TRUE.
preserve_order	Logical. If TRUE, the original row and column order of the dissimilarity matrix is preserved in the output coordinates. If FALSE, the data is reordered based on their available distances for a faster convergence. Set to TRUE when domain knowledge supports a specific order (e.g., directional evolution) or when downstream analyses depend on specific point ordering matching the input matrix. Default: FALSE

Details

Initial Parameter Optimization using Latin Hypercube Sampling

The function performs these steps in an epoch-based evolutionary strategy:

1. **Initialization:** Starts with the user-provided parameter ranges.
2. **Epoch Loop:** For each epoch:
 - a. Generates `num_samples` using LHS within the current parameter ranges.
 - b. If `opt_subsample` is specified, each evaluation uses a random subsample.
 - c. Evaluates parameter sets via cross-validation (in parallel batches).
 - d. **Range Update** (after all but the final epoch):
 - Sorts results by NLL and keeps the top 50%.
 - Updates parameter ranges for the next epoch based on survivors: $\text{New Min} = 0.75 * \text{Min}(\text{Survivors})$, $\text{New Max} = 1.25 * \text{Max}(\text{Survivors})$.
 - This allows the search to drift and zoom in on optimal regions.
3. **Finalization:** Automatically log-transforms the results from the **final epoch** for direct use with adaptive sampling.

Note on Cross-Validation with Subsampling: When `opt_subsample` is used, each parameter evaluation receives a different random subsample. Since the underlying data differs between evaluations, each evaluation also gets its own CV fold structure (created internally by `likelihood_function`). This approach tests parameter robustness across different data samples and fold structures, which is appropriate for the exploration phase of parameter optimization.

Value

A `data.frame` containing the log-transformed parameter sets and their performance metrics from the **final epoch**. Columns include: `log_N`, `log_k0`, `log_cooling_rate`, `log_c_repulsion`, `Holdout_MAE`, `NLL`, and if `opt_subsample` was used: `opt_subsample`, `original_n_points`.

Note

Breaking Change in v2.0.0: This function now returns log-transformed parameters directly. The returned data frame has columns `log_N`, `log_k0`, `log_cooling_rate`, `log_c_repulsion` instead of the original scale parameters. This eliminates the need to call `log_transform_parameters()` separately before using `run_adaptive_sampling()`.

Breaking Change in v2.0.0: The parameter `distance_matrix` has been renamed to `dissimilarity_matrix`. Please update your code accordingly.

See Also

[euclidean_embedding](#) for the core optimization algorithm, [subsample_dissimilarity_matrix](#) for subsampling details.

Examples

```
# This example can exceed 5 seconds on some systems.
# 1. Create a simple synthetic dataset for the example
synth_coords <- matrix(rnorm(60), nrow = 20, ncol = 3)
dist_mat <- coordinates_to_matrix(synth_coords)
```

```

# 2. Run the optimization on the synthetic data (full data)
results <- initial_parameter_optimization(
  dissimilarity_matrix = dist_mat,
  mapping_max_iter = 100,
  relative_epsilon = 1e-3,
  convergence_counter = 2,
  scenario_name = "test_opt_synthetic",
  N_min = 2, N_max = 5,
  k0_min = 1, k0_max = 10,
  c_repulsion_min = 0.001, c_repulsion_max = 0.05,
  cooling_rate_min = 0.001, cooling_rate_max = 0.02,
  num_samples = 4,
  epochs = 2, # Use 2 epochs
  max_cores = 1,
  verbose = FALSE
)

# 3. With subsampling for faster computation
results_sub <- initial_parameter_optimization(
  dissimilarity_matrix = dist_mat,
  mapping_max_iter = 100,
  relative_epsilon = 1e-3,
  convergence_counter = 2,
  scenario_name = "test_opt_subsampled",
  N_min = 2, N_max = 5,
  k0_min = 1, k0_max = 10,
  c_repulsion_min = 0.001, c_repulsion_max = 0.05,
  cooling_rate_min = 0.001, cooling_rate_max = 0.02,
  num_samples = 4,
  epochs = 1,
  max_cores = 1,
  folds = 10,
  opt_subsample = 15, # Use only 15 points
  verbose = TRUE
)

```

list_to_matrix

topolow Data Preprocessing Functions

Description

Converts data from long/list format (one measurement per row) to a symmetric dissimilarity matrix. The function handles both similarity and dissimilarity data, with optional conversion from similarity to dissimilarity.

Usage

```
list_to_matrix(  
  data,  
  object_col,  
  reference_col,  
  value_col,  
  is_similarity = FALSE  
)
```

Arguments

data	Data frame in long format with columns for objects, references, and values.
object_col	Character. Name of the column containing object identifiers.
reference_col	Character. Name of the column containing reference identifiers.
value_col	Character. Name of the column containing measurement values.
is_similarity	Logical. Whether values are similarities (TRUE) or dissimilarities (FALSE). If TRUE, similarities will be converted to dissimilarities by subtracting from the maximum value per reference. Default: FALSE.

Details**Convert List Format Data to Dissimilarity Matrix**

The function expects data in long format with at least three columns:

- A column for object names
- A column for reference names
- A column containing the (dis)similarity values

When `is_similarity = TRUE`, the function converts similarities to dissimilarities by subtracting each similarity value from the maximum similarity value within each reference group. Threshold indicators (`<` or `>`) are handled appropriately and inverted during similarity-to-dissimilarity conversion.

Value

A symmetric matrix of dissimilarities with row and column names corresponding to the union of unique objects and references in the data. NA values represent unmeasured pairs, and the diagonal is set to 0.

Examples

```
# Example with dissimilarity data  
data_dissim <- data.frame(  
  object = c("A", "B", "A", "C"),  
  reference = c("X", "X", "Y", "Y"),  
  dissimilarity = c(2.5, 1.8, 3.0, 4.2)  
)
```

```

mat_dissim <- list_to_matrix(
  data = data_dissim,
  object_col = "object",
  reference_col = "reference",
  value_col = "dissimilarity",
  is_similarity = FALSE
)

# Example with similarity data (will be converted to dissimilarity)
data_sim <- data.frame(
  object = c("A", "B", "A", "C"),
  reference = c("X", "X", "Y", "Y"),
  similarity = c(7.5, 8.2, 7.0, 5.8)
)

mat_from_sim <- list_to_matrix(
  data = data_sim,
  object_col = "object",
  reference_col = "reference",
  value_col = "similarity",
  is_similarity = TRUE
)

```

log_transform_parameters

Log Transform Parameter Samples

Description

Reads parameter samples from a CSV file and applies a log transformation to specified parameter columns (e.g., N, k0, cooling_rate, c_repulsion).

Note: As of version 2.0.0, this function is primarily for backward compatibility with existing parameter files. The `initial_parameter_optimization()` function now returns log-transformed parameters directly, eliminating the need for this separate transformation step in the normal workflow.

Usage

```
log_transform_parameters(samples_file, output_file = NULL)
```

Arguments

<code>samples_file</code>	Character. Path to the CSV file containing the parameter samples.
<code>output_file</code>	Character. Optional path to save the transformed data as a new CSV file.

Details

This function is maintained for users who have existing parameter files from older versions of the package or who need to work with parameter files that contain original-scale parameters. In the current workflow:

- `initial_parameter_optimization()` → returns log-transformed parameters directly
- `run_adaptive_sampling()` → works with log-transformed parameters
- `euclidean_embedding()` → works with original-scale parameters

If you are working with the current workflow (using `Euclidify()` or calling `initial_parameter_optimization()` directly), you typically do not need to call this function.

Value

A data.frame with the log-transformed parameters. If `output_file` is specified, the data frame is also written to a file and returned invisibly.

Note

Backward Compatibility Note: This function is maintained for compatibility with existing workflows and parameter files. For new workflows, consider using `initial_parameter_optimization()` which returns log-transformed parameters directly.

Examples

```
# This example uses a sample file included with the package.
sample_file <- system.file("extdata", "sample_params.csv", package = "topolow")

# Ensure the file exists before running the example
if (nzchar(sample_file)) {
  # Transform the data from the sample file and return as a data frame
  transformed_data <- log_transform_parameters(sample_file, output_file = NULL)

  # Display the first few rows of the transformed data
  print(head(transformed_data))
}
```

make_interactive

Create Interactive Plot

Description

Converts a static ggplot visualization to an interactive plotly visualization with customizable tooltips and interactive features.

Usage

```
make_interactive(plot, tooltip_vars = NULL)
```

Arguments

`plot` ggplot object to convert

`tooltip_vars` Vector of variable names to include in tooltips

Details

The function enhances static plots by adding:

- Hover tooltips with data values
- Zoom capabilities
- Pan capabilities
- Click interactions
- Double-click to reset

If `tooltip_vars` is `NULL`, the function attempts to automatically determine relevant variables from the plot's mapping.

Value

A plotly object with interactive features.

Examples

```
if (interactive() && requireNamespace("plotly", quietly = TRUE)) {
# Create sample data and plot
data <- data.frame(
  V1 = rnorm(100), V2 = rnorm(100), name=1:100,
  antigen = rep(c(0,1), 50), antiserum = rep(c(1,0), 50),
  year = rep(2000:2009, each=10), cluster = rep(1:5, each=20)
)

# Create temporal plot
p1 <- plot_temporal_mapping(data, ndim=2)

# Make interactive with default tooltips
p1_interactive <- make_interactive(p1)

# Create cluster plot with custom tooltips
p2 <- plot_cluster_mapping(data, ndim=2)
p2_interactive <- make_interactive(p2,
  tooltip_vars = c("cluster", "year", "antigen")
)
}
```

new_aesthetic_config *Plot Aesthetic Configuration Class*

Description

S3 class for configuring plot visual aesthetics including points, colors, labels and text elements.

Usage

```
new_aesthetic_config(  
  point_size = 3.5,  
  point_alpha = 0.8,  
  point_shapes = c(antigen = 16, antiserum = 0),  
  color_palette = c25,  
  gradient_colors = list(low = "blue", high = "red"),  
  show_labels = FALSE,  
  show_title = FALSE,  
  label_size = 3,  
  title_size = 14,  
  subtitle_size = 12,  
  axis_title_size = 12,  
  axis_text_size = 10,  
  legend_text_size = 10,  
  legend_title_size = 12,  
  show_legend = TRUE,  
  legend_position = "right",  
  arrow_head_size = 0.2,  
  arrow_alpha = 0.6,  
  antiserum_color = NULL,  
  antiserum_alpha = NULL  
)
```

Arguments

point_size	Base point size
point_alpha	Point transparency
point_shapes	Named vector of shapes for different point types
color_palette	Color palette name or custom palette
gradient_colors	List with low and high colors for gradients
show_labels	Whether to show point labels
show_title	Whether to show plot title (default: FALSE)
label_size	Label text size
title_size	Title text size

subtitle_size	Subtitle text size
axis_title_size	Axis title text size
axis_text_size	Axis text size
legend_text_size	Legend text size
legend_title_size	Legend title text size
show_legend	Whether to show the legend
legend_position	Legend position ("none", "right", "left", "top", "bottom")
arrow_head_size	Size of the arrow head for velocity arrows (in cm)
arrow_alpha	Transparency of arrows (0 = invisible, 1 = fully opaque)
antiserum_color	Optional character. If supplied, antiserum points are drawn in this single fixed color instead of the cluster color scale, and are kept off the cluster color legend. If NULL (default), antisera share the cluster colors (original behavior).
antiserum_alpha	Optional numeric. Transparency for antiserum points when antiserum_color is set. If NULL, point_alpha is used.

Value

An S3 object of class `aesthetic_config`, which is a list containing the specified configuration parameters for plot aesthetics.

`new_annotation_config` *Visualization functions for the topolow package Plot Annotation Configuration Class*

Description

S3 class for configuring point annotations in plots, including labels, connecting lines, and visual properties.

Usage

```
new_annotation_config(
  notable_points = NULL,
  size = 4.9,
  color = "black",
  alpha = 0.9,
  fontface = "plain",
  box = FALSE,
```

```

    segment_size = 0.3,
    segment_alpha = 0.6,
    min_segment_length = 0,
    max_overlaps = Inf,
    outline_size = 0.4,
    annotate_antiseras = TRUE,
    notable_labels = NULL
  )

```

Arguments

<code>notable_points</code>	Character vector of notable points to highlight
<code>size</code>	Numeric. Size of annotations for notable points
<code>color</code>	Character. Color of annotations for notable points
<code>alpha</code>	Numeric. Alpha transparency of annotations
<code>fontface</code>	Character. Font face of annotations ("plain", "bold", "italic", etc.)
<code>box</code>	Logical. Whether to draw a box around annotations
<code>segment_size</code>	Numeric. Size of segments connecting annotations to points
<code>segment_alpha</code>	Numeric. Alpha transparency of connecting segments
<code>min_segment_length</code>	Numeric. Minimum length of connecting segments
<code>max_overlaps</code>	Numeric. Maximum number of overlaps allowed for annotations
<code>outline_size</code>	Numeric. Size of the outline for annotations
<code>annotate_antiseras</code>	Logical. If TRUE (default), notable antisera are emphasized (filled square) and labeled. If FALSE, notable antisera are drawn as ordinary antisera and are neither emphasized nor labeled, so each notable strain is labeled once (at its antigen point).
<code>notable_labels</code>	Optional named character vector mapping notable-point names (matching <code>notable_points</code> / the cleaned point name) to the text actually drawn, e.g. <code>c("A/DARWIN/9/2021" = "DA/21")</code> . Names not found fall back to the cleaned point name. If NULL (default), the cleaned point name is shown.

Value

An S3 object of class `annotation_config`, which is a list containing the specified configuration parameters for plot annotations.

new_dim_reduction_config

Dimension Reduction Configuration Class

Description

S3 class for configuring dimension reduction parameters including method selection and algorithm-specific parameters.

Usage

```
new_dim_reduction_config(
  method = "pca",
  n_components = 2,
  scale = TRUE,
  center = TRUE,
  pca_params = list(tol = sqrt(.Machine$double.eps), rank. = NULL),
  umap_params = list(n_neighbors = 15, min_dist = 0.1, metric = "euclidean", n_epochs =
    200),
  tsne_params = list(perplexity = 30, mapping_max_iter = 1000, theta = 0.5),
  compute_loadings = FALSE,
  random_state = NULL
)
```

Arguments

method	Dimension reduction method ("pca", "umap", "tsne")
n_components	Number of components to compute
scale	Scale the data before reduction
center	Center the data before reduction
pca_params	List of PCA-specific parameters
umap_params	List of UMAP-specific parameters
tsne_params	List of t-SNE-specific parameters
compute_loadings	Compute and return loadings
random_state	Random seed for reproducibility

Value

An S3 object of class `dim_reduction_config`, which is a list containing the specified configuration parameters for dimensionality reduction.

new_layout_config	<i>Plot Layout Configuration Class</i>
-------------------	--

Description

S3 class for configuring plot layout including dimensions, margins, grids and coordinate systems.

Usage

```
new_layout_config(
  width = 8,
  height = 8,
  dpi = 300,
  aspect_ratio = 1,
  show_grid = TRUE,
  grid_type = "major",
  grid_color = "grey80",
  grid_linetype = "dashed",
  show_axis = TRUE,
  axis_lines = TRUE,
  plot_margin = margin(1, 1, 1, 1, "cm"),
  coord_type = "fixed",
  background_color = "white",
  panel_background_color = "white",
  panel_border = TRUE,
  panel_border_color = "black",
  save_plot = FALSE,
  save_format = "png",
  reverse_x = 1,
  reverse_y = 1,
  swap_axes = FALSE,
  x_limits = NULL,
  y_limits = NULL,
  arrow_plot_threshold = 0.1
)
```

Arguments

width	Plot width in inches
height	Plot height in inches
dpi	Plot resolution
aspect_ratio	Plot aspect ratio
show_grid	Show plot grid
grid_type	Grid type ("none", "major", "minor", "both")
grid_color	Grid color

grid_linetype	Grid line type
show_axis	Show axes
axis_lines	Show axis lines
plot_margin	Plot margins in cm
coord_type	Coordinate type ("fixed", "equal", "flip", "polar")
background_color	Plot background color
panel_background_color	Panel background color
panel_border	Show panel border
panel_border_color	Panel border color
save_plot	Logical. Whether to save the plot to a file.
save_format	Plot save format ("png", "pdf", "svg", "eps")
reverse_x	Numeric multiplier for x-axis direction (1 or -1)
reverse_y	Numeric multiplier for y-axis direction (1 or -1)
swap_axes	Logical. If TRUE, map the first reduced dimension (dim1) to the x-axis and dim2 to the y-axis. Default FALSE keeps dim2 on x and dim1 on y. Useful when the dominant axis of variation should read horizontally.
x_limits	Numeric vector of length 2 specifying c(min, max) for x-axis. If NULL, limits are set automatically.
y_limits	Numeric vector of length 2 specifying c(min, max) for y-axis. If NULL, limits are set automatically.
arrow_plot_threshold	Threshold for velocity arrows to be drawn in the same antigenic distance unit (default: 0.10)

Value

An S3 object of class `layout_config`, which is a list containing the specified configuration parameters for plot layout.

parameter_sensitivity_analysis

Parameter Sensitivity Analysis

Description

Analyzes the sensitivity of the model performance (measured by MAE) to changes in a single parameter. This function bins the parameter range to identify the minimum MAE for each bin, helping to understand how robust the model is to parameter choices.

Usage

```
parameter_sensitivity_analysis(  
  param,  
  samples,  
  bins = 30,  
  mae_col = "Holdout_MAE",  
  threshold_pct = 5,  
  min_samples = 1  
)
```

Arguments

param	The character name of the parameter to analyze.
samples	A data frame containing parameter samples and performance metrics.
bins	The integer number of bins to divide the parameter range into.
mae_col	The character name of the column containing the Mean Absolute Error (MAE) values.
threshold_pct	A numeric percentage above the minimum MAE to define an acceptable performance threshold.
min_samples	The integer minimum number of samples required in a bin for it to be included in the analysis.

Details

The function performs these steps:

1. Cleans the input data using Median Absolute Deviation (MAD) to remove outliers.
2. Bins the parameter values into equal-width bins.
3. Calculates the minimum MAE within each bin to create an empirical performance curve.
4. Identifies a performance threshold based on a percentage above the global minimum MAE.
5. Returns an S3 object for plotting and further analysis.

Value

An object of class "parameter_sensitivity" containing:

param_values	Vector of parameter bin midpoints
min_mae	Vector of minimum MAE values per bin
param_name	Name of analyzed parameter
threshold	Threshold value (default: min. +5%)
min_value	Minimum MAE value across all bins
sample_counts	Number of samples per bin

plot.parameter_sensitivity

Plot Parameter Sensitivity Analysis

Description

The S3 plot method for parameter_sensitivity objects. It creates a visualization showing how the model's performance (minimum MAE) changes across the range of a single parameter. A threshold line is included to indicate the region of acceptable performance.

Usage

```
## S3 method for class 'parameter_sensitivity'
plot(
  x,
  width = 3.5,
  height = 3.5,
  save_plot = FALSE,
  output_dir,
  y_limit_factor = NULL,
  ...
)
```

Arguments

x	A parameter_sensitivity object, typically from parameter_sensitivity_analysis().
width	The numeric width of the output plot in inches.
height	The numeric height of the output plot in inches.
save_plot	A logical indicating whether to save the plot to a file.
output_dir	A character string specifying the directory for output files. Required if save_plot is TRUE.
y_limit_factor	A numeric factor to set the upper y-axis limit as a percentage above the threshold value (e.g., 1.10 for 10% above). If NULL, scaling is automatic.
...	Additional arguments (not currently used).

Value

A ggplot object representing the sensitivity plot.

plot.profile_likelihood

Plot Method for profile_likelihood Objects

Description

Creates a visualization of the profile likelihood for a parameter, showing the maximum likelihood estimates and the 95% confidence interval. It supports mathematical notation for parameter names for clearer plot labels.

Usage

```
## S3 method for class 'profile_likelihood'
plot(x, LL_max, width = 3.5, height = 3.5, save_plot = FALSE, output_dir, ...)
```

Arguments

x	A profile_likelihood object returned by profile_likelihood().
LL_max	The global maximum log-likelihood value from the entire sample set, used as the reference for calculating the confidence interval.
width, height	Numeric. The width and height of the output plot in inches.
save_plot	Logical. If TRUE, the plot is saved to a file.
output_dir	Character. The directory where the plot will be saved. Required if save_plot is TRUE.
...	Additional arguments passed to the plot function.

Details

The 95% confidence interval is determined using the likelihood ratio test, where the cutoff is based on the chi-squared distribution: $LR(\theta_{ij}) = -2[\log L_{max}(\theta_{ij}) - \log L_{max}(\hat{\theta})]$. The interval includes all parameter values θ_{ij} for which $LR(\theta_{ij}) \leq \chi^2_{1,0.05} \approx 3.84$.

Value

A ggplot object representing the profile likelihood plot.

Examples

```
# This example can take more than 5 seconds to run.
# Create a sample data frame of MCMC samples
samples <- data.frame(
  log_N = log(runif(50, 2, 10)),
  log_k0 = log(runif(50, 1, 5)),
  log_cooling_rate = log(runif(50, 0.01, 0.1)),
  log_c_repulsion = log(runif(50, 0.1, 1)),
  NLL = runif(50, 20, 100)
)
```

```
# Calculate profile likelihood for the "log_N" parameter
pl_result <- profile_likelihood("log_N", samples, grid_size = 10)

# Provide the global maximum log-likelihood from the samples
LL_max <- max(-samples$NLL)

# The plot function requires the ggplot2 package
if (requireNamespace("ggplot2", quietly = TRUE)) {
  plot(pl_result, LL_max, width = 4, height = 3)
}
```

plot.topolow_convergence

Plot Method for topolow Convergence Diagnostics

Description

Creates visualizations of convergence diagnostics from a sampling run, including parameter mean trajectories and covariance matrix stability over iterations. This helps assess whether parameter estimation has converged.

Usage

```
## S3 method for class 'topolow_convergence'
plot(x, param_names = NULL, ...)
```

Arguments

<code>x</code>	A <code>topolow_convergence</code> object from <code>check_gaussian_convergence()</code> .
<code>param_names</code>	Optional character vector of parameter names for plot titles. If <code>NULL</code> , names are taken from the input object.
<code>...</code>	Additional arguments (not currently used).

Details

The function generates two types of plots:

1. Parameter mean plots: Shows how the mean value for each parameter changes over iterations. Stabilization of these plots indicates convergence.
2. Covariance change plot: Shows the relative change in the Frobenius norm of the covariance matrix. A decreasing trend approaching zero indicates stable relationships between parameters.

Value

A grid of plots showing convergence metrics.

See Also

[check_gaussian_convergence](#) for generating the convergence object.

Examples

```
# Example with simulated data
chain_data <- data.frame(
  param1 = rnorm(1000, mean = 1.5, sd = 0.1),
  param2 = rnorm(1000, mean = -0.5, sd = 0.2)
)

# Check convergence
results <- check_gaussian_convergence(chain_data)

# Plot diagnostics
plot(results)

# With custom parameter names
plot(results, param_names = c("Parameter 1 (log)", "Parameter 2 (log)"))
```

plot.topolow_diagnostics

Plot Method for topolow parameter estimation Diagnostics

Description

Creates trace and density plots for multiple chains to assess convergence and mixing. This is an S3 method that dispatches on topolow_diagnostics objects.

Usage

```
## S3 method for class 'topolow_diagnostics'
plot(
  x,
  output_dir,
  output_file = "topolow_param_diagnostics.png",
  save_plot = FALSE,
  ...
)
```

Arguments

x	A topolow_diagnostics object from calculate_diagnostics().
output_dir	Character. Directory for output files. Required if save_plot is TRUE.
output_file	Character path for saving the plot.
save_plot	Logical. Whether to save the plot.
...	Additional arguments passed to create_diagnostic_plots.

Value

A ggplot object of the combined plots.

plot_3d_mapping	<i>Create 3D Visualization</i>
-----------------	--------------------------------

Description

Creates an interactive or static 3D visualization using rgl. Supports both temporal and cluster-based coloring schemes with configurable point appearances and viewing options.

Usage

```
plot_3d_mapping(
  df,
  ndim,
  dim_config = new_dim_reduction_config(),
  aesthetic_config = new_aesthetic_config(),
  layout_config = new_layout_config(),
  interactive = TRUE,
  output_dir
)
```

Arguments

df	Data frame containing: • V1, V2, ... Vn: Coordinate columns • antigen: Binary indicator for antigen points • antiserum: Binary indicator for antiserum points • cluster: (Optional) Factor or integer cluster assignments • year: (Optional) Numeric year values for temporal coloring
ndim	Number of dimensions in input coordinates (must be >= 3)
dim_config	Dimension reduction configuration object
aesthetic_config	Aesthetic configuration object
layout_config	Layout configuration object
interactive	Logical; whether to create an interactive plot
output_dir	Character. Directory for output files. Required if interactive is FALSE.

Details

The function supports two main visualization modes:

1. Interactive mode: Creates a manipulatable 3D plot window
2. Static mode: Generates a static image from a fixed viewpoint

Color schemes are automatically selected based on available data:

- If cluster data is present: Uses discrete colors per cluster
- If year data is present: Uses continuous color gradient
- Otherwise: Uses default point colors

For data with more than 3 dimensions, dimension reduction is applied first.

Note: This function requires the `rgl` package and OpenGL support. If `rgl` is not available, the function will return a 2D plot with a message explaining how to enable 3D visualization.

Value

Invisibly returns the `rgl` scene ID for further manipulation if `rgl` is available, or a 2D `ggplot` object as a fallback.

See Also

[plot_temporal_mapping](#) for 2D temporal visualization [plot_cluster_mapping](#) for 2D cluster visualization [make_interactive](#) for converting 2D plots to interactive versions

Examples

```
# Create sample data
set.seed(123)
data <- data.frame(
  V1 = rnorm(100), V2 = rnorm(100), V3 = rnorm(100), V4 = rnorm(100), name = 1:100,
  antigen = rep(c(0,1), 50), antiserum = rep(c(1,0), 50),
  cluster = rep(1:5, each=20), year = rep(2000:2009, each=10)
)

# Create a static plot and save to a temporary file
# This example requires an interactive session and the 'rgl' package.
if (interactive() && requireNamespace("rgl", quietly = TRUE)) {
  temp_dir <- tempdir()
  # Basic interactive plot (will open a new window)
  if(interactive()) {
    plot_3d_mapping(data, ndim=4)
  }
}

# Custom configuration for temporal visualization
aesthetic_config <- new_aesthetic_config(
  point_size = 5,
  point_alpha = 0.8,
  gradient_colors = list(
    low = "blue",
```

```

        high = "red"
    )
)

layout_config <- new_layout_config(
  width = 12,
  height = 12,
  background_color = "black",
  show_axis = TRUE
)
# Create customized static plot and save it
plot_3d_mapping(data, ndim=4,
  aesthetic_config = aesthetic_config,
  layout_config = layout_config,
  interactive = FALSE, output_dir = temp_dir
)
list.files(temp_dir)
unlink(temp_dir, recursive = TRUE)
}

```

plot_cluster_mapping *Create Clustered Mapping Plots*

Description

Antigenic Mapping and Antigenic Velocity Function. See `plot_temporal_mapping` for the meaning of the new `sigma_x_nd` parameter and the structure of the returned `antigenic_velocity` object. All velocity behaviour is identical between the two functions; the on-screen arrows are 2-D plot-space, the returned velocities are n-D.

Usage

```

plot_cluster_mapping(
  df_coords,
  ndim,
  dim_config = new_dim_reduction_config(),
  aesthetic_config = new_aesthetic_config(),
  layout_config = new_layout_config(),
  annotation_config = new_annotation_config(),
  output_dir,
  show_shape_legend = TRUE,
  cluster_legend_title = "Cluster",
  draw_arrows = FALSE,
  annotate_arrows = TRUE,
  phylo_tree = NULL,
  sigma_t = NULL,
  sigma_x = NULL,
  sigma_x_nd = NULL,

```



```

    clade_node_depth = NULL,
    show_one_arrow_per_cluster = FALSE,
    cluster_legend_order = NULL
)

```

Arguments

df_coords	Data frame containing: • V1, V2, ... Vn: Coordinate columns • antigen: Binary indicator for antigen points • antiserum: Binary indicator for antiserum points • year: Numeric year values for temporal colouring
ndim	Number of dimensions in input coordinates
dim_config	Dimension reduction configuration object.
aesthetic_config	Aesthetic configuration object.
layout_config	Layout configuration object. Use x_limits / y_limits to set axis limits.
annotation_config	Annotation configuration object.
output_dir	Character. Directory for output files. Required if layout_config\$save_plot is TRUE.
show_shape_legend	Logical.
cluster_legend_title	Character. Custom title for the cluster legend.
draw_arrows	logical; if TRUE, compute velocity (n-D and 2-D) and draw the 2-D vectors on the map.
annotate_arrows	logical; if TRUE, show names of the points having arrows.
phylo_tree	Optional; phylo object. Does not need to be rooted. If provided, antigens from different clades do not contribute to each other's velocity.
sigma_t	Optional numeric; bandwidth for the Gaussian kernel on time (years, or whatever units year is in). Used in BOTH the n-D and 2-D passes. Default: Silverman's rule on pairwise time differences.
sigma_x	Optional numeric; bandwidth for the Gaussian kernel on antigenic distance in the 2-D plot space. Controls the arrows on the map. Default: Silverman's rule on the 2-D coordinates.
sigma_x_nd	Optional numeric; bandwidth for the Gaussian kernel on antigenic distance in the original n-D space. Controls the velocities returned to the caller in antigenic_velocity. Default: RMS of per-dimension Silverman bandwidths over V1...V_ndim.
clade_node_depth	Optional integer; number of parent levels used to define a clade. Default: median leaf-to-backbone node-distance of the tree.

`show_one_arrow_per_cluster`
 Show only the largest 2-D arrow per cluster (filtering is on the 2-D plot magnitude, matching prior behaviour).

`cluster_legend_order`
 Optional ordering for clusters in the legend.

Details

The function performs these steps:

1. Validates input data structure and types
2. Applies dimension reduction if `ndim > 2`
3. Creates visualization with cluster-based coloring
4. Applies specified aesthetic and layout configurations
5. Applies custom axis limits if specified in `layout_config`

Different shapes distinguish between antigens and antisera points, while color represents cluster assignment. The color palette can be customized through the `aesthetic_config`.

Value

If `draw_arrows = FALSE`, a single ggplot object. If `draw_arrows = TRUE`, a list with elements `plot` and `antigenic_velocity` (see `plot_temporal_mapping`).

See Also

[plot_temporal_mapping](#) for temporal visualization [plot_3d_mapping](#) for 3D visualization [new_dim_reduction_config](#) for dimension reduction options [new_aesthetic_config](#) for aesthetic options [new_layout_config](#) for layout options [new_annotation_config](#) for annotation options

Examples

```
# Basic usage with default configurations
data <- data.frame(
  V1 = rnorm(100), V2 = rnorm(100), V3 = rnorm(100), name = 1:100,
  antigen = rep(c(0,1), 50), antiserum = rep(c(1,0), 50),
  cluster = rep(1:5, each=20)
)
p1 <- plot_cluster_mapping(data, ndim=3)

# Save plot to a temporary directory
temp_dir <- tempdir()
# Custom configurations with specific color palette and axis limits
aesthetic_config <- new_aesthetic_config(
  point_size = 4,
  point_alpha = 0.7,
  color_palette = c("red", "blue", "green", "purple", "orange"),
  show_labels = TRUE,
  label_size = 3
)
```

```

layout_config_save <- new_layout_config(save_plot = TRUE,
  width = 10,
  height = 8,
  coord_type = "fixed",
  show_grid = TRUE,
  grid_type = "major",
  x_limits = c(-10, 10),
  y_limits = c(-8, 8)
)

p_saved <- plot_cluster_mapping(data, ndim=3,
  layout_config = layout_config_save,
  aesthetic_config = aesthetic_config,
  output_dir = temp_dir
)

list.files(temp_dir)
unlink(temp_dir, recursive = TRUE)

```

plot_euclidify_diagnostics

Plot Euclidify Optimization Diagnostics

Description

Creates comprehensive diagnostic plots for the Euclidify optimization process, including parameter search trajectory and embedding quality metrics.

Usage

```

plot_euclidify_diagnostics(
  euclidify_result,
  plot_types = "all",
  save_plots = TRUE,
  output_dir = NULL,
  width = 12,
  height = 8,
  dpi = 300,
  return_plots = TRUE
)

```

Arguments

euclidify_result	A result object from Euclidify() with create_diagnostic_plots=TRUE.
plot_types	Character vector specifying which plots to create. Options: "all", "parameter_search", "convergence", "quality", "cv_errors". Default: "all"

save_plots	Logical. Whether to save plots to files. Default: TRUE
output_dir	Character. Directory for saving plots. Required if save_plots=TRUE.
width	Numeric. Plot width in inches. Default: 12
height	Numeric. Plot height in inches. Default: 8
dpi	Numeric. Resolution for saved plots. Default: 300
return_plots	Logical. Whether to return plot objects. Default: TRUE

Details

This function creates several diagnostic visualizations:

- Parameter Search: Shows explored parameter space in 2D projections
- Convergence Trace: Plots MAE and parameter evolution during final embedding
- Quality Metrics: Scatter plots of predicted vs true distances
- CV Errors: Distribution of cross-validation errors across folds

Value

A list of ggplot objects if return_plots=TRUE, otherwise NULL invisibly.

Examples

```
# A small dissimilarity matrix to keep the example fast.
test_data <- data.frame(
  object = rep(paste0("Obj", 1:4), each = 4),
  reference = rep(paste0("Ref", 1:4), 4),
  score = sample(c(1, 2, 4, 8, 16, 32, 64, "<1", ">12"), 16, replace = TRUE)
)
dist_mat <- list_to_matrix(
  data = test_data, object_col = "object", reference_col = "reference",
  value_col = "score", is_similarity = TRUE
)

# Run Euclidify with diagnostics enabled. Write only to a directory you
# choose; tempdir() is used here.
result <- Euclidify(
  dissimilarity_matrix = dist_mat,
  output_dir = tempdir(check = TRUE),
  n_initial_samples = 5,
  n_adaptive_samples = 3,
  folds = 3,
  mapping_max_iter = 50,
  ndim_range = c(2, 4),
  max_cores = 1,
  verbose = "off",
  create_diagnostic_plots = TRUE,
  diagnostic_plot_types = "parameter_search"
)
```

```
# Build the plots without writing them to disk. plot_types = "all" (the
# default) additionally produces the convergence, quality and cv_errors
# panels; one type is requested here to keep the example quick.
plots <- plot_euclidify_diagnostics(
  result,
  plot_types = "parameter_search",
  save_plots = FALSE
)

# Text summary of the same run
report <- create_diagnostic_report(result)
cat(report, sep = "\n")
```

plot_ll_improvement *Plot Log-Likelihood Improvement Over Iterations*

Description

A visualization showing the improvement in log-likelihood relative to the first sample, making it easy to see when gains diminish.

Usage

```
plot_ll_improvement(chain_files, combine_chains = TRUE)
```

Arguments

chain_files Character vector. Paths to CSV files containing sampling chains.
combine_chains Logical. If TRUE, combines all chains. Default: TRUE

Value

A ggplot object.

plot_mcmc_diagnostics *Model Diagnostics and Convergence Testing Create Diagnostic Plots for Multiple Sampling Chains*

Description

Creates trace and density plots for multiple sampling or optimization chains to help assess convergence and mixing. It displays parameter trajectories and their distributions across all chains.

Usage

```
plot_mcmc_diagnostics(
  chain_files,
  mutual_size = 2000,
  output_file = "diagnostic_plots.png",
  output_dir,
  save_plot = FALSE,
  width = 3000,
  height = 3000,
  dpi = 300
)
```

Arguments

<code>chain_files</code>	A character vector of paths to CSV files, where each file contains data for one chain.
<code>mutual_size</code>	Integer. The number of samples to use from the end of each chain for plotting.
<code>output_file</code>	Character. The path for saving the plot. Required if <code>save_plot</code> is TRUE.
<code>output_dir</code>	Character. The directory for saving output files. Required if <code>save_plot</code> is TRUE.
<code>save_plot</code>	Logical. If TRUE, saves the plot to a file. Default: FALSE.
<code>width, height, dpi</code>	Numeric. The dimensions and resolution for the saved plot.

Value

A ggplot object of the combined plots.

Examples

```
# This example uses sample data files that would be included with the package.
chain_files <- c(
  system.file("extdata", "diag_chain1.csv", package = "topolow"),
  system.file("extdata", "diag_chain2.csv", package = "topolow"),
  system.file("extdata", "diag_chain3.csv", package = "topolow")
)

# Only run the example if the files are found
if (all(nzchar(chain_files))) {
  # Create diagnostic plot without saving to a file
  plot_mcmc_diagnostics(chain_files, mutual_size = 50, save_plot = FALSE)
}
```

`plot_network_structure`*Plot Network Structure*

Description

Creates a visualization of the dissimilarity matrix as a network graph, showing data availability patterns and connectivity between points.

Usage

```
plot_network_structure(  
  network_results,  
  output_file = NULL,  
  width = 3000,  
  height = 3000,  
  dpi = 300  
)
```

Arguments

<code>network_results</code>	The list output from <code>analyze_network_structure()</code> .
<code>output_file</code>	Character. An optional full path to save the plot. If <code>NULL</code> , the plot is not saved.
<code>width</code>	Numeric. Width in pixels for saved plot (default: 3000).
<code>height</code>	Numeric. Height in pixels for saved plot (default: 3000).
<code>dpi</code>	Numeric. Resolution in dots per inch (default: 300).

Value

A ggplot object representing the network graph.

Examples

```
# Create a sample dissimilarity matrix  
adj_mat <- matrix(runif(25), 5, 5)  
rownames(adj_mat) <- colnames(adj_mat) <- paste0("Point", 1:5)  
adj_mat[lower.tri(adj_mat)] <- t(adj_mat)[lower.tri(adj_mat)]  
diag(adj_mat) <- 0  
net_analysis <- analyze_network_structure(adj_mat)  
  
# Create and display the plot  
plot_network_structure(net_analysis)
```

plot_performance_trace

Plot Running Minimum Error to Show Sampling Convergence

Description

Creates a diagnostic plot showing how the best (minimum) NLL or MAE found improves over iterations and eventually plateaus. This is a standard way to demonstrate that sampling/optimization has converged in terms of the objective function.

Usage

```
plot_performance_trace(
  chain_files,
  metric = "both",
  combine_chains = TRUE,
  sort_combined = TRUE,
  trim_worst = 0.1,
  show_raw = TRUE,
  window_size = NULL,
  output_file = NULL,
  width = 10,
  height = 6,
  dpi = 300
)
```

Arguments

chain_files	Character vector. Paths to CSV files containing sampling chains.
metric	Character. Which metric to plot: "NLL", "MAE", or "both". Default: "both"
combine_chains	Logical. If TRUE, combines all chains into one sequence. If FALSE, plots each chain separately. Default: TRUE
sort_combined	Logical. If TRUE and combine_chains is TRUE, sorts combined samples from worst to best for each metric independently before computing the running minimum. This produces a smooth monotonic improvement curve that is not biased by chain ordering. If FALSE, chains are concatenated sequentially (original behavior). Default: TRUE
trim_worst	Numeric between 0 and 1. Fraction of the worst-performing samples to remove before plotting when combine_chains = TRUE and sort_combined = TRUE. This focuses the visualization on the convergence plateau. Default: 0.1
show_raw	Logical. If TRUE, shows raw values as points behind the running minimum. Default: TRUE
window_size	Integer. Window size for computing rolling mean (optional smoothing). Set to NULL for no smoothing. Default: NULL
output_file	Character. Optional path to save the plot. Default: NULL

width, height	Numeric. Dimensions for saved plot. Default: 10, 6
dpi	Numeric. Resolution for saved plot. Default: 300

Details

The "running minimum" (or cumulative minimum) shows the best value found so far at each iteration. When this line flattens out (plateaus), it indicates that the sampling has found the optimal region and additional samples are not improving the objective.

When `combine_chains = TRUE` and `sort_combined = TRUE`, samples from all chains are sorted from worst (highest) to best (lowest) independently for each metric before computing the running minimum. This avoids the problem where the first chain in the list dominates the visual, and produces a smooth improvement curve showing how quality improves across the full pool of samples. Each metric facet has its own sort order, so the NLL panel sorts by NLL and the MAE panel sorts by MAE.

This is distinct from:

- Trace plots (which show parameter values, not objective function)
- R-hat (which measures between-chain variance)
- ESS (which measures autocorrelation)

Value

A ggplot object showing the error plateau diagnostic.

Examples

```
# Chain files are CSVs written by run_adaptive_sampling(); each must have
# NLL and Holdout_MAE columns. Small synthetic chains stand in here.
# tempdir(check = TRUE) recreates the session temp directory if it has been
# removed, and the subdirectory is created explicitly before writing.
ex_dir <- file.path(tempdir(check = TRUE), "topolow_perf_example")
dir.create(ex_dir, showWarnings = FALSE, recursive = TRUE)
chain_files <- file.path(ex_dir, paste0("perf_chain", 1:2, ".csv"))
for (f in chain_files) {
  utils::write.csv(
    data.frame(NLL = rnorm(50, 100, 10), Holdout_MAE = runif(50, 0.5, 2)),
    f, row.names = FALSE
  )
}

# Basic usage (sorted by default)
p <- plot_performance_trace(chain_files)

# Original sequential concatenation behaviour
p <- plot_performance_trace(chain_files, sort_combined = FALSE)

# Plot only NLL, with the chains kept separate
p <- plot_performance_trace(chain_files, metric = "NLL", combine_chains = FALSE)

# Save the plot. Write only to a directory you choose; tempdir() here.
```

```

p <- plot_performance_trace(
  chain_files,
  output_file = file.path(ex_dir, "convergence_plateau.png")
)

unlink(ex_dir, recursive = TRUE)

```

plot_temporal_mapping *Create Temporal Mapping Plot*

Description

Antigenic Mapping and Antigenic Velocity Function. Creates a visualization of points coloured by time (year) using dimension reduction, with optional antigenic velocity arrows. Points are coloured on a gradient scale based on their temporal values, with different shapes for antigens and antisera.

Antigenic velocity is computed in two spaces:

- the original `ndim`-dimensional space (using `V1..V_ndim` in `df_coords`); the result is returned to the caller as `antigenic_velocity`.
- the 2-dimensional plot space (after dimension reduction); used only to draw arrows on the map.

This means the magnitudes and directions you see on the map are plot-space quantities, while the magnitudes and directions in the returned `antigenic_velocity` object are the geometrically meaningful `n-D` values. The two will agree exactly only when `ndim == 2`.

Usage

```

plot_temporal_mapping(
  df_coords,
  ndim,
  dim_config = new_dim_reduction_config(),
  aesthetic_config = new_aesthetic_config(),
  layout_config = new_layout_config(),
  annotation_config = new_annotation_config(),
  output_dir,
  show_shape_legend = TRUE,
  draw_arrows = FALSE,
  annotate_arrows = TRUE,
  phylo_tree = NULL,
  sigma_t = NULL,
  sigma_x = NULL,
  sigma_x_nd = NULL,
  clade_node_depth = NULL
)

```

Arguments

<code>df_coords</code>	Data frame containing: • <code>V1, V2, ... Vn</code>: Coordinate columns • <code>antigen</code>: Binary indicator for antigen points • <code>antiserum</code>: Binary indicator for antiserum points • <code>year</code>: Numeric year values for temporal colouring
<code>ndim</code>	Number of dimensions in input coordinates
<code>dim_config</code>	Dimension reduction configuration object.
<code>aesthetic_config</code>	Aesthetic configuration object.
<code>layout_config</code>	Layout configuration object. Use <code>x_limits / y_limits</code> to set axis limits.
<code>annotation_config</code>	Annotation configuration object.
<code>output_dir</code>	Character. Directory for output files. Required if <code>layout_config\$save_plot</code> is TRUE.
<code>show_shape_legend</code>	Logical.
<code>draw_arrows</code>	logical; if TRUE, compute velocity (n-D and 2-D) and draw the 2-D vectors on the map.
<code>annotate_arrows</code>	logical; if TRUE, show names of the points having arrows.
<code>phylo_tree</code>	Optional; phylo object. Does not need to be rooted. If provided, antigens from different clades do not contribute to each other's velocity.
<code>sigma_t</code>	Optional numeric; bandwidth for the Gaussian kernel on time (years, or whatever units year is in). Used in BOTH the n-D and 2-D passes. Default: Silverman's rule on pairwise time differences.
<code>sigma_x</code>	Optional numeric; bandwidth for the Gaussian kernel on antigenic distance in the 2-D plot space. Controls the arrows on the map. Default: Silverman's rule on the 2-D coordinates.
<code>sigma_x_nd</code>	Optional numeric; bandwidth for the Gaussian kernel on antigenic distance in the original n-D space. Controls the velocities returned to the caller in <code>antigenic_velocity</code> . Default: RMS of per-dimension Silverman bandwidths over <code>V1..V_ndim</code> .
<code>clade_node_depth</code>	Optional integer; number of parent levels used to define a clade. Default: median leaf-to-backbone node-distance of the tree.

Details

The function performs these steps:

1. Validates input data structure and types
2. Applies dimension reduction if `ndim > 2`
3. Creates visualization with temporal color gradient

4. Applies specified aesthetic and layout configurations
5. Applies custom axis limits if specified in layout_config

Different shapes distinguish between antigens and antisera points, while color represents temporal progression.

Value

If `draw_arrows = FALSE`, a single ggplot object. If `draw_arrows = TRUE`, a list with elements:

`plot` The ggplot object.

`antigenic_velocity` Data frame with one row per antigen and columns `name`, `V1`...`V_ndim` (original n-D coordinates), `year`, `v1`...`v_ndim` (n-D velocity components), `velocity_magnitude` (Euclidean magnitude of the n-D velocity). This replaces the previous `velocity_data` element, which was 2-D; pre-existing scripts that read `result$velocity_data` or reference `$mag` must be updated.

See Also

[plot_cluster_mapping](#) for cluster-based visualization [plot_3d_mapping](#) for 3D visualization [new_dim_reduction_config](#) for dimension reduction options [new_aesthetic_config](#) for aesthetic options [new_layout_config](#) for layout options [new_annotation_config](#) for annotation options

Examples

```
# Basic usage with default configurations
data <- data.frame(
  V1 = rnorm(100), V2 = rnorm(100), V3 = rnorm(100), name = 1:100,
  antigen = rep(c(0,1), 50), antiserum = rep(c(1,0), 50),
  year = rep(2000:2009, each=10)
)
# Plot without saving
p1 <- plot_temporal_mapping(data, ndim=3)

# Save plot to a temporary directory
temp_dir <- tempdir()
layout_config_save <- new_layout_config(save_plot = TRUE,
  x_limits = c(-10, 10),
  y_limits = c(-8, 8))
p_saved <- plot_temporal_mapping(data, ndim = 3, layout_config = layout_config_save,
  output_dir = temp_dir)
list.files(temp_dir) # Check that file was created
unlink(temp_dir, recursive = TRUE) # Clean up

# With antigenic velocity vectors (returned object structure)
set.seed(1)
n <- 60
data2 <- data.frame(
  V1 = rnorm(n), V2 = rnorm(n), V3 = rnorm(n),
  name = paste0("v", seq_len(n)),
```

```

    antigen = rep(c(0, 1), n / 2),
    antiserum = rep(c(1, 0), n / 2),
    year = sample(2000:2015, n, replace = TRUE)
  )
  result <- plot_temporal_mapping(data2, ndim = 3, draw_arrows = TRUE)
  result$plot
  head(result$antigenic_velocity) # name, V1..V3, year, v1..v3, velocity_magnitude

```

```
print.parameter_sensitivity
```

Print Method for Parameter Sensitivity Objects

Description

The S3 print method for parameter_sensitivity objects. It displays a concise summary of the analysis results, including the parameter analyzed, the minimum error found, and the performance threshold.

Usage

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

Arguments

x	A parameter_sensitivity object.
...	Additional arguments passed to the print function (not currently used).

Value

Invisibly returns the original object. Called for its side effect of printing a summary to the console.

```
print.profile_likelihood
```

Print Method for profile_likelihood Objects

Description

Provides a concise summary of a profile_likelihood object.

Usage

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

Arguments

`x` A profile_likelihood object.

`...` Additional arguments passed to print.

Value

The original profile_likelihood object (invisibly). Called for its side effect of printing a summary to the console.

print.topolow	<i>Print method for topolow objects</i>
---------------	---

Description

Provides a concise display of key optimization results from euclidean_embedding.

Usage

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

Arguments

`x` A topolow object returned by euclidean_embedding().

`...` Additional arguments passed to print (not used).

Value

The original topolow object (invisibly). This function is called for its side effect of printing a summary to the console.

Examples

```
# Create a simple dissimilarity matrix and run the optimization
dist_mat <- matrix(c(0, 2, 3, 2, 0, 4, 3, 4, 0), nrow=3)
result <- euclidean_embedding(dist_mat, ndim=2, mapping_max_iter=50,
                             k0=1.0, cooling_rate=0.001, c_repulsion=0.1,
                             verbose = FALSE)

# Print the result object
print(result)
```

```
print.topolow_convergence
```

Print Method for topolow Convergence Diagnostics

Description

Print Method for topolow Convergence Diagnostics

Usage

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

Arguments

x	A topolow_convergence object.
...	Additional arguments passed to print.

Value

No return value; called for its side effect of printing a summary.

```
print.topolow_diagnostics
```

Print Method for topolow parameter estimation Diagnostics

Description

Print Method for topolow parameter estimation Diagnostics

Usage

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

Arguments

x	A topolow_diagnostics object.
...	Additional arguments passed to print.

Value

No return value; called for its side effect of printing a summary.

process_antigenic_data

Process Raw Antigenic Assay Data

Description

Processes raw antigenic assay data from data frames into standardized long and matrix formats. Handles both similarity data (like titers, which need conversion to distances) and direct dissimilarity measurements like IC50. Preserves threshold indicators (<, >) and handles repeated measurements by averaging.

Usage

```
process_antigenic_data(
  data,
  antigen_col,
  serum_col,
  value_col,
  is_similarity = FALSE,
  metadata_cols = NULL,
  base = NULL,
  scale_factor = 1
)
```

Arguments

data	Data frame containing raw data.
antigen_col	Character. Name of column containing virus/antigen identifiers.
serum_col	Character. Name of column containing serum/antibody identifiers.
value_col	Character. Name of column containing measurements (titers or distances).
is_similarity	Logical. Whether values are measures of similarity such as titers or binding affinities (TRUE) or dissimilarities like IC50 (FALSE). Default: FALSE.
metadata_cols	Character vector. Names of additional columns to preserve.
base	Numeric. Base for logarithm transformation (default: 2 for similarities, e for dissimilarities).
scale_factor	Numeric. Scale factor for similarities. This is the base value that all other dilutions are multiples of. E.g., 10 for HI assay where titers are 10, 20, 40,... Default: 1.

Details

The function handles these key steps:

1. Validates input data and required columns
2. Transforms values to log scale

3. Converts similarities to distances using Smith's method if needed
4. Averages repeated measurements
5. Creates standardized long format
6. Creates symmetric distance matrix
7. Preserves metadata and threshold indicators
8. Preserves virusYear and serumYear columns if present

Input requirements and constraints:

- Data frame must contain required columns
- Column names must match specified parameters
- Values can include threshold indicators (< or >)
- Metadata columns must exist if specified
- Allowed Year-related column names are "virusYear" and "serumYear"

Value

A list containing two elements:

long	A data.frame in long format with standardized columns, including the original identifiers, processed values, and calculated distances. Any specified metadata is also included.
matrix	A numeric matrix representing the processed symmetric distance matrix, with antigens and sera on columns and rows.

Examples

```
# Example 1: Processing HI titer data (similarities)
antigen_data <- data.frame(
  virus = c("A/H1N1/2009", "A/H1N1/2010", "A/H1N1/2011", "A/H1N1/2009", "A/H1N1/2010"),
  serum = c("anti-2009", "anti-2009", "anti-2009", "anti-2010", "anti-2010"),
  titer = c(1280, 640, "<40", 2560, 1280), # Some below detection limit
  cluster = c("A", "A", "B", "A", "A"),
  color = c("red", "red", "blue", "red", "red")
)

# Process HI titer data (similarities -> distances)
results <- process_antigenic_data(
  data = antigen_data,
  antigen_col = "virus",
  serum_col = "serum",
  value_col = "titer",
  is_similarity = TRUE, # Titers are similarities
  metadata_cols = c("cluster", "color"),
  scale_factor = 10 # Base dilution factor
)

# View the long format data
print(results$long)
```

```
# View the distance matrix
print(results$matrix)

# Example 2: Processing IC50 data (already dissimilarities)
ic50_data <- data.frame(
  virus = c("HIV-1", "HIV-2", "HIV-3"),
  antibody = c("mAb1", "mAb1", "mAb2"),
  ic50 = c(0.05, ">10", 0.2)
)

results_ic50 <- process_antigenic_data(
  data = ic50_data,
  antigen_col = "virus",
  serum_col = "antibody",
  value_col = "ic50",
  is_similarity = FALSE # IC50 values are dissimilarities
)
```

profile_likelihood *Profile Likelihood Analysis*

Description

Calculates the profile likelihood for a given parameter by evaluating the conditional maximum likelihood across a grid of parameter values. This "empirical profile likelihood" estimates the likelihood surface based on samples from Monte Carlo simulations.

Usage

```
profile_likelihood(
  param,
  samples,
  grid_size = 40,
  bandwidth_factor = 0.05,
  start_factor = 0.5,
  end_factor = 1.5,
  min_samples = 5
)
```

Arguments

param	The character name of the parameter to analyze (e.g., "log_N").
samples	A data frame containing parameter samples and a log-likelihoods column named "NLL".
grid_size	The integer number of grid points for the analysis.
bandwidth_factor	A numeric factor for the local sample window size.

start_factor, end_factor	Numeric range multipliers for parameter grid (default: 0.5, 1.2)
min_samples	Integer minimum samples required for reliable estimate (default: 10)

Details

For each value in the parameter grid, the function:

1. Identifies nearby samples using a bandwidth window.
2. Calculates the conditional maximum likelihood from these samples.
3. Tracks sample counts to assess the reliability of the estimate.

Value

Object of class "profile_likelihood" containing:

param	Vector of parameter values
ll	Vector of log-likelihood values
param_name	Name of analyzed parameter
bandwidth	Bandwidth used for local windows
sample_counts	Number of samples per estimate

See Also

The S3 methods `print.profile_likelihood` and `summary.profile_likelihood` for viewing results.

Examples

```
# Create a sample data frame of parameter samples
mcmc_samples <- data.frame(
  log_N = log(runif(50, 2, 10)),
  log_k0 = log(runif(50, 1, 5)),
  log_cooling_rate = log(runif(50, 0.01, 0.1)),
  log_c_repulsion = log(runif(50, 0.1, 1)),
  NLL = runif(50, 20, 100)
)

# Calculate profile likelihood for the "log_N" parameter
pl <- profile_likelihood("log_N", mcmc_samples,
  grid_size = 10, # Smaller grid for a quick example
  bandwidth_factor = 0.05)

# Print the results
print(pl)
```

prune_sparse_matrix *Prune Sparse Dissimilarity Matrix to Well-Connected Subset*

Description

Iteratively removes poorly connected points from a sparse dissimilarity matrix to create a well-connected, denser subset suitable for optimization and subsampling. This is particularly useful for very sparse datasets (>95% missing) where random subsampling would create disconnected graphs.

Usage

```
prune_sparse_matrix(
  dissimilarity_matrix,
  min_connections = 4,
  target_completeness = NULL,
  max_iterations = 100,
  min_points = 10,
  ensure_connected = TRUE,
  verbose = TRUE
)
```

Arguments

dissimilarity_matrix	Square symmetric dissimilarity matrix. Can contain NA values and threshold indicators (< or >).
min_connections	Integer. Minimum number of observed dissimilarities required per point. Points with fewer connections are iteratively removed. Default: 4. Higher values create denser but smaller subsets.
target_completeness	Numeric. Target network completeness (0-1) to achieve. If specified, overrides min_connections and adaptively increases the threshold until target is reached. Default: NULL.
max_iterations	Integer. Maximum pruning iterations to prevent infinite loops. Default: 100.
min_points	Integer. Minimum number of points to retain. Stops pruning if fewer points remain. Default: 10.
ensure_connected	Logical. If TRUE, verifies final subset is connected and warns if not. Requires igraph package. Default: TRUE.
verbose	Logical. Print progress messages. Default: TRUE.

Details

The function works by:

1. Counting observed measurements per point (row/column)

2. Identifying points below the threshold
3. Removing those points and their measurements
4. Repeating until all remaining points meet the threshold
5. Optionally verifying connectivity

Choosing min_connections:

- For very sparse data, start with min_connections = 3-10
- For target completeness, use target_completeness instead

Adaptive thresholding with target_completeness: If target_completeness is specified, the function:

1. Starts with min_connections = 4
2. Prunes the matrix
3. Checks if completeness target is met
4. If not, increases min_connections by 1 and repeats
5. Stops when target is reached or min_points is hit

Value

A list containing:

pruned_matrix	Matrix. The pruned dissimilarity matrix.
kept_indices	Integer vector. Row/column indices of kept points.
kept_names	Character vector. Names of kept points (if original had names).
removed_indices	Integer vector. Indices of removed points.
stats	List of pruning statistics: • original_size: Original matrix dimensions • final_size: Final matrix dimensions • points_removed: Number of points removed • percent_removed: Percentage of points removed • original_completeness: Original network completeness • final_completeness: Final network completeness • original_measurements: Original number of observed values • final_measurements: Final number of observed values • iterations: Number of pruning iterations • min_connections_used: Final min_connections threshold • is_connected: Whether final subset is connected (if checked) • n_components: Number of connected components (if checked)

See Also

[check_matrix_connectivity](#) for checking connectivity, [analyze_network_structure](#) for network analysis

Examples

```
# Create a sparse matrix
set.seed(123)
n <- 1000
mat <- matrix(NA, n, n)
diag(mat) <- 0
# Add only 1% observations
n_obs <- floor(n * n * 0.15)
indices <- sample(which(upper.tri(mat)), n_obs)
mat[indices] <- runif(n_obs, 0, 10)
mat[lower.tri(mat)] <- t(mat)[lower.tri(mat)]

# Prune with minimum connections
result <- prune_sparse_matrix(mat, min_connections = 2)
print(result$stats)

# Prune to target completeness
result2 <- prune_sparse_matrix(mat, target_completeness = 0.05)
print(result2$stats)
```

run_adaptive_sampling *Run Adaptive Monte Carlo Sampling for Parameter Refinement*

Description

Refines parameter estimates using adaptive Monte Carlo sampling. This function uses initial parameter samples and iteratively generates new samples concentrated in high-likelihood regions of parameter space. Multiple parallel jobs explore different regions simultaneously. Results from all parallel jobs accumulate in a single output file.

Usage

```
run_adaptive_sampling(
  initial_samples_file,
  scenario_name,
  dissimilarity_matrix,
  max_cores = NULL,
  num_samples = 10,
  mapping_max_iter = 500,
  relative_epsilon = 1e-04,
  folds = 20,
  opt_subsample = NULL,
  output_dir,
  verbose = FALSE,
  preserve_order = FALSE
)
```

Arguments

initial_samples_file	Character. Path to a CSV file containing initial samples (typically from initial_parameter_optimization).
scenario_name	Character. Name for the output files.
dissimilarity_matrix	Matrix. The input dissimilarity matrix.
max_cores	Integer. Number of cores to use for parallel execution. If NULL, uses all available cores minus 1.
num_samples	Integer. Number of new samples to generate via adaptive sampling per parallel job. Default: 10.
mapping_max_iter	Integer. Maximum number of map optimization iterations.
relative_epsilon	Numeric. Convergence threshold for relative change in error. Default: 1e-4.
folds	Integer. Number of cross-validation folds. Default: 20.
opt_subsample	Integer or NULL. If specified, randomly samples this many points from the dissimilarity matrix for each adaptive sampling iteration to reduce computational cost. The function automatically validates that subsampled data forms a connected graph. Default: NULL (use full data).
Important Notes:	
• If connectivity cannot be achieved, another subsample will be attempted up to a maximum number of attempts. • Each parallel job uses a different subsample • Recommended: use same value as in initial_parameter_optimization • The actual subsample size used is reported in output columns	
output_dir	Character. Required directory for output files.
verbose	Logical. Whether to print progress messages. Default: FALSE.
preserve_order	Logical. Whether to preserve the order of points in the output. Default: FALSE.

Details

The function runs multiple parallel jobs, each performing adaptive sampling starting from the initial samples. The jobs are independent and can be run simultaneously. Results from all jobs are combined with the initial samples and written to a single output file.

If `opt_subsample` is used, each parallel job will independently subsample the data, ensuring diversity in the parameter evaluations while maintaining computational efficiency.

Value

No return value. Called for its side effect of writing results to a CSV file in `output_dir`.

See Also

[adaptive_MC_sampling](#) for the underlying sampling algorithm, [initial_parameter_optimization](#) for generating initial samples.

Examples

```

# 1. Locate the example initial samples file included with the package
# In a real scenario, this file would be from an 'initial_parameter_optimization' run.
initial_file <- system.file(
  "extdata", "initial_samples_example.csv",
  package = "topolow"
)

# 2. Create a temporary directory for the function's output
# This function requires a writable directory for its results.
temp_out_dir <- tempdir()

# 3. Create a sample dissimilarity matrix
dissim_mat <- matrix(runif(900, 1, 10), 30, 30)
diag(dissim_mat) <- 0

# 4. Run adaptive sampling (full data)
if (nzchar(initial_file)) {
  run_adaptive_sampling(
    initial_samples_file = initial_file,
    scenario_name = "adaptive_test_example",
    dissimilarity_matrix = dissim_mat,
    output_dir = temp_out_dir,
    max_cores = 1,
    num_samples = 1,
    verbose = FALSE
  )
}

# 5. Run adaptive sampling with subsampling
if (nzchar(initial_file)) {
  run_adaptive_sampling(
    initial_samples_file = initial_file,
    scenario_name = "adaptive_test_subsample",
    dissimilarity_matrix = dissim_mat,
    output_dir = temp_out_dir,
    max_cores = 1,
    num_samples = 1,
    opt_subsample = 15, # Use only 8 points
    verbose = TRUE
  )
}

# Clean up
unlink(temp_out_dir, recursive = TRUE)

```


Description

Validates that a subsampled dissimilarity matrix has sufficient data for reliable parameter optimization with cross-validation. We need enough ground truth in each fold to be able to compare the predictions with them with high confidence.

Usage

```
sanity_check_subsample(
  subsampled_matrix,
  folds = 20,
  min_points_per_fold = 3,
  min_measurements_per_fold = 3,
  verbose = TRUE
)
```

Arguments

subsampled_matrix	Matrix. The subsampled dissimilarity matrix to check.
folds	Integer. Number of cross-validation folds that will be used.
min_points_per_fold	Integer. Minimum number of data points expected per fold. Default: 5.
min_measurements_per_fold	Integer. Minimum number of measurements per fold. Default: 10.
verbose	Logical. Print detailed diagnostic messages. Default: TRUE.

Details

The function performs these checks:

- **Sufficient points:** At least 2x folds points in total
- **Sufficient measurements:** At least folds x min_measurements_per_fold
- **Adequate points per fold:** Average points per fold $\geq$ min_points_per_fold
- **Adequate measurements per fold:** Average measurements per fold $\geq$ min_measurements_per_fold
- **Not too sparse:** Overall sparsity $< 95\%$

If checks fail, warnings are issued but execution continues. This allows the user to proceed with awareness of potential issues.

Value

A list with check results:

all_checks_passed	Logical. TRUE if all checks passed.
checks	List of individual check results (logical values).
diagnostics	List of diagnostic values calculated.
warnings	Character vector of warning messages (empty if no warnings).

Examples

```
# Create a small matrix
small_mat <- matrix(c(0, 1, 2, 1, 0, 1.5, 2, 1.5, 0), nrow = 3)

# Check with 5 folds (will fail - too few points)
result <- sanity_check_subsample(small_mat, folds = 5)
print(result$all_checks_passed)
print(result$warnings)

# Create a larger connected matrix
n <- 50
coords <- matrix(rnorm(n * 2), ncol = 2)
large_mat <- as.matrix(dist(coords))

# Check with 10 folds (should pass)
result <- sanity_check_subsample(large_mat, folds = 10)
print(result$all_checks_passed)
```

save_plot

Save Plot to File

Description

Saves a plot (ggplot or rgl scene) to file with specified configuration. Supports multiple output formats and configurable dimensions.

Usage

```
save_plot(plot, filename, layout_config = new_layout_config(), output_dir)
```

Arguments

plot	ggplot or rgl scene object to save
filename	Output filename (with or without extension)
layout_config	Layout configuration object controlling output parameters
output_dir	Character. Directory for output files. This argument is required.

Details

Supported file formats:

- PNG: Best for web and general use
- PDF: Best for publication quality vector graphics
- SVG: Best for web vector graphics
- EPS: Best for publication quality vector graphics

The function will:

1. Auto-detect plot type (ggplot or rgl)
2. Use appropriate saving method
3. Apply layout configuration settings
4. Add file extension if not provided

Value

No return value, called for side effects (saves a plot to a file).

Examples

```
# The sole purpose of save_plot is to write a file, so its example must demonstrate this.
# For CRAN tests we wrap the example in \donttest{} to avoid writing files.

# Create a temporary directory for saving all plots
temp_dir <- tempdir()

# --- Example 1: Basic ggplot save ---
# Create sample data with 3 dimensions to support both 2D and 3D plots
data <- data.frame(
  V1 = rnorm(10), V2 = rnorm(10), V3 = rnorm(10), name=1:10,
  antigen = rep(c(0,1), 5), antiserum = rep(c(1,0), 5),
  year = 2000:2009
)
p <- plot_temporal_mapping(data, ndim=2)
save_plot(p, "temporal_plot.png", output_dir = temp_dir)

# --- Example 2: Save with custom layout ---
layout_config <- new_layout_config(
  width = 12,
  height = 8,
  dpi = 600,
  save_format = "pdf"
)
save_plot(p, "high_res_plot.pdf", layout_config, output_dir = temp_dir)

# --- Verify files and clean up ---
list.files(temp_dir)
unlink(temp_dir, recursive = TRUE)
```

scatterplot_fitted_vs_true

Plot Fitted vs. True Dissimilarities

Description

Creates diagnostic plots comparing fitted dissimilarities from a model against the true dissimilarities. It generates both a scatter plot with an identity line and prediction intervals, and a residuals plot.

Usage

```
scatterplot_fitted_vs_true(
  dissimilarity_matrix,
  p_dissimilarity_mat,
  scenario_name,
  ndim,
  save_plot = FALSE,
  output_dir,
  confidence_level = 0.95
)
```

Arguments

dissimilarity_matrix	Matrix of true dissimilarities.
p_dissimilarity_mat	Matrix of predicted/fitted dissimilarities.
scenario_name	Character string for output file naming. Used if save_plot is TRUE.
ndim	Integer number of dimensions used in the model.
save_plot	Logical. Whether to save plots to files. Default: FALSE.
output_dir	Character. Directory for output files. Required if save_plot is TRUE.
confidence_level	Numeric confidence level for prediction intervals (default: 0.95).

Value

A list containing the scatter_plot and residuals_plot ggplot objects.

Examples

```
# Create sample data
true_dist <- matrix(runif(100, 1, 10), 10, 10)
pred_dist <- true_dist + rnorm(100)

# Create plots without saving
plots <- scatterplot_fitted_vs_true(
  dissimilarity_matrix = true_dist,
  p_dissimilarity_mat = pred_dist,
  save_plot = FALSE
)

# You can then display a plot, for instance:
# plots$scatter_plot
```

subsample_dissimilarity_matrix

Subsample Dissimilarity Matrix with Connectivity Guarantee

Description

Randomly samples a subset of points from a dissimilarity matrix, ensuring the resulting submatrix forms a connected graph.

Usage

```
subsample_dissimilarity_matrix(
  dissimilarity_matrix,
  sample_size,
  max_attempts = 5,
  min_completeness = 0.1,
  random_seed = NULL,
  verbose = FALSE,
  preserve_order = FALSE
)
```

Arguments

dissimilarity_matrix	Square symmetric dissimilarity matrix.
sample_size	Integer. Target number of points to sample.
max_attempts	Integer. Maximum number of sampling attempts before giving up. Default: 5
min_completeness	Numeric. Minimum acceptable network completeness (0-1). Default: 0.10. Used for warnings, not hard requirement.
random_seed	Integer or NULL. If provided, sets the random seed for reproducibility.
verbose	Logical. Print progress messages. Default: FALSE.
preserve_order	Logical. If TRUE, the selected indices are sorted to maintain the original row/column order of the input matrix. If FALSE (default), indices are returned in random order as sampled. Set to TRUE when downstream analyses depend on specific point ordering.

Details

The function performs the following steps:

1. Validates inputs and checks if subsampling is necessary
2. Attempts to sample sample_size points randomly

3. Checks if the subsample forms a connected graph using [check_matrix_connectivity](#)
4. If not connected, retries
5. Returns the connected subsample or stops with an error if unsuccessful

Why connectivity matters: A disconnected graph has isolated groups of points with no observed dissimilarities between groups. The algorithm cannot determine the relative positions of disconnected components, leading to arbitrary and meaningless results.

Value

A list containing:

subsampled_matrix	Matrix. The subsampled dissimilarity matrix.
selected_indices	Integer vector. Indices of selected points from the original matrix.
selected_names	Character vector or NULL. Names of selected points (if original matrix had row/column names).
is_connected	Logical. Whether the final subsample is connected.
n_components	Integer. Number of connected components in final subsample.
completeness	Numeric. Network completeness of final subsample (0-1).
attempt_number	Integer. Which attempt succeeded.

See Also

[check_matrix_connectivity](#) for connectivity checking,

Examples

```
# Create a well-connected matrix
n <- 100
coords <- matrix(rnorm(n * 3), ncol = 3)
dist_mat <- as.matrix(dist(coords))

# Subsample to 30 points
result <- subsample_dissimilarity_matrix(
  dissimilarity_matrix = dist_mat,
  sample_size = 30,
  random_seed = 123,
  verbose = TRUE
)

print(result$is_connected)
print(dim(result$subsampled_matrix))
```

summary.topolow	<i>Summary method for topolow objects</i>
-----------------	---

Description

Provides a more detailed summary of the optimization results from `euclidean_embedding`, including parameters, convergence, and performance metrics.

Usage

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

Arguments

<code>object</code>	A topolow object returned by <code>euclidean_embedding()</code> .
<code>...</code>	Additional arguments passed to <code>summary</code> (not used).

Value

No return value. This function is called for its side effect of printing a detailed summary to the console.

Examples

```
# Create a simple dissimilarity matrix and run the optimization
dist_mat <- matrix(c(0, 2, 3, 2, 0, 4, 3, 4, 0), nrow=3)
result <- euclidean_embedding(dist_mat, ndim=2, mapping_max_iter=50,
                             k0=1.0, cooling_rate=0.001, c_repulsion=0.1,
                             verbose = FALSE)

# Summarize the result object
summary(result)
```

symmetric_to_nonsymmetric_matrix	<i>Convert distance matrix to assay panel format</i>
----------------------------------	--

Description

Convert distance matrix to assay panel format

Usage

```
symmetric_to_nonsymmetric_matrix(dist_matrix, selected_names)
```

Arguments

dist_matrix Distance matrix
selected_names Names of reference points

Value

A non-symmetric matrix in assay panel format, where rows are test antigens and columns are reference antigens.

table_to_matrix	<i>Convert Table Format Data to Dissimilarity Matrix</i>
-----------------	--

Description

Converts data from table/matrix format (objects as rows, references as columns) to a symmetric dissimilarity matrix. The function creates a matrix where both rows and columns contain the union of all object and reference names.

Usage

table_to_matrix(data, is_similarity = FALSE)

Arguments

data Matrix or data frame where rownames represent objects, columnnames represent references, and cells contain (dis)similarity values.

is_similarity Logical. Whether values are similarities (TRUE) or dissimilarities (FALSE). If TRUE, similarities will be converted to dissimilarities by subtracting from the maximum value per column (reference). Default: FALSE.

Details

The function takes a table where:

- Rows represent objects
- Columns represent references
- Values represent (dis)similarities

It creates a symmetric matrix where both rows and columns contain the union of all object names (row names) and reference names (column names). The original measurements are preserved, and the matrix is made symmetric by filling both (i,j) and (j,i) positions with the same value.

When is_similarity = TRUE, similarities are converted to dissimilarities by subtracting each value from the maximum value in its respective column (reference). Threshold indicators (< or >) are handled and inverted during conversion.

Value

A symmetric matrix of dissimilarities with row and column names corresponding to the union of all object and reference names. NA values represent unmeasured pairs, and the diagonal is set to 0.

Examples

```
# Example with dissimilarity data in table format
dissim_table <- matrix(c(1.2, 2.1, 3.4, 1.8, 2.9, 4.1),
                      nrow = 2, ncol = 3,
                      dimnames = list(c("Obj1", "Obj2"),
                                      c("Ref1", "Ref2", "Ref3")))

mat_dissim <- table_to_matrix(dissim_table, is_similarity = FALSE)

# Example with similarity data (will be converted to dissimilarity)
sim_table <- matrix(c(8.8, 7.9, 6.6, 8.2, 7.1, 5.9),
                  nrow = 2, ncol = 3,
                  dimnames = list(c("Obj1", "Obj2"),
                                  c("Ref1", "Ref2", "Ref3")))

mat_from_sim <- table_to_matrix(sim_table, is_similarity = TRUE)
```

`titers_list_to_matrix` *Convert Long Format Data to Distance Matrix*

Description

Converts a dataset from long format to a symmetric distance matrix. The function handles antigenic cartography data where measurements may exist between antigens and antisera points. Row and column names can be optionally sorted by a time variable.

Usage

```
titers_list_to_matrix(
  data,
  chnames,
  chorder = NULL,
  rnames,
  rorder = NULL,
  values_column,
  rc = FALSE,
  sort = FALSE
)
```

Arguments

<code>data</code>	Data frame in long format
<code>chnames</code>	Character. Name of column holding the challenge point names.
<code>chorder</code>	Character. Optional name of column for challenge point ordering.
<code>rnames</code>	Character. Name of column holding reference point names.
<code>rorder</code>	Character. Optional name of column for reference point ordering.
<code>values_column</code>	Character. Name of column containing distance/difference values. It should be from the nature of "distance" (e.g., antigenic distance or IC50), not "similarity" (e.g., HI Titer.)
<code>rc</code>	Logical. If TRUE, reference points are treated as a subset of challenge points. If FALSE, they are treated as distinct sets. Default is FALSE.
<code>sort</code>	Logical. Whether to sort rows/columns by <code>chorder/rorder</code> . Default FALSE.

Details

The function expects data in long format with at least three columns:

- A column for challenge point names
- A column for reference point names
- A column containing the distance/difference values

Optionally, ordering columns can be provided to sort the output matrix. The `'rc'` parameter determines how to handle shared names between references and challenges.

Value

A symmetric matrix of distances with row and column names corresponding to the unique points in the data. NA values represent unmeasured pairs.

Examples

```
data <- data.frame(
  antigen = c("A", "B", "A"),
  serum = c("X", "X", "Y"),
  distance = c(2.5, 1.8, 3.0),
  year = c(2000, 2001, 2000)
)

# Basic conversion
mat <- titers_list_to_matrix(data,
  chnames = "antigen",
  rnames = "serum",
  values_column = "distance")

# With sorting by year
mat_sorted <- titers_list_to_matrix(data,
  chnames = "antigen",
  chorder = "year",
```

```
rnames = "serum",  
rorder = "year",  
values_column = "distance",  
sort = TRUE)
```

`weighted_kde`*Weighted Kernel Density Estimation*

Description

Performs weighted kernel density estimation for univariate data. This is useful for analyzing parameter distributions where each sample has an associated importance weight (e.g., a likelihood).

Usage

```
weighted_kde(x, weights, n = 512, from = NULL, to = NULL)
```

Arguments

<code>x</code>	A numeric vector of samples.
<code>weights</code>	A numeric vector of weights corresponding to each sample in <code>x</code> .
<code>n</code>	The integer number of points at which to evaluate the density.
<code>from, to</code>	The range over which to evaluate the density.

Value

A list containing the evaluation points (`x`) and the estimated density values (`y`).

Index

* datasets

- denv_data, [14](#)
- example_positions, [23](#)
- h3n2_data, [25](#)
- hiv_titers, [26](#)
- hiv_viruses, [26](#)

adaptive_MC_sampling, [71](#)

analyze_network_structure, [4](#), [9](#), [69](#)

c25 (color_palettes), [11](#)

calculate_diagnostics, [5](#)

calculate_prediction_interval, [6](#)

calculate_weighted_marginals, [7](#)

check_gaussian_convergence, [8](#), [45](#)

check_matrix_connectivity, [9](#), [69](#), [78](#)

clean_data, [10](#)

color_palettes, [11](#)

coordinates_to_matrix, [11](#)

create_cv_folds, [11](#)

create_diagnostic_report, [13](#)

create_topolow_map(), [18](#), [19](#)

denv_data, [14](#)

error_calculator_comparison, [15](#)

error_calculator_comparison(), [19](#)

euclidean_embedding, [16](#), [29](#)

euclidean_embedding(), [22](#)

Euclidify, [20](#)

example_positions, [23](#)

extract_numeric_values, [24](#)

ggsave_white_bg, [25](#)

h3n2_data, [25](#)

hiv_titers, [26](#)

hiv_viruses, [26](#)

initial_parameter_optimization, [21](#), [27](#), [71](#)

list_to_matrix, [30](#)

log_transform_parameters, [32](#)

make_interactive, [33](#), [47](#)

new_aesthetic_config, [35](#), [50](#), [60](#)

new_annotation_config, [36](#), [50](#), [60](#)

new_dim_reduction_config, [38](#), [50](#), [60](#)

new_layout_config, [39](#), [50](#), [60](#)

parameter_sensitivity_analysis, [40](#)

plot.parameter_sensitivity, [42](#)

plot.profile_likelihood, [43](#)

plot.topolow_convergence, [44](#)

plot.topolow_diagnostics, [45](#)

plot_3d_mapping, [46](#), [50](#), [60](#)

plot_cluster_mapping, [47](#), [48](#), [60](#)

plot_euclidify_diagnostics, [51](#)

plot_ll_improvement, [53](#)

plot_mcmc_diagnostics, [53](#)

plot_network_structure, [55](#)

plot_performance_trace, [56](#)

plot_temporal_mapping, [47](#), [50](#), [58](#)

print.parameter_sensitivity, [61](#)

print.profile_likelihood, [61](#)

print.topolow, [62](#)

print.topolow_convergence, [63](#)

print.topolow_diagnostics, [63](#)

process_antigenic_data, [64](#)

profile_likelihood, [66](#)

prune_sparse_matrix, [68](#)

run_adaptive_sampling, [70](#)

sanity_check_subsample, [72](#)

save_plot, [74](#)

scatterplot_fitted_vs_true, [75](#)

scatterplot_fitted_vs_true(), [19](#)

stats::runif(), [18](#)

subsample_dissimilarity_matrix, [29](#), [77](#)

summary.topolow, [79](#)

`symmetric_to_nonsymmetric_matrix`, [79](#)

`table_to_matrix`, [80](#)

`titers_list_to_matrix`, [81](#)

`weighted_kde`, [83](#)