

Package ‘mathmodels’

July 23, 2025

Type Package

Title Comprehensive Mathematical Modeling Algorithms in R

Version 0.0.5

Author Jingxin Zhang

Maintainer Jingxin zhang <zhjx_19@163.com>

Description A versatile R package for mathematical modeling, developed as a companion to “Mathematical Modeling: Algorithms and Programming Implementation” (China Machine Press). The package implements algorithms across differential and difference equations, statistical analysis, optimization, evaluation, and prediction. Currently, it focuses on evaluation algorithms, including indicator data preprocessing (e.g., standardization, rescaling), subjective and objective weighting methods (e.g., AHP, entropy weighting, CRITIC, PCA weighting) and weight combination, comprehensive evaluation techniques (e.g., TOPSIS, fuzzy comprehensive evaluation, Rank Sum Ratio, DEA), inequality measures (e.g., Gini and Theil indices), and grey prediction models (e.g., GM(1,1), GM(1,N), Verhulst). Designed for researchers and practitioners in mathematical modeling.

License AGPL (>= 3)

URL <https://github.com/zhjx19/mathmodels>

BugReports <https://github.com/zhjx19/mathmodels/issues>

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports dplyr (>= 1.0.0),
tidyr (>= 1.1.0),
ggplot2 (>= 3.5.0)

Depends R (>= 4.1)

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

VignetteBuilder knitr, quarto

Config/testthat/edition 3

Contents

AHP	2
combine_preds	3
combine_weights	3
compute_mf	4
critic_weight	5
cv_weight	6
DEA	7
defuzzify	11
entropy_weight	12
fuzzy_eval	13
grey_analysis	14
grey_models	15
inequality	16
membership	18
pca_weight	21
preprocess	22
rank_sum_ratio	25
system_evaluation	26
topsis	27
water_quality	28
Index	30

AHP

AHP: Analytic Hierarchy Process

Description

AHP is a multi-criteria decision analysis method developed by Saaty, which can also be used to determine indicator weights.

Usage

```
AHP(A)
```

Arguments

A a numeric matrix, i.e. pairwise comparison matrix

Value

a list object that contains: w (Weight vector), CR (Consistency ratio), Lmax (Maximum eigenvalue), CI (Consistency index)

Examples

```
A = matrix(c(1, 1/2, 4, 3, 3,
              2, 1, 7, 5, 5,
              1/4, 1/7, 1, 1/2, 1/3,
              1/3, 1/5, 2, 1, 1,
              1/3, 1/5, 3, 1, 1), byrow = TRUE, nrow = 5)
AHP(A)
```

`combine_preds`*Combine Multiple Prediction Results*

Description

Combines multiple prediction results (e.g., from grey prediction, time series, or machine learning models) into a single prediction using a similarity-based weighting approach, improving prediction accuracy.

Usage

```
combine_preds(x)
```

Arguments

`x` Numeric vector, prediction results to be combined (length ≥ 2).

Details

The function combines prediction results by constructing a similarity matrix based on cosine transformation of pairwise differences. Weights are derived from the principal eigenvector of the similarity matrix, ensuring predictions closer to each other have higher influence. For two predictions, equal weights (0.5, 0.5) are used. If all predictions are identical, equal weights are assigned. Compatible with the `mathmodels` package for enhancing prediction models, including grey prediction, time series, or ensemble machine learning.

Value

A list with two elements:

- `a`: Numeric, the combined prediction value.
- `w`: Numeric vector, weights for each prediction in `x`, summing to 1.

Examples

```
# Example: Combine three prediction results
preds = c(100, 102, 98) # E.g., from grey prediction, ARIMA, or ML models
combine_preds(preds)
```

`combine_weights`*Combine Subjective and Objective Weights*

Description

Combines subjective and objective weights using linear, multiplicative, or game theory-based methods (geometric mean or linear system).

Usage

```
combine_weights(w_subj, w_obj, type = "linear", alpha = 0.5)
```

Arguments

w_subj	Numeric vector of subjective weights.
w_obj	Numeric vector of objective weights.
type	Character string specifying the combination method: "linear", "multiplicative", "game", or "game_linear".
alpha	Numeric value between 0 and 1, used only for the linear method to weight subjective weights. Defaults to 0.5.

Details

The function supports four methods:

- Linear: Combines weights as $\alpha * w_{\text{subj}} + (1 - \alpha) * w_{\text{obj}}$.
- Multiplicative: Combines weights as $w_{\text{subj}} * w_{\text{obj}}$, requiring positive weights.
- Game: Uses the geometric mean ($\sqrt{w_{\text{subj}} * w_{\text{obj}}}$) to balance weights.
- Game_linear: Uses a game-theoretic approach by solving a linear system based on the cross-product of weights.

Value

A numeric vector of combined weights, normalized to sum to 1.

Examples

```
w_subj = c(0.4, 0.3, 0.2, 0.1)
w_obj = c(0.25, 0.2, 0.3, 0.25)
combine_weights(w_subj, w_obj, type = "linear", alpha = 0.6)
combine_weights(w_subj, w_obj, type = "multiplicative")
combine_weights(w_subj, w_obj, type = "game")
combine_weights(w_subj, w_obj, type = "game_linear")
```

compute_mf	<i>Compute fuzzy membership vector and return corresponding membership functions.</i>
------------	---

Description

compute_mf transforms a single indicator value into a fuzzy membership vector, where each element represents the degree of membership to a specific evaluation level. compute_mf_funs returns the list of membership functions for visualization purposes.

Usage

```
compute_mf_funs(thresholds)

compute_mf(x, thresholds)
```

Arguments

- thresholds** A numeric vector containing at least two threshold values that define the boundaries between evaluation levels.
- x** A numeric scalar representing the value of an indicator.

Value

A list with two elements:

mv A numeric vector, membership degrees to each level.

mfs A list of functions, one per level, for plotting membership functions.

Examples

```
# Example: SO2 concentration = 0.07, thresholds = c(0.05, 0.15, 0.25, 0.5)
th = c(0.05, 0.15, 0.25, 0.5)
compute_mf(0.07, th)

## Not run:
mfs = compute_mf_funs(th)
plots = lapply(mfs, \(x) plot_mf(x, xlim = c(0, 0.6)))
gridExtra::grid.arrange(grobs = plots, nrow = 2)

## End(Not run)
```

critic_weight

CRITIC Weight Method

Description

Computes objective weights of indicators and scores of samples using the CRITIC method. The method considers both the contrast intensity (e.g., standard deviation or entropy) and conflict among indicators (based on correlation) to determine indicator importance. This version supports different methods for contrast intensity and correlation types.

Usage

```
critic_weight(
  X,
  index = NULL,
  method = "std",
  cor_method = "pearson",
  epsilon = 0.002
)
```

Arguments

X	A numeric data frame or matrix where rows represent samples (observations) and columns represent indicators (variables).
index	A character vector indicating the direction of each indicator: Use "+" for positive indicators (higher is better), "-" for negative indicators (lower is better), and NA for already normalized indicators (no rescaling will be applied). If <code>`index = NULL`</code> (default), all indicators are treated as <code>`NA`</code> , meaning no normalization.
method	Character scalar; specifies the method used to compute contrast intensity. Options: "std" (standard deviation, default), or "entropy" (based on information redundancy).
cor_method	Character scalar; specifies the method for computing correlations. Options: "pearson" (default), "spearman", or "kendall".
epsilon	A small constant used to replace exact 0s and 1s in the data to prevent log(0) errors. Default is 0.002. Only used when method = "entropy".

Value

A list containing:	
w	Numeric vector of weights for each indicator.
s	Numeric vector of scores for each sample (row), scaled by 100.

Examples

```
# Example: Using CRITIC method on a simple dataset
X = data.frame(
  x1 = c(3, 5, 2, 7),
  x2 = c(10, 20, 15, 25)
)
index = c("+", "-")
critic_weight(X, index)
critic_weight(X, index, method = "entropy")
```

cv_weight	<i>Coefficient of Variation Weighting</i>
-----------	---

Description

Computes weights for indicators using the Coefficient of Variation (CV) method. Weights are derived by normalizing the CV (standard deviation divided by mean) for each indicator.

Usage

```
cv_weight(X)
```

Arguments

data	Numeric matrix or data frame with positive indicator data.
------	--

Details

The `cv_weight` function calculates weights using the CV method. For each column in `data`, the CV is computed as the standard deviation divided by the mean. Weights are obtained by normalizing the CVs to sum to 1. This lightweight implementation uses base R and assumes all columns are numeric indicators.

Value

Numeric vector of weights for the indicators, summing to 1.

Examples

```
X = data.frame(x1 = c(10, 20, 15), x2 = c(5, 10, 8))
cv_weight(X)
```

DEA

Data Envelopment Analysis and Malmquist Productivity Index

Description

Computes standard and super-efficiency Data Envelopment Analysis (DEA) models (CCR, BCC, and slacks-based), including efficiency scores, slacks, lambdas, targets, returns to scale, and references, with support for undesirable outputs. Additionally, calculates the Malmquist productivity index to measure productivity changes over time, including efficiency change (EC), technical change (TC), and other decomposition components.

Usage

```
basic_DEA(
  data,
  inputs,
  outputs,
  ud_outputs = NULL,
  orientation = "io",
  rts = "vrs"
)
```

```
super_DEA(
  data,
  inputs,
  outputs,
  ud_outputs = NULL,
  orientation = "io",
  rts = "vrs"
)
```

```
basic_SBM(
  data,
  inputs,
  outputs,
```

```

    ud_outputs = NULL,
    orientation = "io",
    rts = "vrs"
)

super_SBM(data, inputs, outputs, orientation = "io", rts = "vrs")

malmquist(
  data,
  period,
  inputs,
  outputs,
  orientation = "oo",
  rts = "vrs",
  type1 = "glob",
  type2 = "rd"
)

```

Arguments

data	For DEA functions (basic_DEA, super_DEA, basic_SBM, super_SBM): A data frame where the first column contains DMU (Decision Making Unit) names/identifiers, and subsequent columns are input/output variables. For malmquist: A long-format data frame with a period column, DMU identifiers, and input/output variables.
inputs	A numeric vector of column indices or a character vector of column names indicating input variables.
outputs	A numeric vector of column indices or a character vector of column names indicating (desirable) output variables.
ud_outputs	Optional. A numeric vector indicating the position of undesirable outputs within the outputs parameter. Defaults to NULL. Not applicable for super_SBM or malmquist.
orientation	Character string. Model orientation: "io" (input-oriented, default for DEA functions) or "oo" (output-oriented, default for malmquist).
rts	Character string. Returns to scale assumption: "vrs" (variable returns to scale, default) or "crs" (constant returns to scale).
period	For malmquist only: A numeric or character index/name indicating the column in data containing time periods.
type1	For malmquist only: Reference technology for Malmquist index: "cont" (contemporary), "seq" (sequential), or "glob" (global, default).
type2	For malmquist only: Decomposition method for Malmquist index: "fgnz" (Färe et al., 1994), "rd" (Ray and Desli, 1997, default), "gl" (generalized), or "bias" (biased).

Details

This package provides a unified interface for computing efficiency scores and productivity changes using the **deaR** package. It includes five functions: `basic_DEA`, `super_DEA`, `basic_SBM`, `super_SBM`, and `malmquist`, each tailored to specific DEA or productivity analysis models.

- **DEA Models:**

- `basic_DEA`: Implements standard radial DEA models (CCR for CRS, BCC for VRS) as described by Charnes et al. (1978) and Banker et al. (1984), optimizing radial efficiency (input contraction or output expansion).
- `super_DEA`: Computes super-efficiency radial DEA, excluding the evaluated DMU from the reference set to allow efficiency scores beyond 1 (output-oriented) or below 1 (input-oriented) for efficient DMUs (Andersen & Petersen, 1993).
- `basic_SBM`: Implements standard Slacks-Based Measure (SBM) models (Tone, 2001), optimizing input and output slacks for a non-radial efficiency measure.
- `super_SBM`: Combines SBM with super-efficiency properties, excluding the evaluated DMU from the reference set. Note: `super_SBM` does not support undesirable outputs.
- **Malmquist Productivity Index:**
 - `malmquist`: Calculates the Malmquist productivity index to measure productivity changes over time, decomposing it into efficiency change (EC) and technical change (TC), with optional further decomposition into pure efficiency change (PECH) and scale efficiency change (SECH) based on `type2` (Färe et al., 1994; Ray & Desli, 1997). If EC is unavailable (e.g., under `rts = "vrs"` and `type1 = "glob"`), it is computed as $ec = mi / tc$ to ensure consistent output.

Orientation:

- Input-oriented ("`io`"): Minimizes inputs while maintaining outputs. Efficiency scores are in $(0, 1]$ ($\theta \leq 1$ for radial models, ρ or $\delta \leq 1$ for SBM models).
- Output-oriented ("`oo`"): Maximizes outputs for given inputs. Efficiency scores are in $(0, 1]$. Radial models output $\eta \geq 1$, converted to $1/\eta$; SBM models output $1/\rho^*$ or $1/\delta$.

Returns to Scale (RTS):

- CRS ("`crs`"): Assumes constant returns to scale, suitable for long-run analysis.
- VRS ("`vrs`"): Allows variable returns to scale, with increasing ("`irs`") or decreasing ("`drs`") returns determined by the sum of intensity variables (λ).

Undesirable Outputs:

- Supported in `basic_DEA`, `super_DEA`, and `basic_SBM` using directional distance functions (DDF) with direction vector $(g_y, -g_b)$ to increase desirable outputs and decrease undesirable outputs (Färe & Grosskopf, 2004). Not supported in `super_SBM` or `malmquist`.

Malmquist-Specific Parameters:

- `type1`: Defines the reference technology for the Malmquist index:
 - "`cont`": Contemporary technology, using each period's frontier.
 - "`seq`": Sequential technology, incorporating all prior periods.
 - "`glob`": Global technology, using a single frontier across all periods.
- `type2`: Specifies the decomposition method:
 - "`fgnz`": Färe et al. (1994) decomposition.
 - "`rd`": Ray and Desli (1997) decomposition.
 - "`gl`": Generalized decomposition.
 - "`bias`": Bias-corrected decomposition.

Handling NA Values:

- Super-efficiency models (`super_DEA`, `super_SBM`) may return NA for efficient DMUs due to infeasible linear programming solutions, especially under VRS (Andersen & Petersen, 1993). Users can replace NA with standard efficiency scores or exclude affected DMUs.
- For `malmquist`, if `ec` is NULL (e.g., under `rts = "vrs"` and `type1 = "glob"`), it is computed as `mi / tc` to ensure a complete result.

The package leverages **deaR** for robust computation, handling zero values internally and ensuring compatibility with input/output specifications. Efficiency scores are standardized to (0, 1] for DEA models, and Malmquist results are formatted as a data frame for easy analysis.

Value

For DEA functions (`basic_DEA`, `super_DEA`, `basic_SBM`, `super_SBM`): A list containing six elements:

- `efficiencies`: A named numeric vector of efficiency scores for each DMU, standardized to (0, 1] for both input- and output-oriented models.
- `slacks`: A data frame or matrix containing slack values for inputs and outputs (including undesirable outputs, if specified).
- `lambdas`: A matrix or data frame of intensity variables (λ), representing the contribution of reference DMUs to the efficiency frontier (self-excluded in super-efficiency models).
- `targets`: A data frame or matrix of efficient target values for inputs and outputs, adjusted for undesirable outputs in DDF models.
- `returns`: A character vector indicating returns-to-scale status for each DMU: `"crs"` (constant), `"irs"` (increasing), or `"drs"` (decreasing).
- `references`: A matrix or data frame listing reference DMUs (peers) contributing to the efficiency frontier ($\lambda > 0$).

For `malmquist`: A data frame containing:

- `Period`: Time period transitions (e.g., `"t~t+1"`).
- `DMU`: Decision Making Unit identifiers.
- `mi`: Malmquist productivity index, measuring total productivity change.
- `ec`: Efficiency change (EC), computed directly or as `mi / tc` if not available from the model.
- `tc`: Technical change (TC), measuring frontier shift.
- `pech`: Pure efficiency change (if applicable, based on decomposition method).
- `sech`: Scale efficiency change (if applicable, based on decomposition method).

Examples

```
# Sample DEA data
data_dea = data.frame(
  DMU = paste0("DMU", 1:5),
  input1 = c(10, 20, 15, 25, 30),
  input2 = c(5, 8, 7, 10, 12),
  output = c(100, 150, 120, 180, 200),
  ud_output = c(10, 15, 12, 20, 25)
)

# Standard DEA
result = basic_DEA(data_dea, inputs = 2:3, outputs = 4)
result$efficiencies
```

```

# DEA with undesirable outputs
result = basic_DEA(data_dea, inputs = 2:3, outputs = 4:5, ud_outputs = 2)
result$efficiencies

# Super-efficiency DEA
result = super_DEA(data_dea, inputs = 2:3, outputs = 4)
result$efficiencies

# Standard SBM
result = basic_SBM(data_dea, inputs = 2:3, outputs = 4)
result$efficiencies

# Super-efficiency SBM
result = super_SBM(data_dea, inputs = 2:3, outputs = 4)
result$efficiencies

# Sample Malmquist data (long format)
data_malm = data.frame(
  DMU = rep(paste0("DMU", 1:5), 3),
  Period = rep(1:3, each = 5),
  input1 = c(10, 20, 15, 25, 30, 12, 22, 17, 27, 32, 14, 24, 19, 29, 34),
  input2 = c(5, 8, 7, 10, 12, 6, 9, 8, 11, 13, 7, 10, 9, 12, 14),
  output = c(100, 150, 120, 180, 200, 110, 160, 130, 190, 210, 120, 170, 140, 200, 220)
)
malmquist(data_malm, period = 2, inputs = 3:4, outputs = 5)

```

defuzzify

*Defuzzification Methods for Fuzzy Comprehensive Evaluation***Description**

Implements defuzzification methods for fuzzy evaluation vectors, including weighted average and maximum membership methods.

Usage

```
defuzzify(mu, scores, method = "weighted_average")
```

Arguments

mu	Numeric vector, membership degrees for evaluation levels, in 0 , 1 .
scores	Numeric vector, scores corresponding to each evaluation level (e.g., c(100, 80, 60, 40) for "Excellent", "Good", "Fair", "Poor").
method	Character, defuzzification method: "weighted_average", "max_membership", "centroid".

Value

Numeric, defuzzified output value.

Examples

```
# Example: Defuzzify fuzzy evaluation vectors for three schemes
mu = c(0.318, 0.351, 0.203, 0.128)
scores = c(30, 60, 75, 90) # Scores for "Poor", "Fair", "Good", "Excellent"
defuzzify(mu, scores, method = "weighted_average")
defuzzify(mu, scores, method = "max_membership")
defuzzify(mu, scores, method = "centroid")
```

entropy_weight	<i>Entropy Weight Method</i>
----------------	------------------------------

Description

Computes the weights of indicators and scores of samples based on the entropy method. This method objectively determines the importance of each indicator according to the amount of information it contains.

Usage

```
entropy_weight(X, index = NULL, epsilon = 0.002)
```

Arguments

X	A numeric data frame or matrix where rows represent samples (observations) and columns represent indicators (variables).
index	A character vector indicating the direction of each indicator. Use "+" for positive indicators (higher is better), "-" for negative indicators (lower is better), and NA for already normalized indicators (no rescaling will be applied, but minor adjustments will still be made to avoid log(0) errors). If index = NULL (default), all indicators are treated as NA, meaning no normalization or rescaling is performed, but a small adjustment is still applied to prevent log(0) errors.
epsilon	A small constant used to replace exact 0s and 1s in the data to prevent log(0) errors. Default is 0.002.

Value

A list containing:

w	Numeric vector of weights for each indicator.
s	Numeric vector of scores for each sample (row), scaled by 100.

Examples

```
X = data.frame(
  x1 = c(3, 5, 2, 7),
  x2 = c(10, 20, 15, 25)
)
index = c("+", "-")
entropy_weight(X, index)
```

fuzzy_eval

*Fuzzy Comprehensive Evaluation***Description**

Performs fuzzy comprehensive evaluation using different fuzzy composition operators to combine factor weights with a fuzzy evaluation matrix. Suitable for multi-criteria decision analysis with weights from methods like AHP, entropy, CRITIC, CV, or PCA.

Usage

```
fuzzy_eval(w, R, type)
```

Arguments

w	Numeric vector, factor weights (e.g., from <code>combine_weights_linear</code>).
R	Numeric matrix, fuzzy evaluation matrix with columns as factors and rows as evaluation grades. Values should be in 0 , 1 .
type	Integer or character (1-5), specifying the fuzzy composition operator: <ul style="list-style-type: none"> • 1: Min-max (main factor decisive). • 2: Product-max (main factor prominent). • 3: Weighted sum (additive average). • 4: Bounded sum of mins (min-sum bounded). • 5: Normalized min-sum (balanced average).

Details

The function computes a fuzzy comprehensive evaluation vector B based on the weight vector w and fuzzy evaluation matrix R. Five composition operators are supported:

- Type 1 (min-max): $\max(\min(w, R[j,]))$, emphasizes the main factor.
- Type 2 (product-max): $\max(w * R[j,])$, highlights the main factor.
- Type 3 (weighted sum): $\text{sum}(w * R[j,])$, additive average.
- Type 4 (bounded sum): $\min(1, \text{sum}(\min(w, R[j,])))$, bounds the sum of mins.
- Type 5 (normalized min-sum): $\text{sum}(\min(w, R[j,] / \text{sum}(R[j,])))$, balanced average.

The output B is normalized to sum to 1. If the sum is zero, an error is thrown. Uses base R for lightweight implementation.

Value

A numeric vector of normalized comprehensive evaluation results, summing to 1.

Examples

```
w = c(0.3, 0.3, 0.3, 0.1) # weights (e.g., from AHP or entropy)

# fuzzy evaluation matrix (3 grades for 4 factors)
R = matrix(c(0.8, 0.7, 0.6, 0.7,
             0.1, 0.2, 0.2, 0.1,
             0.1, 0.1, 0.2, 0.2), nrow = 3, byrow = TRUE)
# Apply fuzzy comprehensive evaluation
fuzzy_eval(w, R, type = 3) # Weighted sum
```

grey_analysis

Grey Relational Analysis Functions

Description

A collection of functions for performing grey relational analysis, including calculation of grey correlation degree and evaluation based on grey correlation. These functions are designed for decision-making and data analysis by measuring the relational degree between sequences.

Usage

```
grey_corr(ref, cmp, rho = 0.5, w = NULL)

grey_corr_topsis(X, w, index = NULL, rho = 0.5)
```

Arguments

ref	Numeric vector, the reference sequence for <code>grey_corr</code> .
cmp	Numeric matrix or data frame, the comparison sequences for <code>grey_corr</code> .
rho	Numeric scalar, the distinguishing coefficient (default = 0.5).
w	Numeric vector, weights for weighted correlation (default = equal weights).
X	Numeric matrix or data frame, the decision matrix for <code>grey_corr_topsis</code> .
index	Character vector indicating indicator direction: Use "+" for positive indicators (higher is better), "-" for negative indicators (lower is better), and NA for already rescaled indicators (no rescaling will be applied). If <code>index = NULL</code> (default), all indicators are treated as NA, meaning no rescaling is performed.

Details

These functions implement grey relational analysis for evaluating relationships between sequences or decision alternatives:

grey_corr Computes the grey correlation degree between a reference sequence (ref) and comparison sequences (cmp) using the distinguishing coefficient (rho) and optional weights (w).

grey_corr_topsis Evaluates a decision matrix (X) by normalizing it, applying weights (w), computing grey correlation with the ideal sequence. Direction of indicators can be specified via index.

Value

grey_corr Returns a numeric vector of grey correlation degrees for each comparison sequence.

grey_corr_topsis Returns a numeric vector of relative closeness (grey correlation degrees).

Examples

```
# Grey correlation degree
ref = 1:3
cmp = data.frame(x1 = c(1, 2, 4), x2 = c(2, 3, 5))
grey_corr(ref, cmp, rho = 0.5)

# Grey correlation evaluation#'
w = c(0.4, 0.6)
idx = c("+", "-")
grey_corr_topsis(cmp, w, idx, rho = 0.5)
```

grey_models

Grey Prediction Models

Description

Implements grey prediction models for time series forecasting: GM11 applies the GM(1,1) model with level ratio test. GM1N applies the GM(1,N) model with multiple related factors. DGM21 applies the DGM(2,1) model for second-order dynamics. verhulst applies the Verhulst model for logistic growth.

Usage

GM11(X)

GM1N(dat, new_data = NULL)

DGM21(X)

verhulst(X)

Arguments

X For GM11, DGM21, verhulst: Numeric vector of original time series data.

dat For GM1N: Data frame or matrix, last column is characteristic series, others are related factors.

Value

For GM11: List with fitted values (fitted), next prediction (pnext), prediction function (f), matrix (mat), parameters (u), level ratios (lambda), and range (rng). For GM1N: List with fitted values (fitted), posterior variance ratio (C), small error probability (P), and prediction function (f). For DGM21, verhulst: List with fitted values (fitted), next prediction (pnext), prediction function (f), matrix (mat), and parameters (u).

Examples

```
# Sample time series for GM11, DGM21, Verhulst
x = c(100, 120, 145, 175, 210)

# GM11
result = GM11(x)
result$fitted      # Fitted values
result$pnext       # Next prediction
result$f(6:8)      # Predict next 3 periods

# DGM21
x = c(2.874, 3.278, 3.39, 3.679, 3.77, 3.8)
result = DGM21(x)
result$fitted      # Fitted values
result$pnext       # Next prediction
result$f(6:8)      # Predict next 3 periods

# Verhulst
x = c(4.93, 2.33, 3.87, 4.35, 6.63, 7.15, 5.37, 6.39, 7.81, 8.35)
result = verhulst(x)
result$fitted      # Fitted values
result$pnext       # Next prediction
result$f(6:8)      # Predict next 3 periods

# Sample data for GM1N
data = data.frame(
  factor1 = c(50, 55, 60, 65, 70),
  factor2 = c(20, 22, 25, 28, 30),
  output = c(100, 120, 145, 175, 210)
)
result = GM1N(data)
result$fitted
```

inequality

Inequality Indices

Description

Computes inequality indices for individual or grouped data: `gini0` calculates the Gini coefficient for individual sample data. `gini` calculates the Gini coefficient for grouped data using income and population shares. `theil0` calculates the Theil index for individual sample data. `theil` calculates the Theil index for grouped average data. `theil0_g` calculates the Theil index and decomposition for grouped sample data. `theil_g` calculates the Theil index and decomposition for grouped average data. `theil_g2_cross` calculates the Theil index and decomposition for two-level cross-grouped average data. `theil_g2_nest` calculates the Theil index and decomposition for two-level nested grouped average data.

For `theil`, `theil_g`, `theil_g2_cross`, `theil_g2_nest`: Name of population variable (character).

Usage

```
gini0(y)
```



```

gini(y, pop)

theil0(y)

theil(y, pop)

theil0_g(data, group, y)

theil_g(data, group, y, pop)

theil_g2_cross(data, group1, group2, y, pop)

theil_g2_nest(data, group1, group2, y, pop)

```

Arguments

<code>y</code>	For <code>gini0</code> , <code>gini</code> , <code>theil0</code> : Numeric vector of individual incomes.
<code>pop</code>	For <code>gini</code> : Numeric vector of group populations or population shares. For <code>theil</code> , <code>theil0_g</code> , <code>theil_g</code> , <code>theil_g2_cross</code> , <code>theil_g2_nest</code> : Name of income variable (character).
<code>data</code>	For <code>theil0_g</code> , <code>theil_g</code> , <code>theil_g2_cross</code> , <code>theil_g2_nest</code> : Data frame containing variables.
<code>group</code>	For <code>theil0_g</code> , <code>theil_g</code> : Name of grouping variable (e.g., province).
<code>group1</code>	For <code>theil_g2_cross</code> , <code>theil_g2_nest</code> : Name of first grouping variable (e.g., region or province).
<code>group2</code>	For <code>theil_g2_cross</code> , <code>theil_g2_nest</code> : Name of second grouping variable (e.g., type or city).

Value

For `gini0`, `gini`: Numeric Gini coefficient (0 to 1). For `theil0`, `theil`: Numeric Theil index. For `theil0_g`, `theil_g`, `theil_g2_cross`, `theil_g2_nest`: List with two vectors:

- `theil`: `theil` (Theil index and its decomposition),
- `ratio`: `ratio` (contribution rates of each component).

Examples

```

# Sample data
income = c(10, 20, 30, 40, 100)
pop = c(100, 150, 200, 250, 300)

# Gini coefficient (individual data)
gini0(income)

# Gini coefficient (grouped data)
gini(income, pop)

# Theil index (individual sample)
data = data.frame(g = c("A", "A", rep("B", 10), rep("A", 6)),
                  y = c(10, 10, rep(8, 4), rep(6, 6), rep(4, 4), 2, 2))
theil0(data$y)

```

```

# Theil index (grouped average)
data2 = data |> dplyr::count(g, y, name = "pop")
theil(data2$y, data2$pop)

# Theil index with grouping (sample data)
theil0_g(data, "g", "y")

# Theil index with grouping (average data)
theil_g(data2, "g", "y", "pop")

# Theil index with two-level cross-grouping
data3 = data.frame(
  industry = c("A", "A", "A", "A", "B", "B", "B", "B"),
  area = c("East", "East", "West", "West", "East", "East", "West", "West"),
  province = c("Shanghai", "Beijing", "Sichuan", "Yunnan", "Shanghai", "Beijing", "Sichuan", "Yunnan"),
  avg_wage = c(500, 400, 80, 60, 300, 250, 50, 40),
  emp_num = c(100, 80, 90, 70, 120, 100, 80, 60)
)
theil_g2_cross(data3, "industry", "area", "avg_wage", "emp_num")

# Theil index with two-level nested grouping
data4 = data.frame(
  province = c("A", "A", "A", "A", "B", "B"),
  city = c("A1", "A1", "A2", "A2", "B1", "B1"),
  industry = c("Manu", "Serv", "Manu", "Serv", "Manu", "Serv"),
  y = c(50000, 45000, 60000, 55000, 70000, 65000),
  pop = c(10000, 8000, 15000, 12000, 10000, 8000)
)
theil_g2_nest(data4, "province", "city", "y", "pop")

```

membership

*Membership Functions for Fuzzy Logic***Description**

A collection of functions to compute membership values for various fuzzy sets, including triangular, trapezoidal, Gaussian, generalized bell, two-parameter Gaussian, sigmoid, difference of sigmoids, product of sigmoids, Z-shaped, PI-shaped, and S-shaped membership functions. Includes a function to visualize membership functions using ggplot2. These are designed for evaluation models in mathematical modeling, compatible with fuzzy_eval in the mathmodels package.

Usage

```

tri_mf(x, params)

trap_mf(x, params)

gauss_mf(x, params)

gbell_mf(x, params)

gauss2mf(x, params)

```

```

sigmoid_mf(x, params)

dsigmoid_mf(x, params)

psigmoid_mf(x, params)

z_mf(x, params)

pi_mf(x, params)

s_mf(x, params)

plot_mf(mf, xlim = c(0, 10), main = NULL)

```

Arguments

x	Numeric vector, input values for which to compute membership.
params	Numeric vector, parameters defining the membership function: <ul style="list-style-type: none"> • For <code>tri_mf</code>: $c(a, b, c)$, where $a \leq b \leq c$ (left base, peak, right base). • For <code>trap_mf</code>: $c(a, b, c, d)$, where $a \leq b \leq c \leq d$ (left base, left top, right top, right base). • For <code>gauss_mf</code>: $c(\text{sigma}, c)$, where $\text{sigma} > 0$ (spread, center). • For <code>gbell_mf</code>: $c(a, b, c)$, where $a > 0, b > 0$ (width, shape, center). • For <code>gauss2mf</code>: $c(s1, c1, s2, c2)$, where $s1 > 0, s2 > 0$ (left spread, left center, right spread, right center). • For <code>sigmoid_mf</code>: $c(a, b)$, where $a > 0$ (slope, inflection point). • For <code>dsigmoid_mf</code>: $c(a1, c1, a2, c2)$, where $a1 > 0, a2 > 0$ (slopes and inflection points for two sigmoids). • For <code>psigmoid_mf</code>: $c(a1, c1, a2, c2)$, where $a1 > 0, a2 > 0$ (slopes and inflection points for two sigmoids). • For <code>z_mf</code>: $c(a, b)$, where $a < b$ (left base, right base). • For <code>pi_mf</code>: $c(a, b, c, d)$, where $a < b < c < d$ (left base, left shoulder, right shoulder, right base). • For <code>s_mf</code>: $c(a, b)$, where $a < b$ (left base, right base).
mf	Function, a membership function with fixed parameters (e.g., <code>function(x) tri_mf(x, c(2, 5, 8))</code>).
xlim	Numeric vector of length 2, x-axis limits for plotting (default $c(0, 10)$).
main	Character, plot title (default <code>NULL</code> , no title).

Details

These functions support evaluation models in mathematical modeling:

- `tri_mf`: Triangular membership, linear rise from a to b (peak) and fall to c .
- `trap_mf`: Trapezoidal membership, linear rise from a to b , plateau from b to c , fall to d .
- `gauss_mf`: Gaussian membership, bell-shaped curve centered at c with spread sigma .
- `gbell_mf`: Generalized bell membership, bell-shaped curve with width a , shape b , and center c .

- `gauss2mf`: Two-parameter Gaussian membership, combining two Gaussians with spreads `s1`, `s2` and centers `c1`, `c2`.
- `sigmoid_mf`: Sigmoid membership, S-shaped curve with slope `a` and inflection point `b`.
- `dsigmoid_mf`: Difference of two sigmoids, combining slopes `a1`, `a2` and inflection points `c1`, `c2`.
- `psigmoid_mf`: Product of two sigmoids, combining slopes `a1`, `a2` and inflection points `c1`, `c2`.
- `z_mf`: Z-shaped membership, decreasing from 1 at `a` to 0 at `b`.
- `pi_mf`: PI-shaped membership, rising from `a` to `b`, plateau from `b` to `c`, falling to `d`.
- `s_mf`: S-shaped membership, increasing from 0 at `a` to 1 at `b`.
- `plot_mf`: Plots a membership function over `xlim` using `ggplot2`, suitable for tidyverse workflows.

Membership values can be used to construct fuzzy evaluation matrices for `fuzzy_eval`. Implemented in base R, except `plot_mf`, which requires `ggplot2`.

Value

- For membership functions (`tri_mf`, `trap_mf`, `gauss_mf`, `gbell_mf`, `gauss2mf`, `sigmoid_mf`, `dsigmoid_mf`, `psigmoid_mf`, `z_mf`, `pi_mf`, `s_mf`): A numeric vector of membership values in `0, 1`, same length as `x`.
- For `plot_mf`: A `ggplot2` object, plotting the membership function.

Examples

```
# Define input values
x = 0:10

# Triangular membership
tri_mf(x, params = c(3, 6, 8))

# Trapezoidal membership
trap_mf(x, params = c(1, 5, 7, 8))

# Gaussian membership
gauss_mf(x, params = c(2, 5))

# Generalized bell membership
gbell_mf(x, params = c(2, 4, 6))

# Two-parameter Gaussian membership
gauss2mf(x, params = c(1, 3, 3, 4))

# Sigmoid membership
sigmoid_mf(x, params = c(2, 4))

# Difference of sigmoids membership
dsigmoid_y = dsigmoid_mf(x, params = c(5, 2, 5, 7))

# Product of sigmoids membership
psigmoid_mf(x, params = c(2, 3, -5, 8))

# Z-shaped membership
z_mf(x, params = c(3, 7))
```

```

# PI-shaped membership
pi_mf(x, params = c(1, 4, 5, 10))

# S-shaped membership
s_mf(x, params = c(1, 8))

## Not run:
# Visualize membership functions
plot_mf(\(x) tri_mf(x, c(3, 6, 8)), main = "Triangular MF")
plot_mf(\(x) trap_mf(x, c(1, 5, 7, 8)), main = "Trapezoidal MF")
plot_mf(\(x) gauss_mf(x, c(2, 5)), main = "Gaussian MF")
plot_mf(\(x) gbell_mf(x, c(2, 4, 6)), main = "Generalized Bell MF")
plot_mf(\(x) gauss2mf(x, c(1, 3, 3, 4)), main = "Two-Parameter Gaussian MF")
plot_mf(\(x) sigmoid_mf(x, c(2, 4)), main = "Sigmoid MF")
plot_mf(\(x) dsigmoid_mf(x, c(5, 2, 5, 7)), main = "Difference of Sigmoids MF")
plot_mf(\(x) psigmoid_mf(x, c(2, 3, -5, 8)), main = "Product of Sigmoids MF")
plot_mf(\(x) z_mf(x, c(3, 7)), main = "Z-Shaped MF")
plot_mf(\(x) pi_mf(x, c(1, 4, 5, 10)), main = "PI-Shaped MF")
plot_mf(\(x) s_mf(x, c(1, 8)), main = "S-Shaped MF")

## End(Not run)

```

pca_weight

PCA-Based Weighting Method

Description

Computes indicator weights using Principal Component Analysis (PCA). The method extracts principal components and uses their variance contribution to derive objective weights for indicators. Optionally handles positive/negative directions of indicators, and supports pre-standardized data.

Usage

```
pca_weight(X, index = NULL, nfs = NULL, varimax = TRUE, method = "abs")
```

Arguments

X	A numeric data frame or matrix where rows represent samples and columns represent indicators.
index	A character vector indicating the direction of each indicator. Use "+" for positive indicators (higher is better), "-" for negative indicators (lower is better), and NA for already standardized indicators (no standardization will be applied). If `index = NULL` (default), all indicators are treated as `NA`, meaning no standardization is performed.
nfs	Number of principal components to use; by default, all are used.
method	Weighting Method, "abs" (default, la_{jil}) or "squared" (a_{ji}^2)
varvarimax	Whether to perform Varimax rotation, default is TRUE.

Value

A list containing:

w	Numeric vector of normalized weights for each indicator.
s	Numeric vector of scores for each sample.
lambda	Eigenvalues of principal components (explained variance).
varP	Proportion of variance explained by selected PCs.

Examples

```
# Example: Using PCA to compute indicator weights
ind = c("+", "+", "-", "-")
pca_weight(iris[1:10, 1:4], ind, nfs = 2)
```

preprocess

Preprocessing Functions for Data Normalization and Standardization

Description

A collection of functions to preprocess numeric data, including standardization, L2 norm normalization, Min-Max scaling, centered-type normalization, interval-type normalization, extreme-value-based normalization, initial-value-based normalization, mean-based normalization, and negative-to-positive transformation. These functions transform a numeric vector to a standardized or normalized scale, suitable for various indicator types (positive, negative, centered, interval-based, or extreme-based).

Usage

```
standardize(x, center = TRUE, scale = TRUE)

normalize(x)

rescale(x, type = "+", a = 0, b = 1)

rescale_middle(x, m)

rescale_interval(x, a, b)

rescale_extreme(x, type = "+")

rescale_initial(x, type = "+")

rescale_mean(x)

to_positive(x, type = "minmax")
```

Arguments

<code>x</code>	Numeric vector to be preprocessed.
<code>center</code>	Logical or numeric scalar, passed to <code>base::scale</code> for centering (for <code>standardize</code>). Default is <code>TRUE</code> .
<code>scale</code>	Logical or numeric scalar, passed to <code>base::scale</code> for scaling (for <code>standardize</code>). Default is <code>TRUE</code> .
<code>type</code>	Character scalar specifying the transformation direction or method: " + " Positive direction (larger values are better, for <code>rescale</code> , <code>rescale_extreme</code> and <code>rescale_initial</code>). " - " Negative direction (smaller values are better, for <code>rescale</code> , <code>rescale_extreme</code> and <code>rescale_initial</code>). " minmax " Min-max transformation (for <code>to_positive</code>). " reciprocal " Reciprocal transformation (for <code>to_positive</code>).
<code>a</code>	Numeric scalar, lower bound of the output range or optimal interval (for <code>rescale</code> and <code>rescale_interval</code>).
<code>b</code>	Numeric scalar, upper bound of the output range or optimal interval (for <code>rescale</code> and <code>rescale_interval</code>).
<code>m</code>	Numeric scalar, optimal value for centered-type normalization (for <code>rescale_middle</code>).

Details

These functions support various preprocessing needs in data analysis:

- `standardize`: Applies Z-score standardization (mean = 0, sd = 1), ideal for equalizing variances or normally distributed data.
- `normalize`: Scales the vector to unit length by dividing by its L2 (Euclidean) norm, useful for machine learning or similarity calculations.
- `rescale`: Performs Min-Max scaling to a specified range (default 0, 1), supporting positive or negative indicators.
- `rescale_middle`: Normalizes centered-type indicators, where values closer to an optimal value `m` are better, mapping to 0, 1.
- `rescale_interval`: Normalizes interval-type indicators, where values within `[a, b]` are optimal, mapping to 0, 1.
- `rescale_extreme`: Normalizes using extreme values: $\min(x)/x$ for positive indicators or $x/\max(x)$ for negative indicators, often used in grey relational analysis.
- `rescale_initial`: Normalizes by dividing by the first value ($x/x[1]$ or $x[1]/x$), commonly used in grey relational analysis.
- `rescale_mean`: Normalizes by dividing by the mean ($x/\text{mean}(x)$), commonly used in grey relational analysis.
- `to_positive`: Converts negative indicators to positive using either min-max ($\max(x) - x$) or reciprocal ($1/x$) transformation.

Missing values (NA) are preserved in the output. For `rescale_initial` and `rescale_mean`, the initial value or mean must be non-zero, respectively.

Value

A numeric vector of the same length as `x`, transformed as follows:

- `standardize`: Standardized values (mean = 0, sd = 1).
- `normalize`: L2 norm normalized values (Euclidean norm, unit length).
- `rescale`: Min-Max scaled values in `[a, b]` (default `0, 1`).
- `rescale_middle`: Centered-type normalized values in `0, 1`, where 1 indicates $x = m$.
- `rescale_interval`: Interval-type normalized values in `0, 1`, where 1 indicates x in `[a, b]`.
- `rescale_extreme`: Extreme-based normalized values using $\min(x)/x$ (positive) or $x/\max(x)$ (negative).
- `rescale_initial`: Initial-based normalized values using $x/x[1]$ or $x[1]/x$.
- `rescale_mean`: Mean-based normalized values using $x/\text{mean}(x)$.
- `to_positive`: Transformed values converting negative indicators to positive using min-max or reciprocal transformation.

Examples

```
# Standardization
x = c(4, 1, NA, 5, 8)
standardize(x)

# L2 norm normalization
normalize(x)

# Min-Max normalization (positive direction)
rescale(x)          # Scale to [0, 1]
rescale(x, type = "-", a = 0.002, b = 0.996) # Reverse scaling

# Negative-to-positive transformation
to_positive(x)      # Min-max transformation
to_positive(x, type = "reciprocal") # Reciprocal transformation

# Centered-type normalization
PH = 6:9
rescale_middle(PH, 7)

# Interval-type normalization
Temp = c(35.2, 35.8, 36.6, 37.1, 37.8, 38.4)
rescale_interval(Temp, 36, 37)

# Extreme-based normalization
rescale_extreme(x)    # min(x)/x
rescale_extreme(x, "-") # x/max(x)

# Initial-based normalization
rescale_initial(x)

# Mean-based normalization
rescale_mean(x)
```

rank_sum_ratio*Rank Sum Ratio (RSR) Evaluation*

Description

Performs Rank Sum Ratio (RSR) evaluation on a dataset of positive indicators, computing ranks, weighted RSR values, and a linear regression model to fit RSR against probit-transformed ranks. Supports integer or non-integer ranking methods.

Usage

```
rank_sum_ratio(data, w = NULL, method = "int")
```

Arguments

data	Data frame with positive indicator data; first column is an ID column for identifying evaluation objects.
w	Numeric vector, weights for indicators (default = equal weights).
method	Character scalar, ranking method: "int" for integer ranks or "non-int" for scaled ranks in 1 , n (default = "int").

Details

The `rank_sum_ratio` function implements the RSR method for evaluating objects based on positive indicators. It ranks the indicators (using integer or non-integer methods), computes weighted RSR values, adjusts ranks with probit transformation, and fits a linear regression model to relate RSR to probit values. The function assumes the first column of data is an ID column, and weights (`w`) can be provided or set to equal weights by default.

Value

A list containing:

- `resultTable`: Data frame with RSR values, ranks, cumulative frequencies, probit values, and fitted RSR values.
- `reg`: Linear model object fitting RSR against probit values.
- `rankTable`: Data frame with ranked indicator values.

Examples

```
# Example data
data = data.frame(ID = c("A", "B", "C"), X1 = c(10, 20, 15), X2 = c(5, 10, 8))
w = c(0.4, 0.6)
rank_sum_ratio(data, w, method = "int")
```

system_evaluation	<i>System Evaluation Functions for Coupling and Obstacle Analysis</i>
-------------------	---

Description

These functions provide tools for system-level evaluation in multi-indicator systems:

- `coupling_degree()`: Computes coupling degree, coordination index, and coupling coordination degree for subsystems.
- `obstacle_degree()`: Computes obstacle degrees for secondary indicators to identify key constraints in the system, enabling batch processing with tidyverse for grouping and summarization.

Usage

```
coupling_degree(data, w = NULL, id_cols = NULL)
```

```
obstacle_degree(data, w = NULL, id_cols = NULL)
```

Arguments

<code>data</code>	A numeric matrix or data frame with normalized scores (usually in 0,1) as columns.
<code>w</code>	Optional vector of weights for indicators or subsystems; defaults to equal weights if NULL.
<code>id_cols</code>	Optional character vector of column names in data to preserve as identifiers (not used in calculations).

Value

A tibble depending on the function:

`coupling_degree` A tibble with columns:

- ID: Identifier columns specified by `id_cols` (if provided).
- `coupling`: Coupling Degree (range 0-1).
- `coord`: Coordination Index (range 0-1).
- `coupling_coord`: Coupling Coordination Degree (range 0-1).

`obstacle_degree` A tibble with:

- ID: Identifier columns specified by `id_cols` (if provided).
- Columns for secondary indicator obstacle degrees ($O_{ij} = (1 - X_{ij}) \cdot w_{ij}$).

Suitable for grouping and summarizing (e.g., with tidyverse) to compute primary indicator obstacle degrees ($\bigwedge (U_i \setminus)$).

Examples

```
# Sample normalized subsystem scores
df = data.frame(
  s1 = c(0.0162, 0.1782, 0.5490, 0.6730, 0.0207, 0.9875),
  s2 = c(0.2720, 0.6824, 0.0593, 0.4812, 0.8891, 0.5573)
)
# Coupling Degree Analysis
coupling_degree(df) # Equal weights
coupling_degree(df, c(0.6, 0.4))
# Obstacle Degree Analysis
obstacle_degree(df) # Equal weights
obstacle_degree(df, c(0.6, 0.4))
```

topsis

TOPSIS Method for Multi-Criteria Decision Making

Description

Implements the Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) to rank alternatives based on multiple criteria. The function normalizes the decision matrix using Min-Max method, applies weights, and computes relative closeness to the ideal solution.

Usage

```
topsis(X, w = NULL, index = NULL)
```

Arguments

X	A numeric matrix or data frame where rows represent alternatives and columns represent criteria.
w	A numeric vector of weights for each criterion. Must be non-negative and sum to 1. If not provided, equal weights are used.
index	A character vector indicating the direction of each indicator: Use "+" for positive indicators (higher is better), "-" for negative indicators (lower is better). If index = NULL (default), all indicators are treated as "+".

Details

The TOPSIS method ranks alternatives by:

1. Normalizing the decision matrix using Min-Max normalization.
2. Applying weights to form a weighted normalized matrix.
3. Identifying positive and negative ideal solutions based on indicator directions.
4. Computing Euclidean distances to ideal solutions.
5. Calculating relative closeness as $S_0 / (S_0 + S_{star})$, where S_0 is the distance to the negative ideal and S_{star} is the distance to the positive ideal.

This implementation supports both positive and negative indicators via the index parameter.

Value

A named numeric vector of relative closeness scores (in 0, 1) for each alternative. Higher values indicate better alternatives. Names are taken from rownames(X) or default to "Sample1", "Sample2", etc.

Examples

```
A = data.frame(
  X1 = c(2, 5, 3), # "+"
  X2 = c(8, 1, 6)  # "-"
)
w = c(0.6, 0.4)
idx = c("+", "-")
topsis(A, w, idx)
```

water_quality	Water Quality Dataset
---------------	-----------------------

Description

A dataset containing water quality evaluation metrics for 20 rivers, including dissolved oxygen (O2, positive indicator), pH value (PH, centered indicator), total bacteria count (germ, negative indicator), and plant nutrient content (nutrient, interval indicator with optimal range 10-20). This dataset is suitable for multi-criteria decision analysis, such as weight calculation and fuzzy comprehensive evaluation in the mathmodels package.

Usage

```
water_quality
```

Format

- A data frame with 20 rows and 5 columns:
- ID** Numeric, unique identifier for each river (1 to 20).
 - O2** Numeric, dissolved oxygen content (mg/L), higher values are better (positive indicator).
 - PH** Numeric, pH value, values closer to 7 are optimal (centered indicator).
 - germ** Numeric, total bacteria count, lower values are better (negative indicator).
 - nutrient** Numeric, plant nutrient content (mg/L), optimal range is 10-20 (interval indicator).

Details

Water Quality Dataset

Source

Simulated data for water quality evaluation, created for demonstration purposes.

water_quality

29

Examples

```
# Load the dataset  
data(water_quality)
```

```
# Preview the data  
head(water_quality)
```

Index

* datasets

water_quality, 28

0, 1, 11, 13, 20, 23, 24, 28

0, 1, 26

1, n, 25

AHP, 2

basic_DEA (DEA), 7

basic_SBM (DEA), 7

combine_preds, 3

combine_weights, 3

compute_mf, 4

compute_mf_funs (compute_mf), 4

coupling_degree (system_evaluation), 26

critic_weight, 5

cv_weight, 6

DEA, 7

defuzzify, 11

DGM21 (grey_models), 15

dsigmoid_mf (membership), 18

entropy_weight, 12

fuzzy_eval, 13

gauss2mf (membership), 18

gauss_mf (membership), 18

gbell_mf (membership), 18

gini (inequality), 16

gini0 (inequality), 16

GM11 (grey_models), 15

GM1N (grey_models), 15

grey_analysis, 14

grey_corr (grey_analysis), 14

grey_corr_topsis (grey_analysis), 14

grey_models, 15

inequality, 16

malmquist (DEA), 7

membership, 18

normalize (preprocess), 22

obstacle_degree (system_evaluation), 26

pca_weight, 21

pi_mf (membership), 18

plot_mf (membership), 18

preprocess, 22

psigmoid_mf (membership), 18

rank_sum_ratio, 25

rescale (preprocess), 22

rescale_extreme (preprocess), 22

rescale_initial (preprocess), 22

rescale_interval (preprocess), 22

rescale_mean (preprocess), 22

rescale_middle (preprocess), 22

s_mf (membership), 18

sigmoid_mf (membership), 18

standardize (preprocess), 22

super_DEA (DEA), 7

super_SBM (DEA), 7

system_evaluation, 26

theil (inequality), 16

theil0 (inequality), 16

theil0_g (inequality), 16

theil_g (inequality), 16

theil_g2_cross (inequality), 16

theil_g2_nest (inequality), 16

to_positive (preprocess), 22

topsis, 27

trap_mf (membership), 18

tri_mf (membership), 18

verhulst (grey_models), 15

water_quality, 28

z_mf (membership), 18