Package: LCPA (via r-universe)

June 9, 2026

Type Package

Title A General Framework for Latent Classify and Profile Analysis

Version 1.0.2

Date 2026-04-10

Author Haijiang Qin [aut, cre, cph] (ORCID: <https://orcid.org/0009-0000-6721-5653>), Lei Guo [aut, cph] (ORCID: <https://orcid.org/0000-0002-8273-3587>)

Maintainer Haijiang Qin <haijiang133@outlook.com>

Description A unified latent class modeling framework that encompasses both latent class analysis (LCA) and latent profile analysis (LPA), offering a one-stop solution for latent class modeling. It implements state-of-the-art parameter estimation methods, including the expectation–maximization (EM) algorithm, neural network estimation (NNE; requires users to have 'Python' and its dependent libraries installed on their computer), and integration with 'Mplus' (requires users to have 'Mplus' installed on their computer). In addition, it provides commonly used model fit indices such as the Akaike information criterion (AIC) and Bayesian information criterion (BIC), as well as classification accuracy measures such as entropy. The package also includes fully functional likelihood ratio tests (LRT) and bootstrap likelihood ratio tests (BLRT) to facilitate model comparison, along with bootstrap-based and observed information matrix-based standard error estimation. Furthermore, it supports the standard three-step approach for LCA, LPA, and latent transition analysis (LTA) with covariates, enabling detailed covariate analysis. Finally, it includes several user-friendly auxiliary functions to enhance interactive usability.

License GPL-3

Depends R (>= 4.1.0)

Imports reticulate, clue, ggplot2, tidyr, dplyr, mvtnorm, Matrix, MASS, MplusAutomation, tidyselect, numDeriv, nloptr, patchwork, Rcpp, reshape2, scales

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.3.3

Encoding UTF-8

NeedsCompilation yes

Collate 'adjust.response.R' 'adjust.model.R' 'check.response.R'
'compare.model.R' 'EM.LCA.R' 'EM.LPA.R' 'get.AvePP.R'
'get.CEP.R' 'get.entropy.R' 'get.fit.index.R'
'get.Log.Lik.LCA.R' 'get.Log.Lik.LPA.R' 'get.Log.Lik.LTA.R'
'get.npar.LCA.R' 'get.npar.LPA.R' 'get.npar.LTA.R'
'get.P.Z.Xn.LCA.R' 'get.P.Z.Xn.LPA.R' 'get.SE.R'
'install_python_dependencies.R' 'Kmeans.LCA.R' 'LCA.R' 'LCPA.R'
'logit.R' 'LPA.R' 'LTA.R' 'LRT.test.R' 'LRT.test.Bootstrap.R'
'LRT.test.VLMR.R' 'Mplus.LCA.R' 'Mplus.LPA.R' 'normalize.R'
'plotResponse.R' 'RcppExports.R' 'rdirichlet.R' 'S3extract.R'
'S3plot.R' 'S3print.R' 'S3summary.R' 'S3update.R'
'sim.correlation.R' 'sim.LCA.R' 'sim.LPA.R' 'sim.LTA.R'
'tools.R' 'utils.R' 'logpdf_component.R' 'zzz.R'

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

Date/Publication 2026-04-10 08:02:47 UTC

RemoteUrl https://github.com/cran/LCPA

RemoteRef HEAD

RemoteSha ed93c5f74bfaa46d488873548914836cb4f49b86

Contents

`adjust.model` *Align Latent Class/Profile Models via Optimal Permutation*

Description

This function reorders the latent classes/profiles of `object2` to best match those in `object1` by minimizing the total assignment cost based on posterior class membership (MAP classification). It uses the Linear Sum Assignment Problem (LSAP) solver to find the optimal one-to-one mapping between latent classes. Useful for comparing or averaging models across replications, initializations, or algorithms where class labels may be permuted.

Usage

```
adjust.model(object1, object2)
```

Arguments

`object1`	An object of class `"LCA"` or `"LPA"`, typically the reference model.
`object2`	An object of class `"LCA"` or `"LPA"`, whose latent classes will be reordered to align with `object1`.

Details

The alignment is performed by:

1. Computing Maximum A Posteriori (MAP) classification matrices for both models.
2. Calculating a distance matrix between classes (typically Euclidean distance between binary MAP vectors).
3. Solving the Linear Sum Assignment Problem (LSAP) via `solve_LSAP` to find the permutation minimizing total mismatch cost.
4. Reordering all class-specific components of `object2` according to this optimal assignment.

Value

A modified version of `object2`, with all parameters and posterior probabilities reordered to best match the latent class structure of `object1`. The returned object retains its original class (`"LCA"` or `"LPA"`) and includes aligned:

- Prior probabilities (`P.Z`)
- Posterior probabilities (`P.Z.Xn`)
- MAP classifications (`Z`)
- Profile/Class-specific parameters:
  - For `"LPA"`: `means`, `covs`
  - For `"LCA"`: `par`, `probability`
- All relevant `dimnames` and `names` are synchronized with `object1`

Note

- Both models must have identical numbers of observations (N), latent classes (L), and indicators (I).
- Designed for use after fitting multiple models (e.g., different random starts) to ensure consistent class labeling.
- Does not modify `object1`; only returns a reordered `object2`.

Examples

```
## Not run:
# need Mplus and Pyrthon

library(LCPA)
set.seed(123)

data.obj <- sim.LCA(N = 500, I = 4, L = 3)

# Fit two models with different random seeds
fit1 <- LCA(data.obj$response, L = 3, method = "Mplus", nrep = 1)
fit2 <- LCA(data.obj$response, L = 3, method = "NNE", nrep = 1)

# Align fit2 to fit1's class ordering
```

```
fit2_aligned <- adjust.model(fit1, fit2)

# Compare prior probabilities before and after
print("Before alignment:")
print(fit2$params$P.Z)
print("After alignment:")
print(fit2_aligned$params$P.Z)


## End(Not run)
```

`adjust.response` *Adjust Categorical Response Data for Polytomous Indicators*

Description

Standardizes polytomous response data by converting raw category values to consecutive integers starting from 0. Records original category values for potential reverse transformation. Handles varying numbers of response categories across indicators.

Usage

```
adjust.response(response)
```

Arguments

`response`	A matrix or data frame containing response data where: • Rows represent respondents (N observations) • Columns represent indicators/items/questions (I indicators) • Cells contain raw response values (numeric) Non-numeric columns will be coerced to numeric with warning.

Details

The function processes each indicator column independently:

1. Extracts unique response values and sorts them in ascending order
2. Maps smallest value to 0, second smallest to 1, etc.
3. Records original values in `poly.orig` for possible reverse transformation
4. Handles indicators with different numbers of categories through NA-padding

Missing values (`NA`) in input are preserved as `NA` in output.

Value

A named list containing:

`poly.orig` $I \times K_{max}$ matrix. Original sorted category values for each indicator. Rows correspond to indicators, columns to category positions. Empty cells filled with `NA`.

`poly.value` Integer vector of length I. Number of unique response categories per indicator.

`poly.max` Scalar integer. Maximum number of categories across all indicators, i.e., K_{max}.

`response` $N \times I$ matrix. Adjusted response data where original values are replaced by zero-based category indices (0 to $k - 1$ for k categories).

Examples

```
# Simulate response data with 3 indicators and varying categories
set.seed(123)
resp <- data.frame(
  indicator1 = sample(1:3, 10, replace = TRUE),
  indicator2 = sample(c(0, 5, 10), 10, replace = TRUE),
  indicator3 = sample(1:2, 10, replace = TRUE)
)

# Apply adjustment
adjusted <- adjust.response(resp)

# Inspect results
str(adjusted)
print(adjusted$poly.orig)  # Original category values
print(adjusted$response)   # Standardized responses
```

---

check.response *Validate response matrix against expected polytomous category counts*

Description

Checks whether each column in the response matrix contains exactly the number of unique response categories specified in `poly.value`. Handles edge cases where all indicators have identical category counts efficiently.

Usage

```
check.response(response, poly.value)
```

Arguments

`response` A numeric matrix of dimension $N \times I$, where:

- N: Number of subjects/observations (rows)
- I: Number of indicators/items/variables (columns)

Each cell contains the observed response value for a subject on an indicator.

`poly.value` An integer vector of length I specifying the expected number of unique response categories (levels) for each corresponding indicator in `response`. Values must be positive integers.

Value

Logical value indicating validation status:

- `TRUE` if either:
  - All columns have identical numbers of unique values (regardless of `poly.value` specification)
  - Each column's unique value count matches its corresponding `poly.value` entry
- `FALSE` if any column's unique value count mismatches its specified `poly.value` (when columns have varying category counts)

Note

This function contains a specific behavior: When all indicators have identical numbers of unique response categories, it returns `TRUE` immediately without validating against `poly.value`. This may lead to unexpected results if `poly.value` contains inconsistent expectations. Users should ensure `poly.value` accurately reflects their measurement model.

Examples

```
# Valid case: Matching category counts
resp_matrix <- matrix(c(1,1,2,2, 1,2,3,1), ncol = 2)
check.response(resp_matrix, poly.value = c(2, 3))  # Returns TRUE

# Invalid case: Mismatched category counts
check.response(resp_matrix, poly.value = c(2, 2))  # Returns FALSE

# Special case: Uniform category counts bypass poly.value check
uniform_resp <- matrix(rep(1:2, each = 4), ncol = 2)
check.response(uniform_resp, poly.value = c(2, 5))  # Returns TRUE (bypass behavior)
```

`compare.model` *Model Comparison Tool*

Description

Compares two nested latent class/profile models using multiple fit indices, likelihood ratio tests, and classification metrics.

Usage

```
compare.model(object1, object2, n.Bootstrap = 0)
```

Arguments

object1	An object of class LCA or LPA, representing the first latent class/profile model.
object2	An object of class LCA or LPA, representing the second latent class/profile model. Must be of the same type as object1.
n.Bootstrap	Integer specifying the number of bootstrap replications for the parametric bootstrap likelihood ratio test (BLRT). Default is 0 (no bootstrap test performed).

Details

This function performs comprehensive model comparison between two nested LCA/LPA models. Key features include:

- Automatically orders models by parameter count (smaller model first)
- Computes multiple fit indices via get.fit.index
- Calculates classification quality metrics (entropy, average posterior probabilities)
- Performs three types of likelihood ratio tests:
  - Standard LRT, see LRT.test
  - VLMR adjusted LRT, see LRT.test.VLMR
  - Parametric bootstrap LRT (computationally intensive but robust), see LRT.test.Bootstrap
- Computes Bayes Factor using Sample-Size Adjusted BIC (SIC)

Important Requirements:

- Both models must be of the same type (LCA or LPA)
- Models must be nested (one model is a constrained version of the other)
- n.Bootstrap > 0 requires significant computational resources

Value

An object of class compare.model containing:

npar Named vector with number of free parameters for each model

entropy Named vector with entropy values (classification accuracy measure) for each model

AvePP List containing average posterior probabilities per latent class/profile

fit.index List of get.fit.index objects for both models

BF Bayes Factor for model comparison (based on SIC)

LRT.obj Likelihood ratio test (LRT) results

LRT.VLMR.obj Vuong-Lo-Mendell-Rubin (VLMR) adjusted LRT results

LRT.Bootstrap.obj Bootstrap LRT results (if n.Bootstrap > 0)

call The matched function call

arguments List containing the original arguments passed to the function

See Also

LCA, LPA, get.fit.index, extract, LRT.test, LRT.test.VLMR

Examples

```
library(LCPA)
set.seed(123)

data.obj <- sim.LPA(N = 500, I = 5, L = 3, constraint = "V0")
response <- data.obj$response

# need Mplus
## Not run:
# Compare 3-class vs 4-class LPA models
object1 <- LPA(response, L = 3, method = "Mplus", constraint = "V0")
object2 <- LPA(response, L = 4, method = "Mplus", constraint = "V0")

compare.model.obj <- compare.model(object1, object2)

print(compare.model.obj)

## End(Not run)
```

`extract` *S3 Methods: extract*

Description

A generic S3 extractor function designed to retrieve internal components from various model and simulation objects produced by the `LCPA` package. This function provides a consistent interface across different classes, allowing users to access estimated parameters, fit statistics, simulation truths, standard errors, and more.

Usage

```
extract(object, what, ...)

## S3 method for class 'LCA'
extract(object, what, ...)

## S3 method for class 'LPA'
extract(object, what, ...)

## S3 method for class 'LCPA'
extract(object, what, ...)

## S3 method for class 'LTA'
extract(object, what, ...)

## S3 method for class 'sim.LCA'
extract(object, what, ...)
```

```
## S3 method for class 'sim.LPA'
extract(object, what, ...)

## S3 method for class 'sim.LTA'
extract(object, what, ...)

## S3 method for class 'fit.index'
extract(object, what, ...)

## S3 method for class 'compare.model'
extract(object, what, ...)

## S3 method for class 'SE'
extract(object, what, ...)
```

Arguments

`object`	An object of one of the following classes: • `LCA` — Latent Class Analysis model results. • `LPA` — Latent Profile Analysis model results. • `LCPA` — Latent Class/Profile Analysis with covariates. • `LTA` — Latent Transition Analysis model results. • `sim.LCA` — Simulated LCA data with known truth. • `sim.LPA` — Simulated LPA data with known truth. • `sim.LTA` — Simulated LTA data with known truth. • `get.fit.index` — Model fit indices object. • `compare.model` — Model comparison results. • `get.SE` — Standard error estimation results.
`what`	A character string specifying the name of the component to extract. Valid choices depend on the class of `object`. See Details section for full listings.
`...`	Additional arguments passed to methods (currently ignored).

Details

This function supports extraction from ten primary object classes. Below are available components for each:

`LCA` Latent Class Analysis model results. Available components:

- `params` List containing all estimated model parameters.
- `par` 3D array ($L \times I \times K_{\max}$) of conditional response probabilities.
- `P.Z` Vector of length L with latent class prior probabilities.
- `npar` Number of free parameters in the model.
- `Log.Lik` Log-likelihood of the final model.
- `AIC` Akaike Information Criterion.
- `BIC` Bayesian Information Criterion.

`best_BIC` Best BIC value across replication runs (if `nrep > 1`).

`P.Z.Xn` $N \times L$ matrix of posterior class probabilities.

`Z` Vector of length N with MAP-classified latent class memberships.

`probability` List of formatted conditional probability matrices per item.

`Log.Lik.history` Vector tracking log-likelihood at each EM iteration.

`Log.Lik.nrep` Vector of log-likelihoods from each replication run.

`model` Trained neural network model object (only when `method="NNE"`).

`call` The original function call used for model estimation.

`arguments` List containing all input arguments passed to the `LCA` function.

`LPA` Latent Profile Analysis model results. Available components:

`params` List containing all estimated model parameters.

`means` $L \times I$ matrix of estimated mean vectors for each profile.

`covs` $I \times I \times L$ array of estimated covariance matrices.

`P.Z` Vector of length L with profile prior probabilities.

`npar` Number of free parameters (depends on `constraint`).

`Log.Lik` Log-likelihood of the final model.

`AIC` Akaike Information Criterion.

`BIC` Bayesian Information Criterion.

`best_BIC` Best BIC value across replication runs (if `nrep > 1`).

`P.Z.Xn` $N \times L$ matrix of posterior profile probabilities.

`Z` Vector of length N with MAP-classified profile memberships.

`Log.Lik.history` Vector tracking log-likelihood at each EM iteration.

`Log.Lik.nrep` Vector of log-likelihoods from each replication run.

`model` Trained model object (neural network or Mplus).

`call` The original function call used for model estimation.

`arguments` List containing all input arguments passed to the `LPA` function.

`constraint` Covariance structure constraints applied during estimation (from original arguments).

`LCPA` Latent Class/Profile Analysis (with covariates). Available components:

`beta` Initial class coefficients (p1 x L matrix).

`beta.se` Standard errors for beta.

`beta.Z.sta` Z-statistics for beta.

`beta.p.value.tail1` One-tailed p-values for beta.

`beta.p.value.tail2` Two-tailed p-values for beta.

`P.Z.Xn` Posterior probabilities (N x L).

`P.Z` Prior proportions (length L).

`Z` Modal class assignments (length N).

`npar` Number of free parameters.

`Log.Lik` Log-likelihood.

`AIC` AIC.

`BIC` BIC.

`iterations` Optimization iterations in Step 3.

`coveraged` Logical: did optimization converge early?

`params` Step 1 model parameters (LCA/LPA output).

`call` Function call.

`arguments` Input arguments list.

`LTA` Latent Transition Analysis model results. Available components:

`beta` Initial class coefficients (p1 x L matrix).

`gamma` Transition coefficients (nested list).

`beta.se` Standard errors for beta.

`gamma.se` Standard errors for gamma.

`beta.Z.sta` Z-statistics for beta.

`gamma.Z.sta` Z-statistics for gamma.

`beta.p.value.tail1` One-tailed p-values for beta.

`gamma.p.value.tail1` One-tailed p-values for gamma.

`beta.p.value.tail2` Two-tailed p-values for beta.

`gamma.p.value.tail2` Two-tailed p-values for gamma.

`P.Z.Xns` List of posterior probabilities per time (each N x L).

`P.Zs` List of prior proportions per time (each length L).

`Zs` List of modal class assignments per time (each length N).

`npar` Number of free parameters.

`Log.Lik` Log-likelihood.

`AIC` AIC.

`BIC` BIC.

`iterations` Optimization iterations in Step 3.

`coveraged` Logical: did optimization converge early?

`params` Step 1 model parameters (LCA/LPA output).

`call` Function call.

`arguments` Input arguments list.

`sim.LCA` Simulated Latent Class Analysis data. Available components:

`response` Integer matrix ($N \times I$) of simulated categorical observations.

`par` Array ($L \times I \times P_{\max}$) of true class-specific category probabilities.

`Z` Integer vector (length N) of true latent class assignments.

`P.Z` Numeric vector (length L) of true class proportions.

`poly.value` Integer vector (length I) specifying categories per variable.

`P.Z.Xn` Binary matrix ($N \times L$) of true class membership indicators.

`call` The original function call used for simulation.

`arguments` List containing all input arguments passed to `sim.LCA`.

`sim.LPA` Simulated Latent Profile Analysis data. Available components:

`response` Numeric matrix ($N \times I$) of simulated continuous observations.

`means` $L \times I$ matrix of true class-specific means.

`covs` $I \times I \times L$ array of true class-specific covariance matrices.

`P.Z.Xn` $N \times L$ matrix of true class membership indicators.

`P.Z` Numeric vector (length L) of true class proportions.

`Z` Integer vector (length N) of true profile assignments.

`constraint` Original constraint specification passed to `sim.LPA`.

`call` The original function call used for simulation.

`arguments` List containing all input arguments passed to `sim.LPA`.

`sim.LTA` Simulated Latent Transition Analysis data. Available components:

`responses` List of response matrices per time point.

`Zs` List of true latent class assignments per time.

`P.Zs` List of true class proportions per time.

`par` True conditional probabilities (for categorical items).

`means` True profile means (for continuous variables).

`covs` True covariance matrices per class and time.

`poly.value` Categories per variable (for categorical items).

`rate` Transition rate matrix or structure.

`covariates` Simulated covariate matrix.

`beta` True initial class coefficients.

`gamma` True transition coefficients.

`call` Original simulation function call.

`arguments` Input arguments used in simulation.

`fit.index` Model fit indices object. Available components:

`npar` Number of free parameters in the model.

`Log.Lik` Log-likelihood of the model.

`-2LL` Deviance statistic (-2 times log-likelihood).

`AIC` Akaike Information Criterion.

`BIC` Bayesian Information Criterion.

`SIC` Sample-Size Adjusted BIC (-0.5 * BIC).

`CAIC` Consistent AIC.

`AWE` Approximate Weight of Evidence.

`SABIC` Sample-Size Adjusted BIC (alternative formulation).

`call` Original function call that generated the fit indices.

`arguments` List containing input arguments (includes original model object).

`compare.model` Model comparison results. Available components:

`npar` Named numeric vector with free parameters for each model (`model1`, `model2`).

`entropy` Named numeric vector with entropy values for each model.

`AvePP` List of average posterior probabilities per class/profile for each model.

`fit.index` List of `get.fit.index` objects for both models.

`BF` Bayes Factor comparing models (based on SIC differences).

`LRT.obj` Standard likelihood ratio test results (requires nested models).

`LRT.VLMR.obj` Vuong-Lo-Mendell-Rubin adjusted likelihood ratio test results.

`LRT.Bootstrap.obj` Parametric bootstrap likelihood ratio test results (if `n.Bootstrap` > `0`).

`call` The matched function call used for comparison.

`arguments` List containing original input arguments (`object1`, `object2`, `n.Bootstrap`).

`SE` Standard error estimation results. Available components:

`se` List containing standard errors for parameters (components depend on model type).
`vcov` Variance-covariance matrix (only for `method="Obs"`).
`hessian` Observed information matrix (only for `method="Obs"`).
`diagnostics` Method-specific diagnostic information (e.g., estimation method).
`call` Function call that generated the object.
`arguments` Input arguments used in estimation.
`means` Standard errors for profile means (LPA models only — accessed via `se` list).
`covs` Standard errors for covariance parameters (LPA models only — accessed via `se` list).
`P.Z` Standard errors for class proportions (both LCA/LPA — accessed via `se` list).
`par` Standard errors for conditional probabilities (LCA models only — accessed via `se` list).

Value

The requested component. Return type varies depending on `what` and the class of `object`. If an invalid `what` is provided, an informative error is thrown listing valid options.

Methods (by class)

- `extract(LCA)`: Extract fields from a `LCA` object
- `extract(LPA)`: Extract fields from a `LPA` object
- `extract(LCPA)`: Extract fields from a `LCPA` object
- `extract(LTA)`: Extract fields from a `LTA` object
- `extract(sim.LCA)`: Extract fields from a `sim.LCA` object
- `extract(sim.LPA)`: Extract fields from a `sim.LPA` object
- `extract(sim.LTA)`: Extract fields from a `sim.LTA` object
- `extract(fit.index)`: Extractor method for `fit.index` objects
- `extract(compare.model)`: Extract fields from a `compare.model` object
- `extract(SE)`: Extract fields from a `SE` object

Usage Notes

- For `LCA`, `LPA`, `LCPA`, and `LTA` objects, components reflect *estimated* parameters.
- For `sim.LCA`, `sim.LPA`, and `sim.LTA` objects, components reflect *true* data-generating parameters.
- In `SE` objects:
  - Top-level components like `vcov` and `hessian` are only available when `method = "Obs"`. Requesting them under `Bootstrap` triggers a warning and returns `NULL`.
  - Parameter-specific SEs (e.g., `means`, `par`) are stored within the `se` list. You can extract them directly by name (e.g., `extract(se_obj, "means")`).
  - Attempting to extract unavailable parameter SEs (e.g., `par` from an LPA model) triggers an error with available options.
- For `fit.index` and `compare.model` objects, valid components are dynamically determined from the object's names.
- All methods ignore additional arguments (`...`).

Examples

```
set.seed(123)

# Simulate LPA data: 500 observations, 3 continuous variables, 2 latent profiles
# Constraint "E0": Equal variances across classes, zero covariances
data.obj <- sim.LPA(N = 500, I = 3, L = 2, constraint = "E0")

# Extract the simulated response matrix (N x I) for model fitting
response <- extract(data.obj, "response")

# Extract the TRUE covariance matrices (I x I x L array)
extract(data.obj, "covs")

# Fit an LPA model to the simulated data using the SAME constraint ("E0")
fit_E0 <- LPA(response, L = 2, constraint = "E0")

# Extract the ESTIMATED covariance matrices from the fitted model
extract(fit_E0, "covs")

# Simulate LCA data: 30 observations, 5 categorical items, 3 latent classes
sim_data <- sim.LCA(N = 30, I = 5, L = 3)

# Extract the TRUE conditional probability array
extract(sim_data, "par")
```

---

get.AvePP *Calculate Average Posterior Probability (AvePP)*

Description

Computes the average posterior probability for the most likely class assignment in latent class/profile analysis. This metric quantifies classification precision. The total average posterior probability ≥ 0.70 (Nylund-Gibson & Choi, 2018) indicate adequate classification quality.

Usage

```
get.AvePP(object)
```

Arguments

object	An object of class "LCA" or "LPA" returned by LCA or LPA, or any object containing:

- P.Z.Xn: $N \times L$ matrix of posterior class probabilities, where:
  - N = Total number of observations ($n = 1, 2, \ldots, N$)
  - L = Number of latent classes ($l = 1, 2, \ldots, L$)
  - Element $p_{nl} = P(Z_n = l \mid \mathbf{X}_n)$ denotes the posterior probability that observation n belongs to class l given observed data $\mathbf{X}_n$

Value

A $(L+1) \times (L+1)$ matrix with the following structure:

- Rows: Represent each latent class (1 to L) and a final "Total" row.
- Columns: Represent each latent class (1 to L) and a final "Total" column.
- Diagonal elements ave$[l, l]$: Average posterior probability for observations assigned to class l. That is,

$$\overline{P}_{ll} = \frac{1}{N_l} \sum_{n:\hat{z}_n = l} p_{nl},$$

  where N_l is the number of observations assigned to class l, and $\hat{z}_n = \arg\max_{l'} p_{nl'}$.
- Off-diagonal elements ave$[l, k]$ ($l \neq k$): Average posterior probability of class k among observations assigned to class l. Useful for assessing classification confusion.

$$\overline{P}_{lk} = \frac{1}{N_l} \sum_{n:\hat{z}_n = l} p_{nk}.$$

- Bottom-right corner ave$[L+1, L+1]$: Overall average posterior probability across all observations,

$$\overline{P}_{\text{total}} = \frac{1}{N} \sum_{n=1}^{N} \max_{l} p_{nl}.$$

Note

Classification quality is considered acceptable if $\overline{P}_{\text{total}} \geq 0.70$ (Nylund-Gibson & Choi, 2018).

References

Nylund-Gibson, K., & Choi, A. Y. (2018). Ten frequently asked questions about latent class analysis. Translational Issues in Psychological Science, 4(4), 440-461. https://doi.org/10.1037/tps0000176

Examples

```
# Example with simulated data
set.seed(123)
data.obj <- sim.LCA(N = 500, I = 4, L = 2, IQ=0.9)
response <- data.obj$response

# Fit 2-class model with EM algorithm

fit.em <- LCA(response, L = 2, method = "EM", nrep = 10)

AvePP_value <- get.AvePP(fit.em)
print(AvePP_value)
```

`get.CEP` *Compute Classification Error Probability (CEP) Matrices*

Description

Computes the Classification Error Probability (CEP) matrices (Liang et al., 2023) used in the bias-corrected three-step estimation of Latent Class/Profile Analysis with Covariates.

Usage

```
get.CEP(P.Z.Xns, time.cross = TRUE)
```

Arguments

`P.Z.Xns`	A list of length T (number of time points). Each element is an $N \times L$ matrix of posterior probabilities $P(Z_{it} = l \mid X_i)$ from the first-step model. • Rows correspond to individuals ($i = 1, \ldots, N$); • Columns correspond to latent classes ($l = 1, \ldots, L$); • Each row must sum to 1. The list must be ordered chronologically (e.g., time 1 to T).
`time.cross`	Logical. If `TRUE` (default), returns a list where every element is the same pooled CEP matrix (averaged across all time points). If `FALSE`, returns time-specific CEP matrices.

Details

The CEP matrix at time t gives the probability that an individual truly belongs to latent class l' given that they were assigned (via modal assignment) to class l at time t.

Formally, for time point t:

$$\text{CEP}_t(l, l') = P(Z_t = l \mid \hat{Z}_t = l') = \frac{\sum_{i:\, \hat{z}_{it} = l'} P(Z_{it} = l \mid X_i)}{N\, \hat{\pi}_{tl}}$$

where:

- Z_{it} is the true latent class of individual i at time t;
- $P(Z_{it} = l \mid X_i)$ is the posterior probability from the first-step model;
- $\hat{z}_{it} = \arg\max_l P(Z_{it} = l' \mid X_i)$ is the modal (most likely) assigned class;
- $\hat{\pi}_{tl} = \frac{1}{N} \sum_{i=1}^{N} I(\hat{z}_{it} = l)$ is the observed proportion assigned to class l at time t;
- N is the total sample size.

If `time.cross = TRUE` (default), a single pooled CEP matrix is computed by aggregating counts across all time points. This assumes the classification error structure is invariant over time (i.e., measurement invariance), as in Liang et al. (2023). The same pooled matrix is then returned for every time point.

Value

A named list of length T. Each element is an $L \times L$ matrix:

- Row l: true latent class;
- Column l': individuals assigned to class l';
- Entry (l, l'): estimated $P(\text{assigned class} = l' \mid \text{true class} = l)$.

When `time.cross = TRUE`, all matrices in the list are identical. Names are `"t1"`, `"t2"`, ..., `"tT"`.

Note

- Assumes complete data (no missing values in posterior matrices).
- All matrices in `P.Z.Xns` must have identical dimensions (same N and L).
- Assignment is based on modal class (`which.max`).
- If no individual is assigned to a class at a time point, division by zero may occur.

References

Liang, Q., la Torre, J. d., & Law, N. (2023). Latent Transition Cognitive Diagnosis Model With Covariates: A Three-Step Approach. Journal of Educational and Behavioral Statistics, 48(6), 690-718. https://doi.org/10.3102/10769986231163320

Examples

```
# Simulate posterior probabilities for 2 time points, 3 classes, 100 individuals
set.seed(123)
N <- 100; L <- 3; times <- 2
P.Z.Xns <- replicate(times,
  t(apply(matrix(runif(N * L), N, L), 1, function(x) x / sum(x))),
  simplify = FALSE)

# Compute time-specific CEP matrices
cep_time_specific <- get.CEP(P.Z.Xns, time.cross = FALSE)

# Compute time-invariant (pooled) CEP matrix
cep_pooled <- get.CEP(P.Z.Xns, time.cross = TRUE)
```

`get.entropy` *Calculate Classification Entropy*

Description

Computes the relative entropy statistic to evaluate classification quality in Latent Class Analysis (LCA) or Latent Profile Analysis (LPA) models. Entropy measures how accurately cases are assigned to latent classes based on posterior probabilities, with values closer to 1 indicating better separation between classes.

Usage

```
get.entropy(object)
```

Arguments

`object`	An object of class `"LCA"` or `"LPA"` returned by `LCA` or `LPA` functions, or any other object containing: • `P.Z.Xn`: $N \times L$ matrix of posterior class probabilities for each observation. • `params$P.Z`: Vector of length L with latent class prior probabilities.

Value

A numeric value between 0 and 1 representing the relative entropy (Nylund-Gibson et al., 2018; Clark et al., 2013):

- 1.0: Perfect classification (each case belongs exclusively to one class)
- 0.8-1.0: Good classification quality
- 0.6-0.8: Moderate classification quality
- < 0.6: Poor classification quality (consider model simplification)

Calculated using the formula:

$$1 - \frac{\sum_{n=1}^{N} \sum_{l=1}^{L} -p_{nl} \ln(p_{nl})}{N \ln(L)}$$

where:

- N = Sample size
- L = Number of latent classes
- p_{nl} = Posterior probability of observation n belonging to class l

Note

The calculation includes a small constant (`1e-10`) to avoid `log(0)` instability when posterior probabilities are exactly zero.

Values should be interpreted alongside other diagnostics (BIC, bootstrapped LRT) as high entropy alone doesn't guarantee model validity. Low entropy may indicate:

- Overly complex model (too many classes)
- Poorly measured latent constructs
- Violation of local independence assumption

References

Nylund-Gibson, K., & Choi, A. Y. (2018). Ten frequently asked questions about latent class analysis. Translational Issues in Psychological Science, 4(4), 440-461. https://doi.org/10.1037/tps0000176

Clark, S. L., Muthén, B., Kaprio, J., D'Onofrio, B. M., Viken, R., & Rose, R. J. (2013). Models and Strategies for Factor Mixture Analysis: An Example Concerning the Structure Underlying Psychological Disorders. Structural Equation Modeling: A Multidisciplinary Journal, 20(4), 681-703. https://doi.org/10.1080/10705511.2013.824786

Examples

```
# Example with simulated data
set.seed(123)
data.obj <- sim.LCA(N = 500, I = 4, L = 2, IQ=0.9)
response <- data.obj$response

# Fit 2-class model with EM algorithm

fit.em <- LCA(response, L = 2, method = "EM", nrep = 10)

entropy_value <- get.entropy(fit.em)
cat("Classification entropy:", round(entropy_value, 3), "\n")
```

---

get.fit.index *Calculate Fit Indices*

Description

Computes a comprehensive set of model fit indices for objects returned by LCA or LPA. These indices balance model fit (log-likelihood) with model complexity (number of parameters) to facilitate model selection. All indices are derived from the observed-data log-likelihood and parameter count.

Usage

```
get.fit.index(object)
```

Arguments

object	An object of class "LCA" or "LPA" returned by LCA, LPA or any object containing: • Log.Lik: Log-likelihood value • npar: Number of free parameters • N = Total number of observations ($n = 1, 2, \ldots, N$)

Value

An object of class "fit.index" containing:

npar Number of free parameters in the model

Log.Lik Log-likelihood of the model: $\log \mathcal{L}$

-2LL Deviance statistic: $-2 \log \mathcal{L}$

$$-2 \sum_{n=1}^{N} \log \left[\sum_{l=1}^{L} \pi_l \cdot f(\mathbf{x}_n \mid \boldsymbol{\theta}_l) \right]$$

, where π_l is the prior probability of class l, $f(\cdot)$ is the probability density/mass function (multivariate normal for LPA, multinomial for LCA), and $\boldsymbol{\theta}_l$ are class-specific parameters.

AIC Akaike Information Criterion: $\mathrm{AIC} = -2\log\mathcal{L} + 2k$, where $npar$ = number of free parameters. Lower values indicate better fit.

BIC Bayesian Information Criterion: $\mathrm{BIC} = -2\log\mathcal{L} + npar\log(N)$, where N = sample size. Incorporates stronger penalty for complexity than AIC.

SIC Sample-Size Adjusted BIC: $\mathrm{SIC} = -\frac{1}{2}\mathrm{BIC}$. Equivalent to $\log\mathcal{L} - \frac{npar}{2}\log(N)$. Often used in latent class modeling.

CAIC Consistent AIC: $\mathrm{CAIC} = -2\log\mathcal{L} + npar\,[\log(N) + 1]$. Consistent version of AIC that converges to true model as $N \to \infty$.

AWE Approximate Weight of Evidence: $\mathrm{AWE} = -2\log\mathcal{L} + 1.5k\,[\log(N) + 1]$. Penalizes complexity more heavily than CAIC.

SABIC Sample-Size Adjusted BIC: $\mathrm{SABIC} = -2\log\mathcal{L} + npar\log\left(\frac{N+2}{24}\right)$. Recommended for latent class/profile analysis with moderate sample sizes.

Examples

```
# Fit LPA model
set.seed(123)
data.obj <- sim.LPA(N = 100, I = 3, L = 2, constraint = "E0")
fit <- LPA(data.obj$response, L = 2, constraint = "VV", method = "EM")

# Compute fit indices
fit_indices <- get.fit.index(fit)

fit_indices

extract(fit_indices, "SABIC")
```

---

get.Log.Lik.LCA *Calculate Log-Likelihood for Latent Class Analysis*

Description

Computes the log-likelihood of observed categorical data under a Latent Class Analysis (LCA) model given class probabilities and conditional response probabilities. The calculation assumes local independence of responses conditional on latent class membership.

Usage

```
get.Log.Lik.LCA(response, P.Z, par)
```

Arguments

response	A numeric matrix of dimension $N \times I$ containing discrete responses. Values can be any categorical encoding (e.g., 1/2/3, A/B/C, or 0/1). The function automatically:

- Converts all responses to 0-based integer encoding internally
- Determines the maximum number of categories ($K_{\max}$) across indicators

`P.Z` A numeric vector of length L containing prior probabilities for latent classes. Must satisfy:

- $\sum_{l=1}^{L} \pi_l = 1$
- $\pi_l > 0$ for all $l = 1, \ldots, L$

`par` A 3-dimensional array of dimension $L \times I \times K_{\max}$ containing conditional probabilities, where $par[l, i, k]$ represents $P(X_i = k - 1 \mid Z = l)$ (after internal 0-based re-encoding). Must satisfy:

- For each class l and indicator i: $\sum_{k=1}^{K_i} par[l, i, k] = 1$
- Probabilities for non-existent categories (where $k > K_i$) are ignored but must be present in the array

Details

The log-likelihood calculation follows these steps:

- Response Standardization: Original responses are converted to 0-based integers using `adjust.response`. For example, original values {1,2,5} become {0,1,2} (ordered and relabeled sequentially).
- Class-Specific Likelihood: For each observation n and class l, compute:

$$P(\mathbf{X}_n \mid Z_n = l) = \prod_{i=1}^{I} P(X_{ni} = x_{ni} \mid Z_n = l)$$

  where x_{ni} is the standardized response value, and probabilities are taken from `par[l, i, x_{ni}+1]`.
- Marginal Likelihood: For each observation n, combine class-specific likelihoods weighted by class probabilities:

$$P(\mathbf{X}_n) = \sum_{l=1}^{L} \pi_l \cdot P(\mathbf{X}_n \mid Z_n = l)$$

- Log Transformation: Sum log-transformed marginal likelihoods across all observations:

$$\log \mathcal{L} = \sum_{n=1}^{N} \log P(\mathbf{X}_n)$$

Value

A single numeric value representing the total log-likelihood:

$$\log \mathcal{L} = \sum_{n=1}^{N} \log \left[\sum_{l=1}^{L} \pi_l \prod_{i=1}^{I} P(X_{ni} = x_{ni} \mid Z = l) \right]$$

where x_{ni} is the standardized (0-based) response for person n on indicator i.

`get.Log.Lik.LPA` *Calculate Log-Likelihood for Latent Profile Analysis*

Description

Computes the log-likelihood of observed continuous data under a Latent Profile Analysis (LPA) model with multivariate normal distributions within each latent profile. Implements robust numerical techniques to handle near-singular covariance matrices.

Usage

```
get.Log.Lik.LPA(response, P.Z, means, covs, jitter = 1e-10)
```

Arguments

`response`	A numeric matrix of dimension $N \times I$ containing continuous observations. Rows represent observations, columns represent variables. Missing values are not permitted.
`P.Z`	A numeric vector of length L containing prior probabilities for latent profiles. Must satisfy: • $\sum_{l=1}^{L} \pi_l = 1$ • $\pi_l > 0$ for all $l = 1, \ldots, L$
`means`	A matrix of dimension $L \times I$ where row l contains the mean vector $\boldsymbol{\mu}_l$ for profile l.
`covs`	An array of dimension $I \times I \times L$ where slice l contains the covariance matrix $\boldsymbol{\Sigma}_l$ for profile l. Must be symmetric positive semi-definite.
`jitter`	A small positive constant (default: 1e-10) added to diagonal elements of covariance matrices to ensure numerical stability during Cholesky decomposition.

Details

The log-likelihood calculation follows these steps:

- Covariance Stabilization: Each covariance matrix $\boldsymbol{\Sigma}_l$ is symmetrized as $(\boldsymbol{\Sigma}_l + \boldsymbol{\Sigma}_l^\top)/2$. If Cholesky decomposition fails:
  - Add `jitter` to diagonal elements iteratively (up to 10 attempts, scaling jitter by 10x each attempt)
  - Fall back to diagonal covariance matrix if decomposition still fails
- Profile-Specific Density for observation n in profile l:

$$\log f(\mathbf{x}_n \mid Z_n = l) = -\frac{I}{2}\log(2\pi) - \frac{1}{2}\log|\boldsymbol{\Sigma}_l| - \frac{1}{2}(\mathbf{x}_n - \boldsymbol{\mu}_l)^\top \boldsymbol{\Sigma}_l^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_l)$$

  Computed efficiently using Cholesky decomposition $\boldsymbol{\Sigma}_l = \mathbf{R}^\top\mathbf{R}$ where applicable.

- Joint Probability for observation n and profile l:

$$\log[\pi_l \cdot f(\mathbf{x}_n \mid Z_n = l)] = \log(\pi_l) + \log f(\mathbf{x}_n \mid Z_n = l)$$

  $\log(\pi_l)$ uses $\log(\pi_l + 10^{-12})$ to avoid undefined values.

- Marginal Likelihood per observation using log-sum-exp trick for numerical stability:

$$\log f(\mathbf{x}_n) = a_{\max} + \log\left(\sum_{l=1}^{L} \exp\left\{\log[\pi_l \cdot f(\mathbf{x}_n \mid Z_n = l)] - a_{\max}\right\}\right)$$

  where $a_{\max} = \max_l \log[\pi_l \cdot f(\mathbf{x}_n \mid Z_n = l)]$.

- Total Log-Likelihood: Sum of $\log f(\mathbf{x}_n)$ across all observations $n = 1, \ldots, N$.

Value

A single numeric value representing the total log-likelihood:

$$\log \mathcal{L} = \sum_{n=1}^{N} \log\left[\sum_{l=1}^{L} \pi_l \cdot \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)\right]$$

where $\mathcal{N}(\cdot)$ denotes the multivariate normal density function.

Note

Critical implementation details:

- Cholesky Decomposition: For non-degenerate cases ($I > 1$), used to compute: $\log|\boldsymbol{\Sigma}_l| = 2\sum_{i=1}^{I} \log(r_{ii})$ and quadratic form $\|\mathbf{R}^{-\top}(\mathbf{x}_n - \boldsymbol{\mu}_l)\|^2$
- Univariate Handling: When $I = 1$, computes density directly without decomposition
- Numerical Safeguards:
  - Densities clamped to -10^{10} when non-finite
  - Marginal likelihoods clamped to -10^{10} when non-finite
  - Explicit dimension checks for `means` and `covs`
- Assumptions:
  - Multivariate normality within profiles
  - No missing data in `response`
  - Positive-definite covariance matrices (after stabilization)

`get.Log.Lik.LTA` *Calculate Log-Likelihood for Latent Transition Analysis*

Description

Computes the observed-data log-likelihood for a Latent Transition Analysis (LTA) model using the three-step approach with measurement error correction. The likelihood integrates over all possible latent class paths while incorporating classification uncertainty via Classification Error Probability (CEP) matrices. This function is designed to work with parameters estimated from the `LTA` function.

Usage

```
get.Log.Lik.LTA(
  params,
  CEP,
  P.Z.Xns,
  Zs,
  covariates,
  covariates.timeCross = FALSE
)
```

Arguments

`params` A named `list` containing model parameters:

- `beta`: Matrix of size $p_1 \times L$ with coefficients for the initial class membership multinomial logit model (time 1). The coefficient vector for reference class L is constrained to $\boldsymbol{\beta}_L = \mathbf{0}$.
- `gama`: Nested `list` of transition coefficients. For transition to time t (from time $t-1$ to t, where $t = 2, \ldots, T$):

  `gama[[t-1]][[from_class]][[to_class]]` Coefficient vector of length p_t for transition from class `from_class` at time $t-1$ to class `to_class` at time t.

  Coefficients for transitions to reference class L are constrained to zero vectors ($\boldsymbol{\gamma}_{klt} = \mathbf{0}$ when $k = L$).

`CEP` A `list` of $L \times L$ matrices (length = number of time points T). Element (k, l) in `CEP[[t]]` estimates:

$$P(\hat{Z}_{nt} = l \mid Z_{nt} = k)$$

where $\hat{Z}_{nt}$ is the modal class assignment and Z_{nt} is the true latent class. Computed via non-parametric approximation in Step 2 of three-step LTA.

`P.Z.Xns` A `list` of matrices (length = T). Each matrix has dimensions $N \times L$, where element (n, l) is:

$$P(Z_{nt} = l \mid \mathbf{X}_{nt})$$

the posterior probability of individual n belonging to class l at time t from Step 1 latent class/profile analysis.

`Zs` A `list` of integer vectors (length = T). Each vector has length N, where `Zs[[t]][n]` is the modal (most likely) class assignment $\hat{Z}_{nt}$ for individual n at time t.

`covariates` A `list` of design matrices (length = T). For time t, matrix dimension is $N \times p_t$. Must include an intercept column (all 1s) as the first column, i.e., $\mathbf{X}_{nt} = (X_{nt0}, X_{nt1}, \ldots, X_{ntM})^\top$ with $X_{nt0} = 1$. Covariates may differ across time points and between initial status ($t = 1$) and transitions ($t \geq 2$).

`covariates.timeCross`
Logical. If `TRUE`, forces identical transition coefficients across all time points (`gama[[t]]` is copied from `gama[[1]]` for $t > 1$). Default is `FALSE`.

Details

The log-likelihood calculation follows these steps:

1. Latent Path Enumeration: All L^T possible latent class trajectories $\mathbf{z}_n$ are generated and cached.
2. **Initial Class Probabilities (time 1):** For individual n, compute using multinomial logit with covariates $\mathbf{X}_{n1}$:
$$P(Z_{n1} = l \mid \mathbf{X}_{n1}) = \frac{\exp(\boldsymbol{\beta}_l^\top \mathbf{X}_{n1})}{\sum_{k=1}^{L} \exp(\boldsymbol{\beta}_k^\top \mathbf{X}_{n1})}$$
where $\boldsymbol{\beta}_L = \mathbf{0}$ (reference class constraint). Numerical stabilization is applied via subtraction of the maximum linear predictor.
3. Transition Probabilities (times $t \geq 2$): For transition from class k at time $t-1$ to class l at time t:
$$P(Z_{nt} = l \mid Z_{n,t-1} = k, \mathbf{X}_{nt}) = \frac{\exp(\boldsymbol{\gamma}_{klt}^\top \mathbf{X}_{nt})}{\sum_{j=1}^{L} \exp(\boldsymbol{\gamma}_{kjt}^\top \mathbf{X}_{nt})}$$
where $\boldsymbol{\gamma}_{kLt} = \mathbf{0}$ for all k (reference class constraint).
4. Path-Specific Likelihood: For each path $\mathbf{z}_n$ and individual n:
   (a) Compute path probability: $P(Z_{n1} = z_{n1} \mid \mathbf{X}_{n1}) \times \prod_{t=2}^{T} P(Z_{nt} = z_{nt} \mid Z_{n,t-1} = z_{n,t-1}, \mathbf{X}_{nt})$
   (b) Apply CEP weights: $\prod_{t=1}^{T} P(\hat{Z}_{nt} = \hat{z}_{nt} \mid Z_{nt} = z_{nt}) = \prod_{t=1}^{T} \text{CEP}_t(z_{nt}, \hat{z}_{nt})$
   (c) Multiply path probability by CEP weights
5. **Marginalization:** Sum path-specific likelihoods over all L^T paths for each individual n, then sum log-transformed marginal likelihoods across all individuals.

Value

A single numeric value representing the total observed-data log-likelihood:

$$\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log \Bigg[\sum_{\mathbf{z}_n \in \{1,\ldots,L\}^T} \Big(\prod_{t=1}^{T} \text{CEP}_t(z_{nt}, \hat{z}_{nt}) \Big) \cdot P(Z_{n1} = z_{n1} \mid \mathbf{X}_{n1}) \cdot \prod_{t=2}^{T} P(Z_{nt} = z_{nt} \mid Z_{n,t-1} = z_{n,t-1}, \mathbf{X}_{nt}) \Bigg]$$

where $\mathbf{z}_n = (z_{n1}, \ldots, z_{nT})$ is a latent class path, $\hat{z}_{nt} =$ `Zs[[t]][n]` is the modal assignment, and $\boldsymbol{\theta}$ denotes all model parameters (`beta`, `gama`).

Note

When no covariates are included:

- Initial probabilities reduce to $P(Z_{n1} = l) = \pi_l$ (multinomial probabilities)
- Transition probabilities reduce to $P(Z_{nt} = l \mid Z_{n,t-1} = k) = \tau_{kl}^{(t)}$ (time-specific Markov transition probabilities)

See Also

`LTA` for three-step LTA estimation, `get.CEP` for CEP matrix computation

---

get.npar.LCA *Calculate Number of Free Parameters in Latent Class Analysis*

Description

Computes the total number of free parameters in an LCA model based on the number of categories per observed variable and the number of latent classes. This follows standard LCA parameterization with local independence assumption.

Usage

```
get.npar.LCA(poly.value, L)
```

Arguments

`poly.value`	A numeric vector of length I where each element K_i represents the number of response categories for observed variable i.
`L`	Integer specifying the number of latent classes.

Details

Parameter count derivation:

Fixed components (always present):
- Conditional response probabilities: $\sum_{i=1}^{I}(L \times K_i - 1)$ parameters
- Independent class proportions: $L - 1$ parameters (since $\sum_{l=1}^{L} \pi_l = 1$)

Per-variable parameterization: For each observed variable i with K_i categories:

- Each latent class requires K_i conditional probabilities $P(X_i = k|Z = l)$
- With constraints $\sum_{k=1}^{K_i} P(X_i = k|Z = l) = 1$ for each class l
- Global constraints reduce total parameters to $L \times K_i - 1$ per variable

Value

Integer representing the total number of free parameters in the model:

$$\text{npar} = \sum_{i=1}^{I} \underbrace{(L \times K_i - 1)}_{\text{free parameters}} + \underbrace{(L-1)}_{\text{class proportions}}$$

Examples

```
# Example 1: 3 binary variables (K_i=2), 2 latent classes
poly.value <- c(2, 2, 2)  # Three binary variables
L <- 2
npar <- sum(poly.value * L - 1) + (L - 1)  # = (4-1)+(4-1)+(4-1) + 1 = 3+3+3+1 = 10
get.npar.LCA(poly.value, L)  # Returns 10

# Example 2: Mixed variable types (binary, ternary, quaternary)
poly.value <- c(2, 3, 4)  # Variables with 2, 3, and 4 categories
L <- 3
npar <- sum(poly.value * L - 1) + (L - 1)  # = (6-1)+(9-1)+(12-1) + 2 = 5+8+11+2 = 26
get.npar.LCA(poly.value, L)  # Returns 26

# Example 3: Single polytomous variable with 5 categories, 4 latent classes
poly.value <- 5
L <- 4
npar <- sum(poly.value * L - 1) + (L - 1)  # = (20-1) + 3 = 19+3 = 22
get.npar.LCA(poly.value, L)  # Returns 22
```

---

get.npar.LPA *Calculate Number of Free Parameters in Latent Profile Analysis*

Description

Computes the total number of free parameters in an LPA model based on the number of observed variables (I), number of latent profiles (L), and covariance structure constraints.

Usage

```
get.npar.LPA(I, L, constraint = "VV")
```

Arguments

I	Integer specifying the number of continuous observed variables.
L	Integer specifying the number of latent profiles.
constraint	Character string specifying covariance structure constraints. Supported options:
	Univariate case ($I = 1$): "UE" Equal variance across all profiles (1 shared variance parameter).

`"UV"` Varying variances across profiles (L profile-specific variance parameters).

Multivariate case ($I > 1$): `"E0"` Equal variances across profiles, zero covariances. Requires I variance parameters.

`"V0"` Varying variances across profiles, zero covariances. Requires $L \times I$ variance parameters.

`"EE"` Equal variances and equal covariances across profiles (homogeneous covariance matrix). Requires $\frac{I(I+1)}{2}$ parameters.

`"VE"` Varying variances per profile, but *equal covariances* across profiles. Requires $L \times I + \frac{I(I-1)}{2}$ parameters.

`"EV"` Equal variances across profiles, but *varying covariances* per profile. Requires $I + L \times \frac{I(I-1)}{2}$ parameters.

`"VV"` Varying variances and varying covariances across profiles (heterogeneous covariance matrices). Requires $L \times \frac{I(I+1)}{2}$ parameters.

`list` Custom constraints. Each element is a 2-element integer vector specifying variables whose covariance parameters are constrained equal across all classes. The constraint applies to:

- Variances: When both indices are identical (e.g., `c(3,3)` forces variance of variable 3 to be equal across classes)
- Covariances: When indices differ (e.g., `c(1,2)` forces covariance between variables 1 and 2 to be equal across classes)

Constraints are symmetric (e.g., `c(1,2)` automatically constrains `c(2,1)`). All unconstrained parameters vary freely across classes while maintaining positive definiteness.

Default: `"VV"`.

Details

Parameter count breakdown:

1. Fixed components (always present):
   - Profile-specific means: $L \times I$ parameters
   - Independent class proportions: $L - 1$ parameters (since $\sum_{l=1}^{L} \pi_l = 1$)
2. Covariance parameters (varies by constraint):

$I = 1$:
- `"UE"`: 1 shared variance parameter
- `"UV"`: L profile-specific variance parameters

$I > 1$:
- `"E0"`: I shared variance parameters (no covariances)
- `"V0"`: $L \times I$ profile-specific variance parameters (no covariances)
- `"EE"`: $\frac{I(I+1)}{2}$ parameters for one shared full covariance matrix
- `"VE"`: $L \times I$ diagonal variances (free per profile) + $\frac{I(I-1)}{2}$ off-diagonal covariances (shared across profiles)
- `"EV"`: I diagonal variances (shared across profiles) + $L \times \frac{I(I-1)}{2}$ off-diagonal covariances (free per profile)
- `"VV"`: $L \times \frac{I(I+1)}{2}$ parameters for L distinct full covariance matrices

Value

Integer representing the total number of free parameters in the model:

$$\text{npar} = \underbrace{L \times I}_{\text{means}} + \underbrace{(L-1)}_{\text{class proportions}} + \underbrace{\text{covariance parameters}}_{\text{depends on constraint}}$$

Note

Important considerations:

- For $I = 1$, only `"UE"` and `"UV"` are meaningful; `"EE"`, `"E0"`, `"VV"`, `"V0"`, etc., are treated as `"UE"` or `"UV"` respectively.
- Covariance parameters count only free elements in symmetric matrices (diagonal + upper triangle).
- If an user-defined `constraint` is provided, the function defaults to `"VV"` behavior but subtracts $(L-1) \times$ length(constraint).
- `"VE"` and `"EV"` constraints require $I > 1$ to be meaningful (otherwise no covariances exist).

Examples

```
# Univariate examples (I=1)
get.npar.LPA(I = 1, L = 2, constraint = "UE")
get.npar.LPA(I = 1, L = 3, constraint = "UV")

# Multivariate examples (I=3)
get.npar.LPA(I = 3, L = 2, constraint = "E0")
get.npar.LPA(I = 3, L = 2, constraint = "V0")
get.npar.LPA(I = 3, L = 2, constraint = "EE")
get.npar.LPA(I = 3, L = 2, constraint = "VV")
get.npar.LPA(I = 3, L = 2, constraint = "VE")
get.npar.LPA(I = 3, L = 2, constraint = "EV")

# User defined example
get.npar.LPA(I = 3, L = 2, constraint = list(c(1, 2), c(3, 3)))
```

---

get.npar.LTA *Calculate Number of Free Parameters in Latent Transition Analysis*

Description

Computes the total number of free parameters in a Latent Transition Analysis (LTA) model estimated via the three-step approach. The count depends on the number of latent classes, the number of time points, the number of covariates at each time point, and whether transition coefficients are constrained to be equal across time.

Usage

```
get.npar.LTA(covariates.ncol, L, covariates.timeCross = FALSE)
```

Arguments

`covariates.ncol`
: An integer vector of length T (number of time points). Each element M_t represents the number of covariates (columns) for time point t. Must include an intercept column (all 1s) as the first covariate.

`L`
: Integer scalar. Number of latent classes ($L \geq 2$).

`covariates.timeCross`
: Logical. If `TRUE`, transition coefficients are constrained to be identical across all transitions (time-invariant effects). This requires that the number of covariates is the same for all time points after the first (i.e., $M_2 = M_3 = \cdots = M_T$). If `FALSE` (default), each transition has its own set of coefficients.

Details

Parameterization:

Initial status model (time 1): Multinomial logit model with L classes (last class is reference). Number of free parameters: $M_1 \times (L-1)$.

Transition models (time $t \to t+1$): For each transition, a multinomial logit model conditioned on previous class. For each origin class k and destination class l ($l \neq L$), there is a coefficient vector of length M_{t+1}. Total per transition: $L \times (L-1) \times M_{t+1}$ parameters. The constraint `covariates.timeCross` determines whether these parameters are shared across transitions.

Value

Integer representing the total number of free parameters:

$$npar = M_1 \times (L-1) + \begin{cases} L \times (L-1) \times M_2 & \text{if } T > 1 \text{ and time-invariant effects} \\ \sum_{t=2}^{T} L \times (L-1) \times M_t & \text{if } T > 1 \text{ and time-varying effects} \\ 0 & \text{if } T = 1 \end{cases}$$

where:

- *time-invariant effects* corresponds to `covariates.timeCross = TRUE`
- *time-varying effects* corresponds to `covariates.timeCross = FALSE`

Note

Critical assumptions:

- The last latent class (L) is always the reference category for all logits.
- When `covariates.timeCross = TRUE`, it is assumed that all time points after the first have identical covariate structures ($M_2 = M_3 = \cdots = M_T$). If violated, the function uses M_1 for all transitions to match `LTA`'s internal behavior.
- For $T = 1$, no transition parameters are estimated (pure latent class/profile analysis).

Examples

```
# Example 1: 2 time points, 2 classes, time-invariant transition coefficients
#   Time1: 2 covariates (intercept + 1 predictor)
#   Time2: 3 covariates (but constrained to match Time1 due to timeCross=TRUE)
covariates.ncol <- c(2, 3)
L <- 2
get.npar.LTA(covariates.ncol, L, covariates.timeCross = TRUE)

# Example 2: Same as above but time-varying coefficients
get.npar.LTA(covariates.ncol, L, covariates.timeCross = FALSE)

# Example 3: 3 time points, 3 classes, time-invariant coefficients
covariates.ncol <- c(2, 2, 2)  # All time points have identical covariates
L <- 3
get.npar.LTA(covariates.ncol, L, covariates.timeCross = TRUE)

# Example 4: 3 time points, 3 classes, time-varying coefficients
covariates.ncol <- c(2, 3, 4)
L <- 3
get.npar.LTA(covariates.ncol, L, covariates.timeCross = FALSE)

# Example 5: Single time point (equivalent to LCA)
covariates.ncol <- c(3)
L <- 4
get.npar.LTA(covariates.ncol, L)
```

---

get.P.Z.Xn.LCA *Compute Posterior Latent Class Probabilities Based on Fixed Parameters*

Description

Computes posterior probabilities of latent class membership for each observation using fixed conditional response probabilities (`par`) and fixed class prior probabilities (`P.Z`).

Usage

```
get.P.Z.Xn.LCA(response, par, P.Z)
```

Arguments

`response`	Numeric matrix ($N \times I$) of categorical responses. Categories are automatically remapped to 0-based integers internally via `adjust.response`.
`par`	3D array ($L \times I \times K_{\max}$) of fixed conditional response probabilities where: • L = number of latent classes • I = number of indicators • $K_{\max}$ = maximum categories across indicators

`par[l, i, k]` = $P(X_i = k - 1 \mid Z = l)$ (using 1-based indexing for the array dimension corresponding to category k-1).

`P.Z` Vector of length L with fixed class prior probabilities (π_l). These values are used directly without re-estimation.

Details

Unlike an EM algorithm, this function does NOT iteratively update class prevalences. It performs a single calculation step based on Bayes' theorem:

$$P(Z_n = l \mid \mathbf{X}_n) = \frac{\pi_l \prod_{i=1}^{I} P(X_{ni} = x_{ni} \mid Z_n = l)}{\sum_{k=1}^{L} \pi_k \prod_{i=1}^{I} P(X_{ni} = x_{ni} \mid Z_n = k)}$$

where π_l are the fixed priors provided in the `P.Z` argument.

Value

Numeric matrix ($N \times L$) of posterior probabilities. Rows sum to 1. Columns are named "Class.1", "Class.2", etc.

Examples

```
library(LCPA)
set.seed(123)
# Simulate data
data.obj <- sim.LCA(N = 200, I = 3, L = 2, IQ = 0.85)

# Fit a model to get parameters
fit <- LCA(data.obj$response, L = 2, method = "EM", nrep = 5)

# Calculate posteriors using fixed parameters from the fitted model
P.Z.Xn <- get.P.Z.Xn.LCA(
  response = data.obj$response,
  par = fit$params$par,
  P.Z = fit$params$P.Z
)
head(P.Z.Xn)
```

---

get.P.Z.Xn.LPA *Compute Posterior Latent Profile Probabilities Based on Fixed Parameters*

Description

Computes posterior probabilities of latent profile membership for each observation using fixed profile parameters (means, covariances) and fixed prior probabilities.

Usage

```
get.P.Z.Xn.LPA(response, means, covs, P.Z)
```

Arguments

`response`	Numeric matrix ($N \times I$) of continuous responses. Missing values are not allowed. Data should typically be standardized prior to analysis.
`means`	Numeric matrix ($L \times I$) of fixed profile means where: • L = number of latent profiles • I = number of observed variables Row l contains profile-specific means $\boldsymbol{\mu}_l$.
`covs`	3D array ($I \times I \times L$) of fixed profile covariance matrices where: • `covs[, , l]` = profile-specific covariance matrix $\boldsymbol{\Sigma}_l$ Each slice must be symmetric and positive definite.
`P.Z`	Vector of length L with fixed profile prior probabilities (π_l). These values are used directly without re-estimation.

Details

Unlike an EM algorithm, this function does NOT iteratively update profile prevalences. It performs a single E-step calculation:

$$P(Z_n = l \mid \mathbf{x}_n) = \frac{\pi_l \cdot \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}{\sum_{k=1}^{L} \pi_k \cdot \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}$$

where π_l are the fixed priors provided in the `P.Z` argument.

Value

Numeric matrix ($N \times L$) of posterior probabilities. Rows sum to 1. Columns are named "Class.1", "Class.2", etc.

Examples

```
library(LCPA)
set.seed(123)
data.obj <- sim.LPA(N = 300, I = 2, L = 2, constraint = "VV")
fit <- LPA(data.obj$response, L = 2, method = "EM", nrep = 5)

# Calculate posteriors using fixed parameters from a fitted model
P.Z.Xn <- get.P.Z.Xn.LPA(
  response = data.obj$response,
  means = fit$params$means,
  covs = fit$params$covs,
  P.Z = fit$params$P.Z
)
head(P.Z.Xn)
```

get.SE *Compute Standard Errors*

Description

Computes approximate standard errors (SEs) for estimated parameters in Latent Class Analysis (LCA) or Latent Profile Analysis (LPA) models using two methods:

- "Bootstrap": Non-parametric bootstrap with label-switching correction. McLachlan & Peel (2004) suggest that 50–100 replicates often provide adequate accuracy for practical purposes, though more (e.g., 500–1000) may be preferred for publication-quality inference.
- "Obs": Numerical evaluation of the observed information matrix (Hessian of negative log-likelihood)

Users should note that get.SE computes standard errors based on the observed information matrix via numerical differentiation, which may lack precision and often yields ill-conditioned matrices. Therefore, we recommend using method = "Bootstrap".

Usage

```
get.SE(object, method = "Bootstrap", n.Bootstrap = 100, vis = TRUE)
```

Arguments

object	An object of class "LCA" or "LPA" returned by LCA or LPA.
method	Character specifying SE calculation method: "Obs" or "Bootstrap" (default).
n.Bootstrap	Integer. Number of bootstrap replicates when method="Bootstrap" (default=100).
vis	Logical. If TRUE, displays progress information during estimation (default: TRUE).

Value

A list of class "SE" containing:

se Named list of SEs matching parameter structure of input model:

- LPA: means (matrix: classes x variables), covs (array: vars x vars x classes), P.Z (vector: class prob SEs)
- LCA: par (array: classes x indicators x categories), P.Z (vector: class prob SEs)
- Critical Note for "Obs" method: Only *free parameters* have non-zero SEs. Non-free parameters (e.g., last class probability in P.Z due to sum-to-1 constraint; last category probability in LCA indicators) have SE=0. Bootstrap provides SEs for all parameters.

vcov NULL for bootstrap. For "Obs": variance-covariance matrix (may be regularized). Diagonal contains squared SEs of free parameters.

hessian NULL for bootstrap. For "Obs": observed information matrix (pre-regularization). Dimension = number of free parameters.

diagnostics Method-specific diagnostics:

- Bootstrap: `n.Bootstrap.requested`, `n.Bootstrap.completed`
- Obs: Hessian computation details, condition number, regularization status, step sizes

`call` Function call that generated the object

`arguments` List of input arguments

References

McLachlan, G. J., & Peel, D. (2004). Finite Mixture Models. Wiley. https://books.google.com.sg/books?id=c2_fAox0DQoC

Examples

```
library(LCPA)
set.seed(123)

# LPA with Bootstrap (minimal replicates for example)
lpa_data <- sim.LPA(N = 500, I = 4, L = 3)
lpa_fit <- LPA(lpa_data$response, L = 3)
se_boot <- get.SE(lpa_fit, method = "Bootstrap", n.Bootstrap = 10)

print(se_boot)
extract(se_boot, "covs")


# LCA with Observed Information (note zeros for constrained parameters)
lca_data <- sim.LCA(N = 500, I = 4, L = 3, poly.value = 5)
lca_fit <- LCA(lca_data$response, L = 3)
se_obs <- get.SE(lca_fit, method = "Obs")

print(se_obs)
extract(se_obs, "par")
```

`install_python_dependencies` *Install Required Python Dependencies for Neural Latent Variable Models*

Description

Checks whether essential Python packages required to run neural latent variable models (e.g., LCAnet, LPAnet) are installed in the current Python environment. If any are missing, the user is interactively prompted to install them via `reticulate::py_install()`. The targeted packages are:

- `numpy` — Fundamental package for numerical computing in Python.
- `torch` — PyTorch deep learning framework (supports CPU/GPU computation).
- `matplotlib` — 2D plotting and visualization library.

- `scikit-learn` — Machine learning utilities.
- `scipy` — Scientific computing and advanced linear algebra routines.
- `six` — Python 3 compatibility library.

For `torch`, users can choose between CPU-only or GPU-enabled versions (with CUDA support). Available CUDA versions are filtered by OS compatibility.

This function is especially useful when deploying models that bridge R and Python via **reticulate**, ensuring all backend dependencies are met before model execution.

Usage

```
install_python_dependencies()
```

Details

The function performs the following steps for each dependency:

1. Uses `reticulate::py_module_available()` to test if the module is importable.
2. If not available, prints a message describing the package's purpose.
3. Prompts the user interactively (via `readline`) whether to proceed with installation.
4. For `torch`, offers CPU/GPU choice and CUDA version selection if GPU is chosen.
5. Installs the package using `reticulate::py_install()` with appropriate index URL if needed.
6. Returns a logical list indicating initial installation status of each package.

Note: This function requires **reticulate** to be loaded and a valid Python environment configured. It does NOT automatically install **reticulate** or configure Python — that must be done separately.

Value

A named list of logical values indicating whether each package was already installed before running this function:

`numpy_installed`
: Logical. Was `numpy` already available?

`torch_installed`
: Logical. Was `torch` already available?

`matplotlib_installed`
: Logical. Was `matplotlib` already available?

`sklearn_installed`
: Logical. Was `scikit-learn` already available?

`scipy_installed`
: Logical. Was `scipy` already available?

`six_installed`
: Logical. Was `six` already available?

Examples

```
library(reticulate)

# Ensure reticulate is loaded and Python is configured
# need python
## Not run:
# Run dependency installer
deps <- install_python_dependencies()

# Check which were missing
print(deps)

## End(Not run)
```

`Kmeans.LCA` *Initialize LCA Parameters via K-means Clustering*

Description

Performs hard clustering of observations using K-means algorithm to generate initial parameter estimates for Latent Class Analysis (LCA) models. This provides a data-driven initialization strategy that often outperforms random starts when the number of observed categorical variables I is large (i.e., $I > 50$).

Usage

```
Kmeans.LCA(response, L, nrep = 10)
```

Arguments

`response`	A numeric matrix of dimension $N \times I$, where N is the number of observations and I is the number of observed categorical variables. Each column must contain nominal-scale discrete responses (e.g., integers representing categories). Non-sequential category values are automatically re-encoded to sequential integers starting from 1.
`L`	Integer specifying the number of latent classes. Must be $2 \leq L < N$.
`nrep`	Integer specifying the number of random starts for K-means algorithm (default: 10). The solution with the lowest within-cluster sum of squares is retained.

Details

The function executes the following steps:

- Data preprocessing: Automatically adjusts non-sequential category values to sequential integers (e.g., categories {1,3,5} become {1,2,3}) using internal adjustment routines.
- K-means clustering: Scales variables to mean=0 and SD=1 before clustering. Uses Lloyd's algorithm with Euclidean distance.

- Parameter estimation:
  - For each cluster l, computes empirical response probabilities $P(X_i = k|Z = l)$ for all indicators i and categories k.
  - Handles singleton clusters by assigning near-deterministic probabilities (e.g., $1 - 10^{-10}$ for observed category, 10^{-10} for others).
- Posterior probabilities: Constructs hard-classification matrix where $P(Z = l|\mathbf{X}_n) = 1$ for the assigned cluster and 0 otherwise.

Value

A list containing:

`params` List of initialized parameters:

`par` An $L \times I \times K_{\max}$ array of initial conditional probabilities, where $K_{\max}$ is the maximum number of categories across indicators. Dimension order: latent classes (1:L), indicators (1:I), response categories (1:K_max).

`P.Z` Numeric vector of length L containing initial class prior probabilities derived from cluster proportions.

`P.Z.Xn` An $N \times L$ matrix of posterior class probabilities. Contains hard assignments (0/1 values) based on K-means cluster memberships.

Note

- Requires at least one observation per cluster. If a cluster has only one observation, probabilities are set to avoid zero values (using 10^{-10}) for numerical stability.
- Data scaling is applied internally. Variables with zero variance are automatically excluded from clustering.
- This function is primarily designed as an initialization method for `LCA` and not intended for final model estimation.

Examples

```
# Simulate response data
set.seed(123)
response <- matrix(sample(1:4, 200, replace = TRUE), ncol = 5)

# Generate K-means initialization for 3-class LCA
init_params <- Kmeans.LCA(response, L = 3, nrep = 5)

# Inspect initial class probabilities
print(init_params$params$P.Z)
```

LCA *Fit Latent Class Analysis Models*

Description

This function estimates parameters of a Latent Class Analysis (LCA; Hagenaars & McCutcheon, 2002) model using either the Expectation-Maximization (EM) algorithm or Neural Network Estimation (NNE). It supports flexible initialization strategies and provides comprehensive model diagnostics.

Usage

```
LCA(
  response,
  L = 2,
  par.ini = "random",
  method = "EM",
  is.sort = TRUE,
  nrep = 20,
  starts = 100,
  maxiter.wa = 20,
  vis = TRUE,
  control.EM = NULL,
  control.Mplus = NULL,
  control.NNE = NULL
)
```

Arguments

`response`	A numeric matrix of dimension $N \times I$, where N is the number of individuals/participants/observations and I is the number of observed categorical items/variables/indicators. Each column must contain nominal-scale discrete responses (e.g., integers representing categories).
`L`	Integer specifying the number of latent classes (default: 2).
`par.ini`	Specification for parameter initialization. Options include: • `"random"`: Completely random initialization (default). • `"kmeans"`: Initializes parameters via K-means clustering on observed data (McLachlan & Peel, 2004). • A `list` containing: `par` An $L \times I \times K_{\max}$ array of initial conditional probabilities for each latent class, indicator, and response category (where $K_{\max}$ is the maximum number of categories across indicators). `P.Z` A numeric vector of length L specifying initial prior probabilities for latent classes.
`method`	Character string specifying estimation algorithm:

- "EM": Expectation-Maximization algorithm (default).
- "NNE": Neural Network Estimation with transformer architecture (experimental; uses transformer + simulated annealing, more reliable than both "EM" and "Mplus"). See install_python_dependencies.
- "Mplus": Calls external Mplus software for estimation. Uses Mplus defaults for optimization unless overridden by control.Mplus.

is.sort — A logical value. If TRUE (Default), the latent classes will be ordered in descending order according to P.Z. All other parameters will be adjusted accordingly based on the reordered latent classes.

nrep — Integer controlling replication behavior:

- If par.ini = "random", number of random initializations.
- If par.ini = "kmeans", number of K-means runs for initialization.
- For method="Mplus", controls number of random starts in Mplus via STARTS option.
- Best solution is selected by log-likelihood/BIC across replications.
- Ignored for user-provided initial parameters.

starts — Number of random initializations to explore during warm-up phase (default: 100).

maxiter.wa — Maximum number of training iterations allowed per warm-up run. After completion, the top nrep solutions (by log-likelihood) are promoted to final training phase (default: 20).

vis — Logical. If TRUE, displays progress information during estimation (default: TRUE).

control.EM — List of control parameters for EM algorithm:

maxiter Maximum iterations (default: 2000).

tol Convergence tolerance for log-likelihood difference (default: 1e-4).

control.Mplus — List of control parameters for Mplus estimation:

maxiter Maximum iterations for Mplus optimization (default: 2000).

tol Convergence tolerance for log-likelihood difference (default: 1e-4).

files.path A character string specifying the directory path where Mplus will write its intermediate files (e.g., .inp model input, .dat data file, .out output, and saved posterior probabilities). This argument is **required** — if NULL (the default), the function throws an error. The specified directory must exist and be writable; if it does not exist, the function attempts to create it recursively. A unique timestamped subdirectory (e.g., "Mplus_LCA_YYYY-MM-DD_HH-MM-SS") will be created within this path to store all run-specific files and avoid naming conflicts. If it is an empty string (""), the timestamped subdirectory "Mplus_LCA_YYYY-MM-DD_HH-MM-SS" will be created directly under R's current working directory (getwd()).

files.clean Logical. If TRUE (default), all intermediate files and the temporary working directory created for this run are deleted upon successful completion or error exit (via on.exit()). If FALSE, all generated files are retained in files.path (or the auto-generated temp dir) for inspection or debugging. Note: when files.path = NULL, even if files.clean = FALSE, the temporary directory may still be cleaned up by the system later — for guaranteed persistence, specify a custom files.path.

`control.NNE` List of control parameters for NNE algorithm:

- `hidden.layers` Integer vector specifying layer sizes in fully-connected network (default: `c(16,16)`).
- `activation.function` Activation function (e.g., `"tanh"`, default: `"tanh"`).
- `use.attention` Whether to enable the self-attention mechanism (i.e., transformer encoder) (default: `TRUE`).
- `d.model` Dimensionality of transformer encoder embeddings (default: 8).
- `nhead` Number of attention heads in transformer (default: 2).
- `dim.feedforward` Dimensionality of transformer feedforward network (default: 16).
- `eps` Small constant for numerical stability (default: 1e-8).
- `lambda` A factor for slight regularization of all parameters (default: 1e-5).
- `initial.temperature` Initial temperature for simulated annealing (default: 1000).
- `cooling.rate` Cooling rate per iteration in simulated annealing (default: 0.5).
- `maxiter.sa` Maximum iterations for simulated annealing (default: 1000).
- `threshold.sa` Minimum temperature threshold for annealing (default: 1e-10).
- `maxiter` Maximum training epochs (default: 1000).
- `maxiter.early` Patience parameter for early stopping (default: 50).
- `maxcycle` Maximum cycles for optimization (default: 10).
- `lr` Learning rate, controlling the step size of neural network parameter updates (default: 0.025).
- `scheduler.patience` Patience for learning rate decay (if the loss function does not improve for more than `patience` consecutive epochs, the learning rate will be reduced) (default: 10).
- `scheduler.factor` Learning rate decay factor; the new learning rate equals the original learning rate multiplied by `scheduler.factor` (default: 0.70).
- `plot.interval` Interval (in epochs) for plotting training diagnostics (default: 100).
- `device` Specifies the hardware device; can be `"CPU"` (default) or `"GPU"`. If the GPU is not available, it automatically falls back to CPU.

Value

An object of class `"LCA"` containing:

`params` List with estimated parameters:

- `par` $L \times I \times K_{\max}$ array of conditional response probabilities per latent class.
- `P.Z` Vector of length L with latent class prior probabilities.

`npar` Number of free parameters in the model. see `get.npar.LCA`

`Log.Lik` Log-likelihood of the final model. see `get.Log.Lik.LCA`

`AIC` Akaike Information Criterion value.

`BIC` Bayesian Information Criterion value.

`best_BIC` Best BIC value across `nrep` runs (if applicable).

`P.Z.Xn` $N \times L$ matrix of posterior class probabilities for each observation.

`P.Z` Vector of length L containing the prior probabilities/structural parameters/proportions for each latent class.

`Z` Vector of length N with MAP-classified latent class memberships.

`probability` Same as `params$par` (redundant storage for convenience).

`Log.Lik.history` Vector tracking log-likelihood at each EM iteration.

`Log.Lik.nrep` Vector of log-likelihoods from each replication run.

`model` The optimal neural network model object (only for `method="NNE"`). Contains the trained transformer architecture corresponding to `best_loss`. This object can be used for further predictions or model inspection.

`arguments` A list containing all input arguments

EM Algorithm

When `method = "EM"`, parameters are estimated via the Expectation-Maximization algorithm, which iterates between:

- **E-step:** Compute posterior class probabilities given current parameters:

$$P(Z_n = l \mid \mathbf{X}_n) = \frac{\pi_l \prod_{i=1}^{I} P(X_{ni} = x_{ni} \mid Z_n = l)}{\sum_{k=1}^{L} \pi_k \prod_{i=1}^{I} P(X_{ni} = x_{ni} \mid Z_n = k)}$$

  where x_{ni} is the standardized (0-based) response for person n on indicator i (see `adjust.response`).

- **M-step:** Update parameters by maximizing expected complete-data log-likelihood:
  - Class probabilities: $\pi_l^{\text{new}} = \frac{1}{N} \sum_{n=1}^{N} P(Z_n = l \mid \mathbf{X}_n)$
  - Conditional probabilities: $P(X_i = k \mid Z = l)^{\text{new}} = \frac{\sum_{n:x_{ni}=k} P(Z_n=l|\mathbf{X}_n)}{\sum_{n=1}^{N} P(Z_n=l|\mathbf{X}_n)}$

- **Convergence**: Stops when $|\log \mathcal{L}^{(t)} - \log \mathcal{L}^{(t-1)}| < \texttt{tol}$ or maximum iterations reached.

Neural Network Estimation (NNE)

When `method = "NNE"`, parameters are estimated using a hybrid neural network architecture that combines feedforward layers with transformer-based attention mechanisms. This approach jointly optimizes profile parameters and posterior probabilities through stochastic optimization enhanced with simulated annealing. See `install_python_dependencies`. Key components include:

Architecture:

Input Representation Observed categorical responses are converted to 0-based integer indices per indicator (not one-hot encoded). For example, original responses $[1, 2, 4]$ become $[0, 1, 2]$.

Feature Estimator (Feedforward Network) A fully-connected neural network with layer sizes specified by `hidden.layers` and activation function `activation.function` processes the integer-indexed responses. This network outputs unnormalized logits for posterior class membership ($N \times L$ matrix).

Attention Refiner (Transformer Encoder) A transformer encoder with `nhead` attention heads that learns latent class prior probabilities $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_L)$ directly from observed responses.

- Input: `response` matrix ($N \times I$), where N = observations, I = continuous variables.

- Mechanism: Self-attention dynamically weighs variable importance during profile assignment, capturing complex multivariate interactions.
- Output: Class prior vector $\boldsymbol{\pi}$ computed as the mean of posteriors:

$$\pi_l = \frac{1}{N} \sum_{n=1}^{N} attention(\mathbf{X}_n)$$

This ensures probabilistic consistency with the mixture model framework.

Profile Parameter Estimation Global conditional probability parameters ($P(X_i = k \mid Z = l)$) are stored as learnable parameters `par` (an $L \times I \times K_{\max}$ tensor). A *masked softmax* is applied along categories to enforce:

- Probabilities sum to 1 within each indicator-class pair
- Non-existent categories (beyond indicator's actual max response) are masked to zero probability

Optimization Strategy:

- **Hybrid Training Protocol**: Alternates between:
  - *Gradient-based phase*: AdamW optimizer minimizes negative log-likelihood with weight decay regularization:
  $$-\log \mathcal{L} + \lambda \|\boldsymbol{\theta}\|_2^2$$
  where λ is controlled by `lambda` (default: 1e-5). Learning rate decays adaptively when loss plateaus (controlled by `scheduler.patience` and `scheduler.factor`).
  - *Simulated annealing phase*: After gradient-based early stopping (`maxiter.early`), parameters are perturbed with noise scaled by temperature:
  $$\theta_{\text{new}} = \theta_{\text{current}} + \mathcal{N}(0, \theta_{\text{current}} \times \frac{T}{T_0})$$
  Temperature T decays geometrically ($T \leftarrow T \times$cooling.rate) from `initial.temperature` until `threshold.sa` is reached. This escapes poor local minima.

  Each full cycle (gradient descent + annealing) repeats up to `maxcycle` times.
- **Model Selection**: Across `nrep` random restarts (using Dirichlet-distributed initializations or K-means), the solution with lowest BIC is retained.
- **Diagnostics**: Training loss, annealing points, and global best solution are plotted when `vis=TRUE`.

Mplus

When `method = "Mplus"`, estimation is delegated to external Mplus software. The function automates the entire workflow:

Workflow:

Temporary Directory Setup Creates `inst/Mplus` to store:

- Mplus input syntax (`.inp`)
- Data file in Mplus format (`.dat`)
- Posterior probabilities output (`.dat`)

Files are automatically deleted after estimation unless `control.Mplus$clean.files = FALSE`.

Syntax Generation Constructs Mplus syntax with:

- `CLASSES = c1(L)` specification for L latent classes
- `CATEGORICAL` declaration for all indicator variables
- `ANALYSIS` block with optimization controls:
  `TYPE = mixture` Standard mixture modeling setup
  `STARTS = starts nrep` Random `starts` and final stage optimizations
  `STITERATIONS = maxiter.wa` max itertions during `starts`.
  `MITERATIONS = maxiter` Maximum EM iterations
  `CONVERGENCE = tol` Log-likelihood convergence tolerance
- `MODEL` block with `%OVERALL%`

Execution Calls Mplus via `MplusAutomation::mplusModeler()`, which:

- Converts R data to Mplus-compatible format with automatic recoding
- Invokes Mplus executable (requires valid license and system PATH configuration)

References

Hagenaars, J. A. , & McCutcheon, A. L. (2002). Applied Latent Class Analysis. United Kingdom: Cambridge University Press.

McLachlan, G. J., & Peel, D. (2004). Finite Mixture Models. Wiley. https://books.google.com.sg/books?id=c2_fAox0DQoC

Examples

```
library(LCPA)

# Example with simulated data
set.seed(123)
data.obj <- sim.LCA(N = 500, I = 4, L = 2, IQ=0.9)
response <- data.obj$response

# Fit 2-class model with EM algorithm

fit.em <- LCA(response, L = 2, method = "EM", nrep = 10)


# Fit 2-profile model using Mplus
# need Mplus
# NOTE: 'files.path' in control.Mplus is REQUIRED — function will error if not provided.
# Example creates a timestamped subfolder (e.g., "Mplus_LCA_YYYY-MM-DD_HH-MM-SS") under './'
# to store all temporary Mplus files (.inp, .dat, .out, etc.).
## Not run:
fit.mplus <- LCA(response, L = 2, method = "Mplus", nrep = 3,
                 control.Mplus = list(files.path = ""))

## End(Not run)

# Fit 2-class model with neural network estimation
# need Python
```

```
## Not run:
fit.nne <- LCA(response, L = 2, method = "NNE", nrep = 3)

## End(Not run)
```

LCPA *Latent Class/Profile Analysis with Covariates*

Description

Implements the three-step estimation method (Vermunt, 2010; Liang et al., 2023) for latent class/profile analysis with covariates, treating latent class membership as an observed variable with measurement error. This is mathematically equivalent to a latent transition analysis (LTA) with `times=1`.

Usage

```
LCPA(
  response,
  L = 2,
  ref.class = L,
  type = "LCA",
  covariate = NULL,
  CEP.error = TRUE,
  par.ini = "random",
  params = NULL,
  is.sort = TRUE,
  constraint = "VV",
  method = "EM",
  tol = 1e-04,
  lower = -10,
  upper = 10,
  method.SE = "Bootstrap",
  n.Bootstrap = 100,
  maxiter = 5000,
  nrep = 20,
  starts = 100,
  maxiter.wa = 20,
  vis = TRUE,
  control.EM = NULL,
  control.Mplus = NULL,
  control.NNE = NULL
)
```

Arguments

`response`	A matrix or data frame of observed responses. Rows of the matrix represent individuals/participants/observations (N), columns of the matrix represent observed

indicators/items/variables (I). For `type = "LCA"`: indicators must be binary or categorical (coded as integers starting from 0). For `type = "LPA"`: indicators must be continuous (numeric), and the response matrix must be standardized using `scale` or `normalize` prior to input.

`L` Integer scalar. Number of latent classes/profiles. Must satisfy $L \geq 2$.

`ref.class` Integer $L \geq ref.class \geq 1$. Specifies which latent class to use as the reference category. Default is `L` (last class). Coefficients for the reference class are fixed to zero. When `is.sort=TRUE`, classes are first ordered by decreasing `P.Z` (class 1 has highest probability), then `ref.class` refers to the position in this sorted order.

`type` Character string. Specifies the type of latent variable model for Step 1:

- `"LCA"` — Latent Class Analysis for categorical indicators.
- `"LPA"` — Latent Profile Analysis for continuous indicators.

See `LCA` and `LPA` for details.

`covariate` Optional. A matrix or data frame of covariates for modeling latent class membership. Must include an intercept column (all 1s) as the first column. If `NULL` (default), only intercept terms are used (i.e., no covariates). Dimension is $N \times p$ where p is the number of covariates including intercept.

`CEP.error` Logical. If `TRUE` (recommended), incorporates classification uncertainty via estimated Classification Error Probability (`get.CEP`) matrices from Step 1. If `FALSE`, uses identity CEP matrices (equivalent to naive modal assignment; introduces bias).

`par.ini` Specification for parameter initialization. Options include:

- `"random"`: Completely random initialization (default).
- `"kmeans"`: Initializes parameters via K-means clustering on observed data (McLachlan & Peel, 2004).
- A `list` for LCA containing:
  - `par` An $L \times I \times K_{\max}$ array of initial conditional probabilities for each latent class, indicator, and response category (where $K_{\max}$ is the maximum number of categories across indicators).
  - `P.Z` A numeric vector of length L specifying initial prior probabilities for latent classes.
- A `list` for LPA containing:
  - `means` An $L \times I$ matrix of initial mean vectors for each profile.
  - `covs` An $I \times I \times L$ array of initial covariance matrices for each profile.
  - `P.Z` A numeric vector of length L specifying initial prior probabilities for profiles.

`params` Optional `list` of pre-estimated Step 1 parameters. If `NULL` (default), Step 1 models are estimated internally. If provided, no LCA or LPA parameter estimation will be performed; instead, the parameters provided in `params` will be used as fixed values. Additionally, `params` must contain:

- A `list` for LCA containing:
  - `par` An $L \times I \times K_{\max}$ array of initial conditional probabilities for each latent class, indicator, and response category (where $K_{\max}$ is the maximum number of categories across indicators).

`P.Z` A numeric vector of length L specifying initial prior probabilities for latent classes.

- A `list` for LPA containing:

  `means` An $L \times I$ matrix of initial mean vectors for each profile.

  `covs` An $I \times I \times L$ array of initial covariance matrices for each profile.

  `P.Z` A numeric vector of length L specifying initial prior probabilities for profiles.

`is.sort` A logical value. If `TRUE` (Default), the latent classes will be ordered in descending order according to `P.Z`. All other parameters will be adjusted accordingly based on the reordered latent classes.

`constraint` Character (LPA only). Specifies structure of within-class covariance matrices:

- `"VV"` — Class-varying variances and covariances (unconstrained; default).
- `"EE"` — Equal variances and covariances across all classes (homoscedastic).

`method` Character. Estimation algorithm for Step 1 models:

- `"EM"` — Expectation-Maximization (default; robust and widely used).
- `"Mplus"` — Interfaces with Mplus software (requires external installation).
- `"NNE"` — Neural Network Estimator (experimental; uses transformer + simulated annealing, more reliable than both `"EM"` and `"Mplus"`).

`tol` Convergence tolerance for log-likelihood difference (default: 1e-4).

`lower` The upper bound for the estimation of regression coefficients, default is -10

`upper` The lower bound for the estimation of regression coefficients, default is 10

`method.SE` Character. Method for estimating standard errors of parameter estimates:

- `"Obs"` — Approximates the observed information matrix via numerical differentiation (Richardson's method). Standard errors are obtained from the inverse Hessian. May fail or be unreliable in small samples or with complex likelihood surfaces.
- `"Bootstrap"` — Uses nonparametric bootstrap resampling to estimate empirical sampling variability. More robust to model misspecification and small-sample bias. Computationally intensive but recommended when asymptotic assumptions are questionable.

Default is `"Bootstrap"`.

`n.Bootstrap` Integer. Number of bootstrap replicates used when `method.SE = "Bootstrap"`. Default is 100. McLachlan & Peel (2004) suggest that 50–100 replicates often provide adequate accuracy for practical purposes, though more (e.g., 500–1000) may be preferred for publication-quality inference. Each replicate involves re-estimating the full three-step LTA model on a resampled dataset.

`maxiter` Maximum number of iterations for optimizing the regression coefficients. Default: 5000.

`nrep` Integer controlling replication behavior:

- If `par.ini = "random"`, number of random initializations.
- If `par.ini = "kmeans"`, number of K-means runs for initialization.

- For `method="Mplus"`, controls number of random starts in Mplus via `STARTS` option.
- Best solution is selected by log-likelihood/BIC across replications.
- Ignored for user-provided initial parameters.

`starts` Number of random initializations to explore during warm-up phase (default: 100).

`maxiter.wa` Maximum number of training iterations allowed per warm-up run. After completion, the top `nrep` solutions (by log-likelihood) are promoted to final training phase (default: 20).

`vis` Logical. If `TRUE`, displays progress information during estimation (default: `TRUE`).

`control.EM` List of control parameters for EM algorithm:

`maxiter` Maximum iterations (default: 2000).

`tol` Convergence tolerance for log-likelihood difference (default: 1e-4).

`control.Mplus` List of control parameters for Mplus estimation:

`maxiter` Maximum iterations for Mplus optimization (default: 2000).

`tol` Convergence tolerance for log-likelihood difference (default: 1e-4).

`files.path` Character string specifying the directory path where Mplus will write its intermediate files (e.g., `.inp` model input, `.dat` data file, `.out` output, and saved posterior probabilities). This argument is **required** — if `NULL` (default), the function throws an error. The specified directory must exist and be writable; if it does not exist, the function attempts to create it recursively. A unique timestamped subdirectory (e.g., `"Mplus_LPA_YYYY-MM-DD_HH-MM-SS"` or `"Mplus_LCA_YYYY-MM-DD_HH-MM-SS"`) will be created within this path to store all run-specific files and avoid naming conflicts. See in `LCA` and `LPA`.

`files.clean` Logical. If `TRUE` (default), all intermediate files and the temporary working directory created for this run are deleted upon successful completion or error exit (via `on.exit()`). If `FALSE`, all generated files are retained in `files.path` (or the auto-generated temp dir) for inspection or debugging. Note: when `files.path = NULL`, even if `files.clean = FALSE`, the temporary directory may still be cleaned up by the system later — for guaranteed persistence, specify a custom `files.path`.

`control.NNE` List of control parameters for NNE algorithm:

`hidden.layers` Integer vector specifying layer sizes in fully-connected network (default: `c(16,16)`).

`activation.function` Activation function (e.g., `"tanh"`, default: `"tanh"`).

`use.attention` Whether to enable the self-attention mechanism (i.e., transformer encoder) (default: `TRUE`).

`d.model` Dimensionality of transformer encoder embeddings (default: 8).

`nhead` Number of attention heads in transformer (default: 2).

`dim.feedforward` Dimensionality of transformer feedforward network (default: 16).

`eps` Small constant for numerical stability (default: 1e-8).

`lambda` A factor for slight regularization of all parameters (default: 1e-5).

`initial.temperature` Initial temperature for simulated annealing (default: 1000).

`cooling.rate` Cooling rate per iteration in simulated annealing (default: 0.5).

`maxiter.sa` Maximum iterations for simulated annealing (default: 1000).

`threshold.sa` Minimum temperature threshold for annealing (default: 1e-10).

`maxiter` Maximum training epochs (default: 1000).

`maxiter.early` Patience parameter for early stopping (default: 50).

`maxcycle` Maximum cycles for optimization (default: 10).

`lr` Learning rate, controlling the step size of neural network parameter updates (default: 0.025).

`scheduler.patience` Patience for learning rate decay (if the loss function does not improve for more than `patience` consecutive epochs, the learning rate will be reduced) (default: 10).

`scheduler.factor` Learning rate decay factor; the new learning rate equals the original learning rate multiplied by `scheduler.factor` (default: 0.70).

`plot.interval` Interval (in epochs) for plotting training diagnostics (default: 100).

`device` Specifies the hardware device; can be `"CPU"` (default) or `"GPU"`. If the GPU is not available, it automatically falls back to CPU.

Value

An object of class `LCPA`, a named list containing:

`beta` Matrix of size $p \times L$. Coefficients for class membership multinomial logit model. Columns 1 to $L-1$ are free parameters; column L (reference class) is constrained to $\boldsymbol{\beta}_L = \mathbf{0}$.

`beta.se` Standard errors for `beta` (if Hessian is invertible). Same dimensions as `beta`. May contain `NA` if variance-covariance matrix is not positive definite.

`beta.Z.sta` Z-statistics for testing null hypothesis that each beta coefficient equals zero. Computed as `beta / beta.se`. Same structure as `beta`.

`beta.p.value.tail1` One-tailed p-values based on standard normal distribution: $P(Z < -|z|)$. Useful for directional hypotheses. Same structure as `beta`.

`beta.p.value.tail2` Two-tailed p-values: $2 \times P(Z < -|z|)$. Standard test for non-zero effect. Same structure as `beta`.

`P.Z.Xn` Matrix of size $N \times L$ of posterior class probabilities $P(Z_n = l \mid \mathbf{X}_n)$ for each individual n and class l.

`P.Z` Vector of length L containing prior class proportions $P(Z = l)$ estimated at Step 1.

`Z` Vector of length N containing modal class assignments (MAP classifications) $\hat{z}_n$ for each individual.

`npar` Number of free parameters in the model (depends on covariates).

`Log.Lik` Observed-data log-likelihood value at convergence.

`Log.Lik.history` Vector tracking log-likelihood at each iteration.

`AIC` Akaike Information Criterion value.

`BIC` Bayesian Information Criterion value.

`iterations` Integer. Number of optimization iterations in Step 3.

`coveraged` Logical. `TRUE` if optimization terminated before reaching `maxiter` (suggesting convergence). Note: This is a heuristic indicator; formal convergence diagnostics should check Hessian properties.

`params` List. Step 1 model parameters (output from `LCA()` or `LPA()`).

`call` The matched function call.

`arguments` List of all input arguments passed to the function (useful for reproducibility).

Methodology Overview

The three-step procedure follows the same principles as LTA but for a single time point:

Step 1 — Unconditional Latent Class/Profile Model: Fit an unconditional LCA or LPA model (ignoring covariates). Obtain posterior class membership probabilities $P(Z_n = l \mid \mathbf{X}_n)$ for each individual n and class l using Bayes' theorem.

Step 2 — Classification Error Probabilities (equal to `get.CEP`): Compute the $L \times L$ CEP matrix where element (k, l) estimates:

$$\text{CEP}(k, l) = P(\hat{Z}_n = l \mid Z_n = k)$$

using a non-parametric approximation:

$$\widehat{\text{CEP}}(k, l) = \frac{\sum_{n=1}^{N} \mathbb{I}(\hat{z}_n = l) \cdot P(Z_n = k \mid \mathbf{X}_n)}{\sum_{n=1}^{N} P(Z_n = k \mid \mathbf{X}_n)}$$

where $\hat{z}_n$ is the modal class assignment.

Step 3 — Class Membership Model with Measurement Error Correction: Estimate the multinomial logit model for class membership:

$$P(Z_n = l \mid \mathbf{X}_n) = \frac{\exp(\boldsymbol{\beta}_l^\top \mathbf{X}_n)}{\sum_{k=1}^{L} \exp(\boldsymbol{\beta}_k^\top \mathbf{X}_n)}$$

where $\mathbf{X}_n = (1, W_{n1}, \ldots, W_{nM})^\top$ is the covariate vector for individual n (with intercept as first column), and $\boldsymbol{\beta}_l = (\beta_{l0}, \beta_{l1}, \ldots, \beta_{lM})^\top$ contains intercept and regression coefficients. Class L is the reference category ($\boldsymbol{\beta}_L = \mathbf{0}$).

The observed-data likelihood integrates over latent classes:

$$\log \mathcal{L}(\boldsymbol{\beta}) = \sum_{n=1}^{N} \log \left[\sum_{l=1}^{L} \text{CEP}(l, \hat{z}_n) \cdot P(Z_n = l \mid \mathbf{X}_n) \right]$$

Parameters $\boldsymbol{\beta}$ are estimated via maximum likelihood using the BOBYQA algorithm.

Important Implementation Details

- Reference Class: Coefficients for the reference class (`ref.class`) are ALWAYS fixed to zero ($\boldsymbol{\beta}_{ref.class} = \mathbf{0}$) in the multinomial logit model.
- CEP Matrices: When `CEP.error = TRUE`, misclassification probabilities are estimated non-parametrically using Step 1 posterior probabilities. This corrects for classification uncertainty. See in `get.CEP`.

- Covariate Requirements: Covariate matrix MUST include an intercept column (all 1s) as the first column. Dimensions must be $N \times (M+1)$, where M represents the number of covariate and 1 is the Intercept.
- **Optimization & Standard Errors**:
  - Step 3 uses BOBYQA algorithm (`nloptr::nloptr`) for stable optimization with box constraints.
  - For `method.SE = "Obs"`: Standard errors derived from inverse Hessian (`hessian`). If Hessian is singular:
    - Uses Moore-Penrose pseudoinverse (`ginv`)
    - Sets negative variances to `NA`
  - For `method.SE = "Bootstrap"`: Each replicate independently re-estimates Steps 1-3. Failed bootstrap runs yield `NA` in SEs and derived statistics. Progress messages include replicate index and optimization diagnostics.
- **Computational Notes**:
  - Step 1 complexity increases with L and I.
  - Bootstrap is computationally intensive: 100 replicates = 100 full re-estimations of Steps 1-3.
- **Bootstrap Reproducibility**: Always set a seed (e.g., `set.seed(123)`) before calling `LCPA()` when using `method.SE = "Bootstrap"`. Monitor convergence in bootstrap runs via progress messages.

References

Vermunt, J. K. (2010). Latent class modeling with covariates: Two improved three-step approaches. Political Analysis, 18(4), 450–469. https://doi.org/10.1093/pan/mpq025

Liang, Q., la Torre, J. d., & Law, N. (2023). Latent Transition Cognitive Diagnosis Model With Covariates: A Three-Step Approach. Journal of Educational and Behavioral Statistics, 48(6), 690-718. https://doi.org/10.3102/10769986231163320

Examples

```
library(LCPA)

set.seed(123)
N <- 2000  # Sample size
L <- 3    # Number of latent classes
I <- 6    # Number of indicators

# Create covariates (intercept + 2 covariates + 1 interaction)
 Intercept = rep(1, N)
 X1 <- rnorm(N)
 X2 <- rbinom(N, 1, 0.5)
 X1.X2 <- X1 * X2
covariate <- cbind(Intercept, X1, X2, X1.X2)

# Simulate data for LPA
sim_data <- sim.LTA(
  N = N, I = I, L = L, times = 1, type = "LPA",
```

```
  covariates = list(covariate), is.sort=TRUE,
  beta = matrix(c(
   -0.2, 0.0, -0.1,  ## fix reference class to class 2
    0.2, 0.0, -0.3,
    0.8, 0.0, -0.6,
   -0.1, 0.0,  0.3
  ), ncol = L, byrow = TRUE)
)
response <- sim_data$responses[[1]]

## It is strongly recommended to perform the following
## standardization to obtain more stable results when LPA.
## Standardization is not performed here in order to
## compare estimated values with true values.
# response <- normalize(response)

# Fit cross-sectional LPA with covariates
## fix reference class to class 2
# need Mplus
## Not run:
fit <- LCPA(
  response = response,
  L = L, ref.class = 2,
  type = "LPA", is.sort=TRUE,
  covariate = covariate,
  method.SE = "Obs",
  CEP.error = TRUE,
  method = "Mplus",
  control.Mplus = list(files.path = ""),
  vis = TRUE
)
print(fit)

## End(Not run)
```

`logit` *Compute the Logistic (Sigmoid) Function*

Description

This function computes the logistic (also known as sigmoid) transformation of the input. The logistic function maps real-valued numbers to the open interval (0, 1), and is widely used in machine learning, statistical modeling (e.g., logistic regression), and neural networks as an activation function or link function.

Usage

```
logit(x)
```

Arguments

x	A numeric vector, matrix, or array. Accepts any real number, including `Inf` and `-Inf`. Missing values (`NA`) are preserved.

Details

The logistic function is defined as:

$$\text{logit}^{-1}(x) = \frac{1}{1 + e^{-x}}$$

Note: Despite the name "logit", this function actually computes the *inverse logit* (i.e., the logistic function). The true logit function is the inverse: $\log(p/(1-p))$. However, in many applied contexts—especially in software—the term "logit" is sometimes informally used to refer to the sigmoid. For clarity, this implementation follows the conventional definition of the logistic/sigmoid function.

Value

A numeric object of the same dimension as `x`, where each element is the logistic transformation of the corresponding input:

- If `x = 0`, returns `0.5`
- As `x -> Inf`, output approaches `1`
- As `x -> -Inf`, output approaches `0`
- `NA` values remain `NA`

Examples

```
logit(0)        # 0.5
logit(c(-Inf, 0, Inf))  # c(0, 0.5, 1)
logit(c(-2, -1, 0, 1, 2))
```

LPA *Fit Latent Profile Analysis*

Description

This function estimates parameters of a Latent Profile Analysis (LPA) model for continuous observed variables using one of three methods: Expectation-Maximization (EM) algorithm, Neural Network Estimation (NNE), or external Mplus software. It supports flexible covariance structures and initialization strategies.

Usage

```
LPA(
  response,
  L = 2,
  par.ini = "random",
  constraint = "VV",
  method = "EM",
  is.sort = TRUE,
  nrep = 20,
  starts = 100,
  maxiter.wa = 20,
  vis = TRUE,
  control.EM = NULL,
  control.Mplus = NULL,
  control.NNE = NULL
)
```

Arguments

`response` A numeric matrix of dimension $N \times I$, where N is the number of individuals/participants/observations and I is the number of continuous observed indicators/items/variables. Missing values are not allowed. Note that `response` must be standardized using `scale` or `normalize` before input.

`L` Integer specifying the number of latent profiles (default: 2).

`par.ini` Specification for parameter initialization. Options include:

- `"random"`: Random initialization of means and covariances (default).
- `"kmeans"`: Initializes parameters via K-means clustering on observed data (McLachlan & Peel, 2004).
- A `list` containing exactly three elements:

  `means` An $L \times I$ matrix of initial mean vectors for each profile.

  `covs` An $I \times I \times L$ array of initial covariance matrices for each profile.

  `P.Z` A numeric vector of length L specifying initial prior probabilities for profiles.

`constraint` Character string specifying covariance structure constraints:

`"VV"` Varying variances and varying covariances across profiles (heterogeneous full covariance; Default).

`"VE"` Varying variances but equal correlations across profiles.

`"EV"` Equal variances but varying covariances across profiles.

`"EE"` Equal variances and equal covariances across profiles (homogeneous full covariance).

`"E0"` Equal variances across profiles, zero covariances (diagonal with shared variances).

`"V0"` Varying variances across profiles, zero covariances (diagonal with free variances).

list Custom constraints. Each element is a 2-element integer vector specifying variables whose covariance parameters are constrained equal across all classes. The constraint applies to:

- Variances: When both indices are identical (e.g., c(3,3) forces variance of variable 3 to be equal across classes).
- Covariances: When indices differ (e.g., c(1,2) forces covariance between variables 1 and 2 to be equal across classes).

Constraints are symmetric (e.g., c(1,2) automatically constrains c(2,1)). All unconstrained parameters vary freely across classes while maintaining positive definiteness.

method Character string specifying estimation algorithm:

- "EM": Expectation-Maximization algorithm (Default).
- "NNE": Neural Network Estimation with transformer architecture (experimental; uses transformer + simulated annealing, more reliable than both "EM" and "Mplus"). See install_python_dependencies.
- "Mplus": Calls external Mplus software for estimation. Uses Mplus defaults for optimization unless overridden by control.Mplus.

is.sort A logical value. If TRUE (Default), the latent classes will be ordered in descending order according to P.Z. All other parameters will be adjusted accordingly based on the reordered latent classes.

nrep Integer controlling replication behavior:

- If par.ini = "random", number of random initializations.
- If par.ini = "kmeans", number of K-means runs for initialization.
- For method="Mplus", controls number of random starts in Mplus via STARTS option.
- Best solution is selected by log-likelihood/BIC across replications.
- Ignored for user-provided initial parameters.

starts Number of random initializations to explore during warm-up phase (default: 100).

maxiter.wa Maximum number of training iterations allowed per warm-up run. After completion, the top nrep solutions (by log-likelihood) are promoted to final training phase (default: 20).

vis Logical. If TRUE, displays progress information during estimation (default: TRUE).

control.EM List of control parameters for EM algorithm:

maxiter Maximum iterations (default: 2000).

tol Convergence tolerance for log-likelihood difference (default: 1e-4).

control.Mplus List of control parameters for Mplus estimation:

maxiter Maximum iterations for Mplus optimization (default: 2000).

tol Convergence tolerance for log-likelihood difference (default: 1e-4).

files.path Character string specifying the directory path where Mplus will write its intermediate files (e.g., .inp model input, .dat data file, .out output, and saved posterior probabilities). This argument is **required** — if NULL (default), the function throws an error. The specified directory must

exist and be writable; if it does not exist, the function attempts to create it recursively. A unique timestamped subdirectory (e.g., "Mplus_LPA_YYYY-MM-DD_HH-MM-SS") will be created within this path to store all run-specific files and avoid naming conflicts.

files.clean Logical. If TRUE (default), all intermediate files and the temporary working directory created for this run are deleted upon successful completion or error exit (via on.exit()). If FALSE, all generated files are retained in files.path (or the auto-generated temp dir) for inspection or debugging. Note: when files.path = NULL, even if files.clean = FALSE, the temporary directory may still be cleaned up by the system later — for guaranteed persistence, specify a custom files.path.

control.NNE List of control parameters for NNE algorithm:

hidden.layers Integer vector specifying layer sizes in fully-connected network (default: c(16,16)).

activation.function Activation function (e.g., "tanh", default: "tanh").

use.attention Whether to enable the self-attention mechanism (i.e., transformer encoder) (default: TRUE).

d.model Dimensionality of transformer encoder embeddings (default: 8).

nhead Number of attention heads in transformer (default: 2).

dim.feedforward Dimensionality of transformer feedforward network (default: 16).

eps Small constant for numerical stability (default: 1e-8).

lambda A factor for slight regularization of all parameters (default: 1e-5).

initial.temperature Initial temperature for simulated annealing (default: 1000).

cooling.rate Cooling rate per iteration in simulated annealing (default: 0.5).

maxiter.sa Maximum iterations for simulated annealing (default: 1000).

threshold.sa Minimum temperature threshold for annealing (default: 1e-10).

maxiter Maximum training epochs (default: 1000).

maxiter.early Patience parameter for early stopping (default: 50).

maxcycle Maximum cycles for optimization (default: 10).

lr Learning rate, controlling the step size of neural network parameter updates (default: 0.025).

scheduler.patience Patience for learning rate decay (if the loss function does not improve for more than patience consecutive epochs, the learning rate will be reduced) (default: 10).

scheduler.factor Learning rate decay factor; the new learning rate equals the original learning rate multiplied by scheduler.factor (default: 0.70).

plot.interval Interval (in epochs) for plotting training diagnostics (default: 100).

device Specifies the hardware device; can be "CPU" (default) or "GPU". If the GPU is not available, it automatically falls back to CPU.

Value

An object of class "LPA" containing:

`params` List with estimated profile parameters:

`means` $L \times I$ matrix of estimated mean vectors for each profile.

`covs` $I \times I \times L$ array of estimated covariance matrices for each profile.

`P.Z` Vector of length L with profile prior probabilities.

`npar` Number of free parameters in the model (depends on `constraint`).

`Log.Lik` Log-likelihood of the final model.

`AIC` Akaike Information Criterion value.

`BIC` Bayesian Information Criterion value.

`best_BIC` Best BIC value across `nrep` runs (if applicable).

`P.Z.Xn` $N \times L$ matrix of posterior profile probabilities for each observation.

`P.Z` Vector of length L containing the prior probabilities/structural parameters/proportions for each latent class.

`Z` Vector of length N with MAP-classified profile memberships.

`Log.Lik.history` Vector tracking log-likelihood at each EM iteration (only for `method="EM"`).

`Log.Lik.nrep` Vector of log-likelihoods from each replication run.

`model` The optimal model object:

- For `method="NNE"`: Trained neural network model.
- For `method="Mplus"`: Estimated Mplus model.

EM Algorithm

When `method = "EM"`, parameter estimation uses the Expectation-Maximization (EM) algorithm to maximize the observed-data log-likelihood:

$$\log \mathcal{L} = \sum_{n=1}^{N} \log \left[\sum_{l=1}^{L} \pi_l \cdot \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l) \right]$$

The algorithm iterates between two steps until convergence (change in log-likelihood < `tol` or max iterations reached):

E-step: Compute posterior class probabilities (responsibilities) for observation n and class l:

$$\tau_{nl} = \frac{\pi_l \cdot \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}{\sum_{k=1}^{L} \pi_k \cdot \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}$$

where $\mathcal{N}(\cdot)$ is the multivariate normal density, π_l is the prior class probability, and $\boldsymbol{\mu}_l$, $\boldsymbol{\Sigma}_l$ are current parameters. Numerical stability is ensured via the log-sum-exp trick.

M-step: Update parameters using responsibilities τ_{nl}:

- Class probabilities: $\pi_l^{\text{new}} = \frac{1}{N} \sum_{n=1}^{N} \tau_{nl}$
- Class means: $\boldsymbol{\mu}_l^{\text{new}} = \frac{\sum_{n=1}^{N} \tau_{nl} \mathbf{x}_n}{\sum_{n=1}^{N} \tau_{nl}}$
- Class covariances: Updated under constraints:

  `"VV"` $\boldsymbol{\Sigma}_l^{\text{new}} = \frac{\sum_{n=1}^{N} \tau_{nl} (\mathbf{x}_n - \boldsymbol{\mu}_l^{\text{new}})(\mathbf{x}_n - \boldsymbol{\mu}_l^{\text{new}})^\top}{\sum_{n=1}^{N} \tau_{nl}}$

`"EE"` Shared covariance: $\boldsymbol{\Sigma}^{\text{new}} = \frac{\sum_{l=1}^{L}\sum_{n=1}^{N}\tau_{nl}(\mathbf{x}_n-\boldsymbol{\mu}_l^{\text{new}})(\mathbf{x}_n-\boldsymbol{\mu}_l^{\text{new}})^\top}{\sum_{l=1}^{L}\sum_{n=1}^{N}\tau_{nl}}$

`"VE"` **(default) /** `"EV"` Hybrid constraints (e.g., `"VE"`: varying variances, equal correlations). Off-diagonal elements use weighted averages across classes; diagonals retain class-specific values.

Custom constraints User-specified variances/covariances (e.g., `list(c(1,2), c(2, 2))`, meaning the covariates of observed variable 1 and observed variable 2 are equal across latent classes, and the variance of observed variable 2 is equal across classes) are forced equal across classes via weighted averaging.

Covariance matrices are regularized to ensure positive definiteness:

- Eigenvalues < `jitter` (1e-10) are replaced with `jitter`
- Failed Cholesky decompositions trigger diagonal jittering or perturbation of non-constrained elements

Edge Handling:

- Empty classes ($\sum_n \tau_{nl} < 10^{-5}$) are reinitialized by redistributing responsibilities.
- Non-finite likelihoods trigger fallback to previous valid parameters or covariance perturbation.
- Univariate cases ($I = 1$) bypass Cholesky decomposition for direct variance updates.

Neural Network Estimation (NNE)

When `method = "NNE"`, parameters are estimated using a hybrid neural network architecture combining fully-connected layers with transformer-based attention mechanisms. This approach jointly optimizes profile parameters and posterior probabilities through stochastic optimization with simulated annealing. See `install_python_dependencies`. Key components include:

Architecture:

Input Representation: Continuous observed variables $\mathbf{x}_n \in \mathbb{R}^I$ are standardized (mean-centered, unit-variance) internally during training to improve numerical stability. No encoding is needed — raw values are fed directly.

Feature Encoder (Feedforward Network): A multi-layer perceptron with architecture defined by `hidden.layers` and `activation.function` maps the continuous input vector into a latent space of dimension `d.model`. This layer learns non-linear feature combinations predictive of latent profile membership.

Attention Refiner (Transformer Encoder) A transformer encoder with `nhead` attention heads that learns latent class prior probabilities $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_L)$ directly from observed responses.

- Input: `response` matrix ($N \times I$), where N = observations, I = continuous variables.
- Mechanism: Self-attention dynamically weighs variable importance during profile assignment, capturing complex multivariate interactions.
- Output: Class prior vector $\boldsymbol{\pi}$ computed as the mean of posteriors:

$$\pi_l = \frac{1}{N}\sum_{n=1}^{N} attention(\mathbf{X}_n)$$

This ensures probabilistic consistency with the mixture model framework.

Parameter Head (Means & Covariances): Two separate projection heads branch from the transformer output:

- Means Head: Linear projection to $L \times I$ matrix $\boldsymbol{\mu}_l$.
- Covariance Head: Outputs lower-triangular elements of Cholesky factors $\mathbf{L}_l$ for each profile. Diagonal elements are passed through `softplus` to ensure positivity; off-diagonals use `tanh` scaled by 1.2 to bound magnitude and promote stability. The full covariance is reconstructed via $\boldsymbol{\Sigma}_l = \mathbf{L}_l \mathbf{L}_l^\top$.

After reconstruction, covariance constraints (e.g., `"EE"`, `"V0"`, or custom lists) are applied by averaging constrained elements across profiles and re-symmetrizing.

Optimization Strategy:

- **Hybrid Training Protocol**: Alternates between:
  - *Gradient-based phase*: AdamW optimizer minimizes negative log-likelihood with weight decay regularization:
$$-\log \mathcal{L} + \lambda \|\boldsymbol{\theta}\|_2^2$$
where λ is controlled by `lambda` (default: 1e-5). Learning rate decays adaptively when loss plateaus (controlled by `scheduler.patience` and `scheduler.factor`).
  - *Simulated annealing phase*: After gradient-based early stopping (`maxiter.early`), parameters are perturbed with noise scaled by temperature:
$$\theta_{\text{new}} = \theta_{\text{current}} + \mathcal{N}(0, \theta_{\text{current}} \times \frac{T}{T_0})$$
Temperature T decays geometrically ($T \leftarrow T \times$cooling.rate) from `initial.temperature` until `threshold.sa` is reached. This escapes poor local minima.

  Each full cycle (gradient descent + annealing) repeats up to `maxcycle` times.
- **Model Selection**: Across `nrep` random restarts (using Dirichlet-distributed initializations or K-means), the solution with lowest BIC is retained.
- **Diagnostics**: Training loss, annealing points, and global best solution are plotted when `vis=TRUE`.

Constraint Handling:

- Covariance constraints (`constraint`) are enforced *after* activation via:
  - Shared Parameters: Variances/covariances marked for equality are replaced by their average across profiles.
  - Positive Definiteness: Non-positive definite matrices are corrected via eigenvalue clamping, diagonal jittering, or adaptive Cholesky decomposition.
- Custom constraints: e.g., `list(c(1,2), c(3,3))`, force equality of specific covariance elements across profiles, with symmetry ($\sigma_{12} = \sigma_{21}$) automatically enforced.

Mplus

When `method = "Mplus"`, estimation is delegated to external Mplus software. The function automates the entire workflow:

Workflow:

Temporary Directory Setup Creates `inst/Mplus` to store:

- Mplus input syntax (`.inp`)
- Data file in Mplus format (`.dat`)
- Posterior probabilities output (`.dat`)

Files are automatically deleted after estimation unless `control.Mplus$clean.files = FALSE`.

Syntax Generation Constructs Mplus syntax with:

- `CLASSES = c1(L)` specification for L latent classes
- `ANALYSIS` block with optimization controls:
  `TYPE = mixture` Standard mixture modeling setup
  `STARTS = starts nrep` Random `starts` and final stage optimizations
  `STITERATIONS = maxiter.wa` max itertions during `starts`.
  `MITERATIONS = maxiter` Maximum EM iterations
  `CONVERGENCE = tol` Log-likelihood convergence tolerance
- `MODEL` block reflecting the specified `constraint` structure

Execution Calls Mplus via `MplusAutomation::mplusModeler()`, which:

- Writes data to disk in Mplus-compatible format
- Invokes the Mplus executable (requires valid license)
- Captures convergence status and output

Constraint Handling:

- Covariance constraints (`constraint`) are enforced *after* activation via:
  - Shared Parameters: Variances/covariances marked for equality are replaced by their average across profiles.
  - Positive Definiteness: Non-positive definite matrices are corrected via eigenvalue clamping, diagonal jittering, or adaptive Cholesky decomposition.
- Custom constraints: e.g., `list(c(1,2), c(3,3))`, force equality of specific covariance elements across profiles, with symmetry ($\sigma_{12} = \sigma_{21}$) automatically enforced.

References

McLachlan, G. J., & Peel, D. (2004). Finite Mixture Models. Wiley. https://books.google.com.sg/books?id=c2_fAox0DQoC

Examples

```
library(LCPA)

# Simulate bivariate continuous data for 2 profiles
set.seed(123)
data.obj <- sim.LPA(N = 500, I = 3, L = 2, constraint = "VV")
response <- data.obj$response

## It is strongly recommended to perform the following
## standardization to obtain more stable results.
## Standardization is not performed here in order to
## compare estimated values with true values.
# response <- normalize(response)
```

```
# Fit 2-profile model with VV constraint (default)
fit_vv <- LPA(response, L = 2, constraint = "VV")

# Fit 2-profile model with E0 constraint using neural network estimation
# need Python
## Not run:
fit_e0_nne <- LPA(response, L = 2, constraint = "E0", method = "NNE", nrep = 2)

## End(Not run)

# Fit 2-profile model using Mplus
# Requires Mplus to be installed and available in system PATH.
# NOTE: 'files.path' in control.Mplus is REQUIRED — the function will
# throw an error if not provided.
# This example creates a timestamped subdirectory
# (e.g., "Mplus_LPA_YYYY-MM-DD_HH-MM-SS") under './inst'
# to store all temporary Mplus files (.inp, .dat, .out, etc.).
# The 'inst' directory will be created if it does not exist.
# Setting files.clean=FALSE means temporary files will be preserved after execution.
## Not run:
fit_mplus <- LPA(response, L = 2, method = "Mplus", constraint = list(c(1, 2), c(3, 3)),
                 control.Mplus = list(files.path = "inst", files.clean=FALSE))

## End(Not run)
```

`LRT.test` *Likelihood Ratio Test*

Description

Conducts a likelihood ratio test to compare the fit of two models. The test evaluates whether a model with more parameters provides a significantly better fit than a model with fewer parameters.

Usage

```
LRT.test(object1, object2)
```

Arguments

`object1`	Fitted model object with fewer parameters (i.e., fewer `npar`, small model).
`object2`	Fitted model object with more parameters (i.e., more `npar`, large model).

Details

Note that since the small model may be nested within the large model, the result of `LRT.test` may not be accurate and is provided for reference only. More reliable conclusions should be based on a combination of fit indices (i.e., `get.fit.index`), classification accuracy measures (i.e., `get.entropy`, `get.AvePP`), and a bootstrapped likelihood-ratio test (i.e., BLRT, `LRT.test.Bootstrap`,

which is very time-consuming). Above all and the most important criterion, is that the better model is the one that aligns with theoretical expectations and offers clear interpretability.

The `LRT.test` test statistic is defined as:

- The models must be *nested* (i.e., the model with fewer parameters is a constrained version of the more one).
- Both models must be fit on the identical dataset with the same response variables.
- The test statistic asymptotically follows a chi-square distribution.

$$LRT = -2 \times (\text{LogLik}_1 - \text{LogLik}_2)$$

where:

- LogLik_1: Log-likelihood of the smaller model (fewer parameters).
- LogLik_2: Log-likelihood of the larger model (more parameters).

Under the null hypothesis (`H_0`: small model is true), LRT asymptotically follows a chi-square distribution with df degrees of freedom.

Value

An object of class `"htest"` containing:

- `statistic`: VLMR adjusted test statistic
- `parameter`: Degrees of freedom ($df = npar_2 - npar_1$)
- `p.value`: P-value from $\chi^2_d f$ distribution
- `method`: Name of the test
- `data.name`: Model comparison description

LRT.test.Bootstrap *Bootstrap Likelihood Ratio Test for Latent Class/Profile Models*

Description

Conducts a bootstrap likelihood ratio test (BLRT) to compare two nested latent class analysis (LCA) or latent profile analysis (LPA) models. The test evaluates whether a model with more classes (the alternative model) provides significantly better fit than a model with fewer classes (the null model). Implements both fixed-replicate and sequential stopping procedures for computational efficiency.

Usage

```
LRT.test.Bootstrap(
  object1,
  object2,
  n.Bootstrap = 100,
  vis = TRUE,
  use.sequential = TRUE
)
```

Arguments

object1	Fitted model object representing the null model (e.g., L-1 classes). Must be of class "LCA" or "LPA".
object2	Fitted model object representing the alternative model (e.g., L classes). Must be of the same class as object1.
n.Bootstrap	Maximum number of bootstrap replicates (default = 100). If use.sequential = FALSE, exactly this many replicates are performed. If use.sequential = TRUE (default), this is an upper bound; the algorithm may stop early. McLachlan & Peel (2000) suggest 100 replicates as typically sufficient for stable p-values, especially near the conventional $\alpha = 0.05$ threshold. This default also aligns with the implementation in Nylund et al. (2007).
vis	Logical. If TRUE (default), displays real-time progress during bootstrapping.
use.sequential	Logical. If TRUE (default), applies the sequential stopping rule from Nylund et al. (2007) to terminate early when the decision is statistically clear. If FALSE, performs exactly n.Bootstrap replicates (traditional fixed bootstrap).

Details

Core Workflow (Parametric Bootstrap):

- Parameter Extraction: Parameters from the null model (object1) are treated as the population truth.
- Data Simulation: Generates datasets using sim.LCA or sim.LPA, preserving the original sample size and indicator structure.
- Model Re-fitting: For each simulated dataset, both the null and alternative models are re-estimated. Initial parameters are set to random values (par.ini = "random") to avoid convergence to local maxima; the EM algorithm includes fallback strategies upon non-convergence.
- LRT Distribution: Computes bootstrap LRT statistics: $LRT_{\text{boot}} = -2 \times (\text{logLik}_{\text{null,boot}} - \text{logLik}_{\text{alt,boot}})$.
- P-value Calculation: The p-value is the proportion of bootstrap LRTs that are greater than or equal to the observed LRT statistic.

Sequential Stopping Rule (Nylund et al., 2007): When use.sequential = TRUE, the algorithm checks three stopping criteria after each replication:

- Upper Stopping Points: Stop if the current estimated p-value ($\hat{p}$) is greater than or equal to a pre-specified threshold at replication n_i, indicating insufficient evidence against the null model.
- Lower Stopping Points: Stop if $\hat{p} \leq p_i$ at replication n_i, indicating strong evidence in favor of the alternative model.
- Conditional Lower Stopping Points: Stop if $\hat{p} = 0$ *and* the observed LRT exceeds the mean of the current bootstrap LRTs by more than s standard deviations. This leverages the approximate normality of the LRT distribution under the null (Nylund et al., 2007).

The exact stopping thresholds follow Appendix A of Nylund et al. (2007):

- Upper: (n, 2/n) for n = 2–3; (n, 3/n) for n = 4–9; (n, 4/n) for n = 10–17; (n, 5/n) for n = 18–26; (n, 6/n) for n = 27–99; (100, 0).
- Lower: (49, 0), (78, 1/78).
- Conditional: (5, 0, 20), (10, 0, 10), (20, 0, 5), where the third value denotes the required z-score threshold.

This rule ensures >95% agreement with the decision that would be made using an infinite number of replicates when the true p-value is below 0.10. Near $p = 0.05$, the algorithm continues until the maximum number of replicates to avoid premature conclusions.

Critical Interpretation Notes:

- The BLRT should **not** be used in isolation. Always integrate results with:
  - Information criteria: e.g., BIC, aBIC, CAIC via `get.fit.index`.
  - Classification quality metrics: e.g., entropy (`get.entropy`), average posterior probability (`get.AvePP`).
  - Substantive interpretability and theoretical coherence of the extracted classes.
- The BLRT is especially valuable in LCA/LPA contexts where the standard asymptotic LRT is invalid due to boundary parameter issues (e.g., class probabilities approaching 0 or 1; McLachlan & Peel, 2000).
- Early termination under sequential stopping (e.g., at $n \ll 100$) yields reliable p-values when the evidence is strong ($p \ll 0.05$ or $p \gg 0.05$). However, if stopping occurs near `n.Bootstrap` with $p \approx 0.05$, the result should be considered inconclusive.
- In small samples ($N < 300$) or with poorly separated classes, the BLRT may exhibit inflated Type I error rates; corroborate findings with BIC and other indices.

Value

An object of class `"htest"` containing:

- `statistic`: Observed likelihood ratio test statistic, $-2 \times (\text{logLik}_{\text{null}} - \text{logLik}_{\text{alt}})$.
- `parameter`: Degrees of freedom (set to `NA`; the bootstrap p-value does not rely on asymptotic chi-square distribution).
- `p.value`: Bootstrap p-value, computed as the proportion of bootstrap LRT statistics greater than or equal to the observed LRT statistic.
- `method`: Character string indicating the procedure used: either "Bootstrap LRT with Sequential Stopping" or "Bootstrap LRT (Fixed Replicates)".
- `data.name`: Descriptive string summarizing the model comparison, including the actual number of bootstrap replicates performed and, if applicable, the reason for early stopping under the sequential rule.
- `LRT.Bootstrap`: Numeric vector of length R (where R is the actual number of bootstrap replicates performed) containing the bootstrap LRT statistics: $LRT_{\text{boot}} = -2 \times (\text{logLik}_{\text{null,boot}} - \text{logLik}_{\text{alt,boot}})$. These are generated under the null hypothesis (i.e., data simulated from the smaller model). This vector can be used for diagnostic purposes, such as inspecting the empirical null distribution or computing alternative p-value estimates.

References

McLachlan, G. J., & Peel, D. (2000). Finite Mixture Models. John Wiley & Sons.

Nylund, K. L., Asparouhov, T., & Muthén, B. O. (2007). Deciding on the Number of Classes in Latent Class Analysis and Growth Mixture Modeling: A Monte Carlo Simulation Study. Structural Equation Modeling: A Multidisciplinary Journal, 14(4), 535–569. https://doi.org/10.1080/10705510701575396

`LRT.test.VLMR` *Lo-Mendell-Rubin likelihood ratio test*

Description

Implements the ad-hoc adjusted likelihood ratio test (LRT) described in Formula 15 of Lo, Mendell, & Rubin (2001), also known as VLMR LRT (Vuong-Lo-Mendell-Rubin Adjusted LRT). This method is typically used to compare models with `L-1` and `L` classes. If the difference in the number of classes exceeds 1, conclusions should be interpreted with extreme caution.

Usage

```
LRT.test.VLMR(object1, object2)
```

Arguments

`object1`	Fitted model object with fewer parameters (i.e., fewer `npar`, small model).
`object2`	Fitted model object with more parameters (i.e., more `npar`, large model).

Details

Note that since the small model may be nested within the large model, the result of `LRT.test.VLMR` may not be accurate and is provided for reference only. More reliable conclusions should be based on a combination of fit indices (i.e., `get.fit.index`), classification accuracy measures (i.e., `get.entropy`, `get.AvePP`), and a bootstrapped likelihood-ratio test (i.e., BLRT, `LRT.test.Bootstrap`, which is very time-consuming). Above all and the most important criterion, is that the better model is the one that aligns with theoretical expectations and offers clear interpretability.

The `LRT.test.VLMR` statistic is defined as:

$$VLMR = \frac{LRT}{c} \quad \text{where} \quad c = 1 + \frac{1}{df \cdot \log(N)}$$

where:

- LRT is the standard likelihood ratio statistic. see `LRT.test`
- $df = npar_2 - npar_1$ is the difference in number of free parameters between models.
- N is the sample size.

Under the null hypothesis (`H_0`: small model is true), VLMR asymptotically follows a chi-square distribution with df degrees of freedom.

Value

An object of class "htest" containing:

- statistic: VLMR adjusted test statistic
- parameter: Degrees of freedom ($df = npar_2 - npar_1$)
- p.value: P-value from $\chi^2_d f$ distribution
- method: Name of the test
- data.name: Model comparison description

References

Lo, Y., Mendell, N. R., & Rubin, D. B. (2001). Testing the number of components in a normal mixture. Biometrika, 88(3), 767-778. https://doi.org/10.1093/biomet/88.3.767

Nylund-Gibson, K., & Choi, A. Y. (2018). Ten frequently asked questions about latent class analysis. Translational Issues in Psychological Science, 4(4), 440-461. https://doi.org/10.1037/tps0000176

LTA *Latent Transition Analysis (LTA)*

Description

Implements the three-step estimation method (Vermunt, 2010; Liang et al., 2023) for Latent Transition Analysis (LTA), treating latent class memberships at each time point as observed variables with measurement error. Classification uncertainty from Step 1 (latent class/profile analysis) is explicitly incorporated into the transition model estimation in Step 3, ensuring asymptotically unbiased estimates of transition probabilities and covariate effects. This avoids the bias introduced by "hard" modal-class assignment.

Usage

```
LTA(
  responses,
  L = 2,
  ref.class = L,
  type = "LCA",
  covariates = NULL,
  CEP.timeCross = FALSE,
  CEP.error = TRUE,
  covariates.timeCross = FALSE,
  par.ini = "random",
  params = NULL,
  is.sort = TRUE,
  constraint = "VV",
  method = "EM",
  tol = 1e-04,
```

```
  lower = -10,
  upper = 10,
  method.SE = "Bootstrap",
  n.Bootstrap = 100,
  maxiter = 5000,
  nrep = 20,
  starts = 100,
  maxiter.wa = 20,
  vis = TRUE,
  control.EM = NULL,
  control.Mplus = NULL,
  control.NNE = NULL
)
```

Arguments

responses	A list of response matrices or data frames. Each matrix corresponds to one time point. Rows of each matrix represent individuals/participants/observations (N), columns of each matrix represent observed items/variables (I). For type = "LCA": items must be binary or categorical (coded as integers starting from 0). For type = "LPA": items must be continuous (numeric), and each response matrix must be standardized using scale or normalize prior to input.
L	Integer scalar. Number of latent classes/profiles at each time point. Must satisfy $L \geq 2$.
ref.class	Integer $L \geq ref.class \geq 1$. Specifies which latent class to use as the reference category. Default is L (last class). Coefficients for the reference class are fixed to zero. When is.sort=TRUE, classes are first ordered by decreasing P.Z (class 1 has highest probability), then ref.class refers to the position in this sorted order.
type	Character string. Specifies the type of latent variable model for Step 1: • "LCA" — Latent Class Analysis for categorical items. • "LPA" — Latent Profile Analysis for continuous items. See LCA and LPA for details.
covariates	Optional. A list of matrices/data frames (length = number of time points). Each matrix contains covariates for modeling transitions or initial status. Must include an intercept column (all 1s) as the first column. If NULL (default), only intercept terms are used (i.e., no covariates). For time t, dimension is $N \times p_t$. Covariates can vary across time.
CEP.timeCross	Logical. If TRUE, assumes measurement invariance and uses the same Classification Error Probability (get.CEP) matrix across all time points. Requires that item parameters are invariant over time (not checked internally). Default is FALSE.
CEP.error	Logical. If TRUE (recommended), incorporates classification uncertainty via estimated CEP matrices from Step 1. If FALSE, uses identity CEP matrices (equivalent to naive modal assignment; introduces bias and not recommended).

`covariates.timeCross`
: Logical. If `TRUE`, forces the use of identical γ parameters across all time points (i.e., a time-invariant probability transition matrix). In this case, users should ensure that the covariate matrices at different time points have the same dimensions (values may differ) to match the fixed form of the γ_{lkt} coefficients. Default is `FALSE`, allowing for potentially different probability transition matrices across time points.

`par.ini`
: Specification for parameter initialization. Options include:

- `"random"`: Completely random initialization (default).
- `"kmeans"`: Initializes parameters via K-means clustering on observed data (McLachlan & Peel, 2004).
- A `list` for LCA containing:
  - `par` An $L \times I \times K_{\max}$ array of initial conditional probabilities for each latent class, item, and response category (where $K_{\max}$ is the maximum number of categories across items).
  - `P.Z` A numeric vector of length L specifying initial prior probabilities for latent classes.
- A `list` for LPA containing:
  - `means` An $L \times I$ matrix of initial mean vectors for each profile.
  - `covs` An $I \times I \times L$ array of initial covariance matrices for each profile.
  - `P.Z` A numeric vector of length L specifying initial prior probabilities for profiles.

`params`
: Optional `list` of pre-estimated Step 1 parameters. If `NULL` (default), Step 1 models are estimated internally. If provided, no LCA or LPA parameter estimation will be performed; instead, the parameters provided in `params` will be used as fixed values. Additionally, `params` must contain:

- A `list` for LCA containing:
  - `par` An $L \times I \times K_{\max}$ array of initial conditional probabilities for each latent class, item, and response category (where $K_{\max}$ is the maximum number of categories across items).
  - `P.Z` A numeric vector of length L specifying initial prior probabilities for latent classes.
- A `list` for LPA containing:
  - `means` An $L \times I$ matrix of initial mean vectors for each profile.
  - `covs` An $I \times I \times L$ array of initial covariance matrices for each profile.
  - `P.Z` A numeric vector of length L specifying initial prior probabilities for profiles.

`is.sort`
: A logical value. If `TRUE` (Default), the latent classes will be ordered in descending order according to `P.Z`. All other parameters will be adjusted accordingly based on the reordered latent classes.

`constraint`
: Character (LPA only). Specifies structure of within-class covariance matrices:

- `"VV"` — Class-varying variances and covariances (unconstrained; default).
- `"EE"` — Equal variances and covariances across all classes (homoscedastic).

`method` Character. Estimation algorithm for Step 1 models:

- `"EM"` — Expectation-Maximization (default; robust and widely used).
- `"Mplus"` — Interfaces with Mplus software (requires external installation).
- `"NNE"` — Neural Network Estimator (experimental; uses transformer + simulated annealing, more reliable than both `"EM"` and `"Mplus"`).

`tol` Convergence tolerance for log-likelihood difference (default: 1e-4).

`lower` The upper bound for the estimation of regression coefficients, default is -10

`upper` The lower bound for the estimation of regression coefficients, default is 10

`method.SE` Character. Method for estimating standard errors of parameter estimates:

- `"Obs"` — Approximates the observed information matrix via numerical differentiation (Richardson's method). Standard errors are obtained from the inverse Hessian. May fail or be unreliable in small samples or with complex likelihood surfaces.
- `"Bootstrap"` — Uses nonparametric bootstrap resampling to estimate empirical sampling variability. More robust to model misspecification and small-sample bias. Computationally intensive but recommended when asymptotic assumptions are questionable.

Default is `"Bootstrap"`.

`n.Bootstrap` Integer. Number of bootstrap replicates used when `method.SE = "Bootstrap"`. Default is 100. McLachlan & Peel (2004) suggest that 50–100 replicates often provide adequate accuracy for practical purposes, though more (e.g., 500–1000) may be preferred for publication-quality inference. Each replicate involves re-estimating the full three-step LTA model on a resampled dataset.

`maxiter` Maximum number of iterations for optimizing the regression coefficients. Default: 5000.

`nrep` Integer controlling replication behavior:

- If `par.ini = "random"`, number of random initializations.
- If `par.ini = "kmeans"`, number of K-means runs for initialization.
- For `method="Mplus"`, controls number of random starts in Mplus via `STARTS` option.
- Best solution is selected by log-likelihood/BIC across replications.
- Ignored for user-provided initial parameters.

`starts` Number of random initializations to explore during warm-up phase (default: 100).

`maxiter.wa` Maximum number of training iterations allowed per warm-up run. After completion, the top `nrep` solutions (by log-likelihood) are promoted to final training phase (default: 20).

`vis` Logical. If `TRUE`, displays progress information during estimation (default: `TRUE`).

`control.EM` List of control parameters for EM algorithm:

`maxiter` Maximum iterations (default: 2000).

`tol` Convergence tolerance for log-likelihood difference (default: 1e-4).

`control.Mplus` List of control parameters for Mplus estimation:

`maxiter` Maximum iterations for Mplus optimization (default: 2000).

`tol` Convergence tolerance for log-likelihood difference (default: 1e-4).

`files.path` Character string specifying the directory path where Mplus will write its intermediate files (e.g., `.inp` model input, `.dat` data file, `.out` output, and saved posterior probabilities). This argument is **required** — if `NULL` (default), the function throws an error. The specified directory must exist and be writable; if it does not exist, the function attempts to create it recursively. A unique timestamped subdirectory (e.g., `"Mplus_LPA_YYYY-MM-DD_HH-MM-SS"` or `"Mplus_LCA_YYYY-MM-DD_HH-MM-SS"`) will be created within this path to store all run-specific files and avoid naming conflicts. See in `LCA` and `LPA`.

`files.clean` Logical. If `TRUE` (default), all intermediate files and the temporary working directory created for this run are deleted upon successful completion or error exit (via `on.exit()`). If `FALSE`, all generated files are retained in `files.path` (or the auto-generated temp dir) for inspection or debugging. Note: when `files.path = NULL`, even if `files.clean = FALSE`, the temporary directory may still be cleaned up by the system later — for guaranteed persistence, specify a custom `files.path`.

`control.NNE` List of control parameters for NNE algorithm:

`hidden.layers` Integer vector specifying layer sizes in fully-connected network (default: `c(16,16)`).

`activation.function` Activation function (e.g., `"tanh"`, default: `"tanh"`).

`use.attention` Whether to enable the self-attention mechanism (i.e., transformer encoder) (default: `TRUE`).

`d.model` Dimensionality of transformer encoder embeddings (default: 8).

`nhead` Number of attention heads in transformer (default: 2).

`dim.feedforward` Dimensionality of transformer feedforward network (default: 16).

`eps` Small constant for numerical stability (default: 1e-8).

`lambda` A factor for slight regularization of all parameters (default: 1e-5).

`initial.temperature` Initial temperature for simulated annealing (default: 1000).

`cooling.rate` Cooling rate per iteration in simulated annealing (default: 0.5).

`maxiter.sa` Maximum iterations for simulated annealing (default: 1000).

`threshold.sa` Minimum temperature threshold for annealing (default: 1e-10).

`maxiter` Maximum training epochs (default: 1000).

`maxiter.early` Patience parameter for early stopping (default: 50).

`maxcycle` Maximum cycles for optimization (default: 10).

`lr` Learning rate, controlling the step size of neural network parameter updates (default: 0.025).

`scheduler.patience` Patience for learning rate decay (if the loss function does not improve for more than `patience` consecutive epochs, the learning rate will be reduced) (default: 10).

`scheduler.factor` Learning rate decay factor; the new learning rate equals the original learning rate multiplied by `scheduler.factor` (default: 0.70).

`plot.interval` Interval (in epochs) for plotting training diagnostics (default: 100).

`device` Specifies the hardware device; can be `"CPU"` (default) or `"GPU"`. If the GPU is not available, it automatically falls back to CPU.

Value

An object of class `LTA`, a named list containing:

`beta` Matrix of size $p_1 \times L$. Coefficients for initial class membership multinomial logit model. Columns 1 to $L-1$ are free parameters; column L (reference class) is constrained to $\boldsymbol{\beta}_L = \mathbf{0}$.

`gamma` List of length $T-1$. Each element `gamma[[t]]` (for transition from time t to $t+1$) is a nested list: `gamma[[t]][[from_class]][[to_class]]` returns coefficient vector of length p_{t+1}. The last class (L) is reference $\rightarrow$ coefficients fixed to $\boldsymbol{\gamma}_{kl,t+1} = \mathbf{0}$ for all k.

`beta.se` Standard errors for `beta` (if Hessian is invertible). Same dimensions as `beta`. May contain `NA` if variance-covariance matrix is not positive definite.

`gamma.se` Standard errors for `gamma`, same nested structure. May contain `NA`s.

`beta.Z.sta` Z-statistics for testing null hypothesis that each beta coefficient equals zero. Computed as `beta / beta.se`. Same structure as `beta`.

`gamma.Z.sta` Z-statistics for gamma coefficients. Same nested structure as `gamma`. Used for testing significance of transition effects.

`beta.p.value.tail1` One-tailed p-values based on standard normal distribution: $P(Z < -|z|)$. Useful for directional hypotheses. Same structure as `beta`.

`gamma.p.value.tail1` One-tailed p-values for gamma coefficients. Same nested structure as `gamma`.

`beta.p.value.tail2` Two-tailed p-values: $2 \times P(Z < -|z|)$. Standard test for non-zero effect. Same structure as `beta`.

`gamma.p.value.tail2` Two-tailed p-values for gamma coefficients. Same nested structure as `gamma`.

`P.Z.Xns` List of length T. Each element is an $N \times L$ matrix of posterior class probabilities $P(Z_{nt} = l \mid \mathbf{X}_{nt})$ for each individual n at time t.

`P.Zs` List of length T. Each element is a vector of length L containing prior class proportions $P(Z_t = l)$ estimated at Step 1 for time t.

`Zs` List of length T. Each element is a vector of length N containing modal class assignments (MAP classifications) $\hat{z}_{nt}$ for each individual at time t.

`npar` Number of free parameters in the model (depends on `covariates`).

`Log.Lik` Observed-data log-likelihood value at convergence.

`Log.Lik.history` Vector tracking log-likelihood at each iteration.

`AIC` Akaike Information Criterion value.

`BIC` Bayesian Information Criterion value.

`iterations` Integer. Number of optimization iterations in Step 3.

`coveraged` Logical. `TRUE` if optimization terminated before reaching `maxiter` (suggesting convergence). Note: This is a heuristic indicator; formal convergence diagnostics should check Hessian properties.

`params` List. Step 1 model parameters (output from `LCA()` or `LPA()`).

`call` The matched function call.

`arguments` List of all input arguments passed to the function (useful for reproducibility).

Methodology Overview

The three-step LTA proceeds as follows:

Step 1 — Unconditional Latent Class/Profile Model: At each time point t, fit an unconditional LCA or LPA model (ignoring transitions and covariates). Obtain posterior class membership probabilities $P(Z_{nt} = l \mid \mathbf{X}_{nt})$ for each individual n and class l using Bayes' theorem.

Step 2 — Classification Error Probabilities (equal to `get.CEP`): Compute the $L \times L$ CEP matrix for each time point t, where element (k, l) estimates:

$$\text{CEP}_t(k, l) = P(\hat{Z}_{nt} = l \mid Z_{nt} = k)$$

using a non-parametric approximation based on posterior weights:

$$\widehat{\text{CEP}}_t(k, l) = \frac{\sum_{n=1}^{N} \mathbb{I}(\hat{z}_{nt} = l) \cdot P(Z_{nt} = k \mid \mathbf{X}_{nt})}{\sum_{n=1}^{N} P(Z_{nt} = k \mid \mathbf{X}_{nt})}$$

where $\hat{z}_{nt}$ is the modal (most likely) class assignment for individual n at time t.

Step 3 — Transition Model with Measurement Error Correction: Estimate the multinomial logit models for:

- Initial class membership (time 1): $P(Z_{n1} = l \mid \mathbf{X}_{n1}) = \frac{\exp(\boldsymbol{\beta}_l^\top \mathbf{X}_{n1})}{\sum_{k=1}^{L} \exp(\boldsymbol{\beta}_k^\top \mathbf{X}_{n1})}$
- Transitions (time $t > 1$): $P(Z_{nt} = l \mid Z_{n,t-1} = k, \mathbf{X}_{nt}) = \frac{\exp(\boldsymbol{\gamma}_{klt}^\top \mathbf{X}_{nt})}{\sum_{j=1}^{L} \exp(\boldsymbol{\gamma}_{kjt}^\top \mathbf{X}_{nt})}$

where $\mathbf{X}_{n1} = (X_{n10}, X_{n11}, \ldots, X_{n1M})^\top$ is the covariate vector for observation/participant n at time 1, with $X_{n10} = 1$ (intercept term) and X_{n1m} $(m = 1, \ldots, M)$ representing the value of the m-th covariate. The coefficient vector $\boldsymbol{\beta}_l = (\beta_{l0}, \beta_{l1}, \ldots, \beta_{lM})^\top$ corresponds element-wise to $\mathbf{X}_{n1}$, where β_{l0} is the intercept and β_{lm} $(m \geq 1)$ are regression coefficients for covariates. Class L is the reference class ($\boldsymbol{\beta}_L = \mathbf{0}$). $\mathbf{X}_{nt} = (X_{nt0}, X_{nt1}, \ldots, X_{ntM})^\top$ is the covariate vector at time t, with $X_{nt0} = 1$ (intercept) and X_{ntm} $(m = 1, \ldots, M)$ as the m-th covariate value. The coefficient vector $\boldsymbol{\gamma}_{lkt} = (\gamma_{lkt0}, \gamma_{lkt1}, \ldots, \gamma_{lktM})^\top$ corresponds element-wise to $\mathbf{X}_{nt}$, where γ_{lkt0} is the intercept and γ_{lktm} $(m \geq 1)$ are regression coefficients. Class L is the reference class ($\boldsymbol{\gamma}_{lLt} = \mathbf{0}$ for all l).

The full observed-data likelihood integrates over all possible latent class paths $\mathbf{z}_n = (z_{n1}, \ldots, z_{nT})$:

$$\begin{aligned}\log \mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \log \Bigg[\sum_{\mathbf{z}_n \in \{1,\ldots,L\}^T} \Big(\prod_{t=1}^{T} \text{CEP}_t(z_{nt}, \hat{z}_{nt}) \Big) \cdot \\ P(Z_{n1} = z_{n1} \mid \mathbf{X}_{n1}) \cdot \\ \prod_{t=2}^{T} P(Z_{nt} = z_{nt} \mid Z_{n,t-1} = z_{n,t-1}, \mathbf{X}_{nt}) \Bigg]\end{aligned}$$

Parameters $\boldsymbol{\theta} = \{\boldsymbol{\beta}, \boldsymbol{\gamma}\}$ are estimated via maximum likelihood using the BOBYQA algorithm (box-constrained derivative-free optimization). Reference class L satisfies $\boldsymbol{\beta}_L = \mathbf{0}$ and $\boldsymbol{\gamma}_{klt} = \mathbf{0}$ for all k when $l = L$.

Bootstrap Standard Error Estimation

When `method.SE = "Bootstrap"`, standard errors are estimated using a nonparametric bootstrap procedure:

1. Draw B (`=n.Bootstrap`) independent samples of size N with replacement from the original data.
2. For each bootstrap sample $b = 1, \ldots, B$, re-estimate the full three-step LTA model (Steps 1–3), yielding parameter vector $\hat{\boldsymbol{\theta}}^{(b)}$.
3. Compute the bootstrap standard error for each parameter as the sample standard deviation across replicates:

$$\widehat{\mathrm{SE}}_{\mathrm{boot}}(\hat{\theta}_j) = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}\left(\hat{\theta}_j^{(b)} - \bar{\theta}_j\right)^2},$$

where $\bar{\theta}_j = \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}_j^{(b)}$.

This approach does not rely on large-sample normality or correct specification of the information matrix, making it particularly suitable for complex models like LTA where analytic derivatives are difficult or unstable. However, it increases computational cost linearly with B.

Important Implementation Details

- Reference Class: The last latent class (L) is always treated as the reference category. All corresponding coefficients in `beta` and `gamma` are fixed to zero ($\boldsymbol{\beta}_L = \mathbf{0}$, $\boldsymbol{\gamma}_{klt} = \mathbf{0}$ for $l = L$).
- CEP Matrices: When `CEP.error = TRUE`, misclassification probabilities are estimated nonparametrically using Step 1 posterior probabilities. This corrects for classification uncertainty. Setting `CEP.timeCross = TRUE` assumes these error structures are identical across time (measurement invariance). See in `get.CEP`.
- Covariate Handling: Covariates for initial status (time 1) and transitions (time $t \geq 2$) can differ. For transitions to time t, the covariate matrix must have dimensions $N \times (M_t + 1)$, i.e., an intercept column of all 1 plus M_t columns of covariates in time t.
- Optimization: Step 3 uses L-BFGS-B via `nloptr` to ensure numerical stability. Standard errors are derived from the inverse Hessian (via `hessian`). If singular, Moore-Penrose pseudoinverse (`ginv`) is used, and negative variances are set to `NA`.
- Computational Complexity: Likelihood evaluation requires enumerating L^T possible latent paths.
- Bootstrap Computation: Each bootstrap iteration re-fits Steps 1–3 independently, including re-estimation of `P`, `Z`, `Xns`, `CEP` matrices and transition parameters. To ensure reproducibility, set a seed before calling `LTA()` when using `method.SE = "Bootstrap"`. Progress messages during bootstrapping include current replicate index and optimization diagnostics. Users should monitor convergence in each bootstrap run; failed runs will result in `NA` entries in SEs and derived statistics.

References

Liang, Q., la Torre, J. d., & Law, N. (2023). Latent Transition Cognitive Diagnosis Model With Covariates: A Three-Step Approach. Journal of Educational and Behavioral Statistics, 48(6), 690-718. https://doi.org/10.3102/10769986231163320

McLachlan, G. J., & Peel, D. (2004). Finite Mixture Models. Wiley. https://books.google.com.sg/books?id=c2_fAox0DQoC

Vermunt, J. K. (2010). Latent class modeling with covariates: Two improved three-step approaches. Political Analysis, 18(4), 450–469. https://doi.org/10.1093/pan/mpq025

Examples

```
library(LCPA)

set.seed(123)
N <- 2000 ## sample size
L <- 3   ## number of latent class
I <- 6   ## number of variables/items

## Covariates at time point T1
covariates.inter <- rep(1, N)             # Intercept term is always 1 for each individual
covariates.X11 <- rnorm(N)                    # Covariate X1 is a continuous variable
# Combine into covariates at T1
covariates.T1 <- cbind(Intercept=covariates.inter, X1=covariates.X11)
## Covariates at time point T2
covariates.inter <- rep(1, N)             # Intercept term is always 1 for each individual
covariates.X21 <- rnorm(N)                    # Covariate X1 is a continuous variable
# Combine into covariates at T1
covariates.T2 <- cbind(Intercept=covariates.inter, X1=covariates.X21)

# Combine into final covariates list
covariates <- list(t1=covariates.T1, t2=covariates.T2)

## Simulate beta coefficients
# last column is zero because the last category is used as reference
## fix reference class to class 2
beta <- matrix(c( 0.8, 0.0,  0.1,
                 -0.3, 0.0, -0.5), ncol=L, byrow=TRUE)

## Simulate gamma coefficients
gamma <- list(
  lapply(1:L, function(l) {
    lapply(1:L, function(k) if(k < L)
           runif(2, -2.0, 2.0) else c(0, 0)) # Last class as reference
  })
)

## Simulate the data
sim_custom <- sim.LTA(
  N=N, I=I, L=L, times=2, type="LCA", IQ=0.9,
  covariates=covariates,
  beta=beta,
  gamma=gamma
)
summary(sim_custom)
responses <- sim_custom$responses
```

```
covariates <- sim_custom$covariates

## fix reference class to class 2
LTA.obj <- LTA(responses, L=L, ref.class=2, type="LCA",
               covariates=covariates,
               method.SE="Bootstrap", n.Bootstrap=10,
               CEP.timeCross=FALSE, CEP.error=TRUE, covariates.timeCross=FALSE,
               par.ini = "random", method="EM", vis = TRUE)

print(LTA.obj)
```

`normalize` *Column-wise Z-Score Standardization*

Description

Standardizes each column of a numeric matrix or data frame to have mean zero and standard deviation one. This transformation is essential for many multivariate techniques that assume standardized inputs. The function preserves all dimension names and returns a pure numeric matrix with attributes storing original column means and standard deviations.

Usage

```
normalize(response)
```

Arguments

`response`	A numeric matrix or data frame of dimension $N \times I$, where: • N = number of observations (rows) • I = number of indicators/items/variables (columns) Non-numeric columns will be coerced to numeric with a warning. Missing values are not allowed and will cause the function to fail. Constant columns (zero variance) will produce `NaN` values.

Value

A standardized numeric matrix of dimension $N \times I$ with attributes:

- `scaled:center`: Vector of original column means (μ_i)
- `scaled:scale`: Vector of original column standard deviations (σ_i)
- Row names: Preserved from original input's row names or row indices
- Column names: Preserved from original input's column names
- Values: Z-scores calculated as $z_{ni} = \frac{x_{ni} - \mu_i}{\sigma_i}$

where:

- x_{ni} = original value for observation n and variable i
- μ_i = sample mean of variable i: $\mu_i = \frac{1}{N}\sum_{n=1}^{N} x_{ni}$
- σ_i = sample standard deviation of variable i: $\sigma_i = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}(x_{ni} - \mu_i)^2}$

The denominator $N - 1$ provides an unbiased estimator of population variance.

Mathematical Details

For each column i in the input matrix X, the standardization is performed as:

$$Z_{\cdot i} = \frac{X_{\cdot i} - \bar{X}_{\cdot i}}{S_{X_{\cdot i}}}$$

where:

- $X_{\cdot i}$ is the i-th column vector of X
- $\bar{X}_{\cdot i}$ is the sample mean of column i
- $S_{X_{\cdot i}}$ is the sample standard deviation of column i

The resulting matrix Z has the properties:

$$\frac{1}{N}\sum_{n=1}^{N} z_{ni} = 0 \quad \text{and} \quad \sqrt{\frac{1}{N-1}\sum_{n=1}^{N} z_{ni}^2} = 1$$

for all $i = 1, \ldots, I$.

Examples

```
# Basic usage with matrix
set.seed(123)
mat <- matrix(rnorm(30, mean = 5:7, sd = 1:3), ncol = 3,
              dimnames = list(paste0("Obs", 1:10), paste0("Var", 1:3)))
norm_mat <- normalize(mat)

# Verify attributes
attr(norm_mat, "scaled:center")  # Original column means
attr(norm_mat, "scaled:scale")   # Original column standard deviations

# Verify properties
apply(norm_mat, 2, mean)  # Should be near zero
apply(norm_mat, 2, sd)    # Should be exactly 1

# With data frame input
df <- as.data.frame(mat)
norm_df <- normalize(df)
all.equal(norm_mat, norm_df, check.attributes = FALSE)  # Should be identical

# Handling constant columns (produces NaN)
const_mat <- cbind(mat, Constant = rep(4.2, 10))
normalize(const_mat)
```

plot *S3 Methods: plot*

Description

Generates user-friendly, publication-ready visualizations for objects generated by the LCPA package. This generic function dispatches to class-specific methods that produce diagnostic and interpretive plots tailored to each model type. Designed for interactive exploration, model interpretation, and presentation-quality output.

Usage

```
## S3 method for class 'LCA'
plot(x, ncol = 2, ...)

## S3 method for class 'LPA'
plot(x, ncol = 2, ...)
```

Arguments

x	An object of one of the following classes: • Model objects: LCA, LPA.
ncol	Number of columns in the multi-panel layout (default: 2). Controls arrangement of latent class/profile panels.
...	Additional arguments passed to methods.

Details

Each method produces a structured, visually intuitive plot optimized for its object type. See specific method documentation (plot.LCA, plot.LPA) for detailed parameter options.

Value

Invisibly returns the final ggplot or patchwork object. No data is modified.

Methods (by class)

- plot(LCA): Plot method for LCA objects
- plot(LPA): Plot method for LPA objects (with covariance heatmap)

`plotResponse` *Visualize Response Distributions with Density Plots*

Description

Creates a publication-quality density plot showing the distribution of responses across multiple indicators/items/variables. Automatically handles variable ordering, color scaling, and legend layout based on the number of variables.

Usage

```
plotResponse(response)
```

Arguments

`response`	A matrix or data frame containing response data where: • Rows represent respondents, samples or observations • Columns represent variables, indicators or questions (must have numeric suffixes, e.g., "indicator1", "Q2") • Cell values contain numeric responses Non-numeric columns (except row identifiers) will cause errors.

Value

A `ggplot` object containing:

- Density curves for each variable colored by indicator
- Adaptive color schemes based on number of variables
- Optimized legend layout for large numbers of indicators
- Publication-ready theme with grid lines and clean styling

The plot can be further customized using standard ggplot2 syntax.

Theming Details

The plot uses a minimal theme with:

- Light grey grid lines for readability
- Black axis lines and ticks (0.8pt thickness)
- White background with no panel border
- Optimized font sizes (13pt axis titles, 11pt tick labels)
- Legend positioned on right with adaptive sizing

Examples

```
# Simulate response data for 5 indicators
set.seed(42)
resp_data <- data.frame(
  indicator1 = rnorm(100, mean = 3, sd = 1),
  indicator2 = rnorm(100, mean = 2, sd = 0.8),
  indicator3 = rnorm(100, mean = 4, sd = 1.2),
  indicator4 = rnorm(100, mean = 3.5, sd = 0.9),
  indicator5 = rnorm(100, mean = 2.5, sd = 1.1)
)

library(LCPA)
# Generate and display plot
p <- plotResponse(resp_data)
print(p)

# For data with many indicators (18 indicators example)
many_indicators <- as.data.frame(replicate(18, rnorm(50, mean = runif(1, 1, 5), sd = 1)))
names(many_indicators) <- paste0("Q", 1:18)
p_large <- plotResponse(many_indicators)
print(p_large)
```

---

print *S3 Methods: print*

Description

Provides user-friendly, formatted console output for objects generated by the LCPA package. This generic function dispatches to class-specific methods that display concise summaries of model results, simulated datasets, fit indices, model comparisons, and standard errors. Designed for interactive use and quick diagnostic inspection.

Usage

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

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

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

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

## S3 method for class 'LTA'
```

```
print(x, ...)

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

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

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

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

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

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

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

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

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

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

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

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

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

## S3 method for class 'SE'
print(x, digits = 4, I.max = 5, L.max = 3, ...)

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

Arguments

x	An object of one of the following classes: • Model objects: LCA, LPA, LCPA, LTA • Simulation objects: sim.LCA, sim.LPA, sim.LTA • Fit/comparison objects: get.fit.index, compare.model • Standard error objects: get.SE • Summary objects: summary.LCA, summary.LPA, summary.LCPA, summary.LTA, summary.sim.LCA, summary.sim.LPA, summary.sim.LTA, summary.fit.index, summary.compare.model, summary.SE
...	Additional arguments passed to methods (currently ignored in most cases).
digits	Number of decimal places for numeric output (default: varies by method, often 4). Used by: print.SE, print.summary.fit.index, print.summary.sim.LCA/LPA/LTA, print.summary.SE.
I.max	Maximum number of variables/items to display before truncation (default: varies, e.g., 5). Used by: print.SE, print.summary.sim.LCA/LPA/LTA.
L.max	Maximum number of latent classes/profiles to display before truncation (default: varies, e.g., 3). Used by: print.SE, print.summary.sim.LTA.

Details

Each method produces a structured, human-readable summary optimized for its object type:

Model Objects (LCA/LPA/LCPA/LTA) Invokes summary() internally and prints comprehensive output including:

- Model call and configuration (method, constraints, reference class)
- Data characteristics (N, I, time points, distribution)
- Fit statistics (LogLik, AIC, BIC, entropy, npar)
- Class/profile prior probabilities and frequencies
- Item-response probabilities (LCA) or profile means (LPA)
- For LCPA/LTA: regression coefficients with significance markers and 95% CIs
- Convergence diagnostics (iterations, tolerance, hardware)
- Replication details (if nrep > 1)

Simulation Objects (sim.LCA/sim.LPA/sim.LTA) Displays simulation design and true parameter structure:

- Configuration (N, I, L, times, constraint, distribution)
- True class/profile proportions and observed frequencies
- For sim.LCA: item category structure and conditional probabilities
- For sim.LPA: covariance constraint description and mean ranges
- For sim.LTA: transition mode (fixed/covariate), initial/transition coefficients

Output is truncated for high-dimensional structures using I.max and L.max.

Fit Index Objects (fit.index) Presents a clean table of model fit criteria:

- Header with dimensions (N, I, L, npar)

- Formatted table: AIC, BIC, SABIC, CAIC, AWE, -2LL, SIC
- Interpretation note: "Lower values preferred for ICs"
- Values rounded to `digits` decimal places

Model Comparison Objects (`compare.model`) Compares two nested models with statistical tests:

- Comparative fit table (npar, LogLik, AIC, BIC, entropy)
- Classification quality (AvePP per class, overall entropy)
- Bayes Factor with interpretive guidance
- Likelihood ratio tests (standard, VLMR, Bootstrap) with p-values and significance codes
- Clear section headers and visual separators

Standard Error Objects (`SE`) Displays uncertainty estimates for model parameters:

- Class probability SEs (always fully shown)
- Profile means SEs (`LPA`) or item-response SEs (`LCA`), truncated by `L.max`/`I.max`
- Covariance SE summary (non-zero count only; full access via `extract()`)
- Diagnostics: Bootstrap completion % or Hessian condition number with stability warnings

Summary Objects All `summary.*` methods are called internally by their corresponding `print.*` methods. They pre-compute and structure output for consistent formatting. Direct calls are also supported.

Value

Invisibly returns the input object `x`. No data is modified.

Methods (by class)

- `print(LCA)`: Print method for `LCA` objects
- `print(summary.LCA)`: Print method for `summary.LCA` objects
- `print(LPA)`: Print method for `LPA` objects
- `print(summary.LPA)`: Print method for `summary.LPA` objects
- `print(LTA)`: Print method for `LTA` objects
- `print(summary.LTA)`: Print method for `summary.LTA` objects
- `print(LCPA)`: Print method for `LCPA` objects
- `print(summary.LCPA)`: Print method for `summary.LCPA` objects
- `print(sim.LCA)`: Print method for `sim.LCA` objects
- `print(summary.sim.LCA)`: Print method for `summary.sim.LCA` objects
- `print(sim.LPA)`: Print method for `sim.LPA` objects
- `print(summary.sim.LPA)`: Print method for `summary.sim.LPA` objects
- `print(sim.LTA)`: Print method for `sim.LTA` objects
- `print(summary.sim.LTA)`: Print method for `summary.sim.LTA` objects
- `print(fit.index)`: Print method for `fit.index` objects
- `print(summary.fit.index)`: Print method for `summary.fit.index` objects

- `print(compare.model)`: Print method for `compare.model` objects
- `print(summary.compare.model)`: Print method for `summary.compare.model` objects
- `print(SE)`: Print method for `SE` objects
- `print(summary.SE)`: Print method for summary.SE objects

Output Conventions

- Numeric values are typically rounded to 4 decimal places unless overridden by `digits`.
- Large matrices (e.g., item parameters, transition coefficients) are truncated with clear messages.
- Significance markers: *** (<0.001), ** (<0.01), * (<0.05), . (<0.1).
- 95% confidence intervals computed as: Estimate ± 1.96 × Std_Error.
- Reference classes (for multinomial models) are explicitly stated.
- Warnings appear for unstable SEs (high condition number) or incomplete Bootstrap runs.

`rdirichlet` *Generate Random Samples from the Dirichlet Distribution*

Description

`rdirichlet` generates `n` random observations from a Dirichlet distribution with a specified concentration parameter vector `alpha`.

Usage

```
rdirichlet(n, alpha)
```

Arguments

`n`	Integer. The number of random vectors to generate.
`alpha`	Numeric vector. The concentration parameters (must be positive). The length of this vector determines the number of dimensions K.

Details

The Dirichlet distribution is a family of continuous multivariate probability distributions parameterized by a vector α of positive reals. It is the multivariate generalization of the beta distribution and is commonly used as a conjugate prior to the multinomial distribution in Bayesian statistics.

Probability Density Function:

For a vector $x = (x_1, \ldots, x_K)$ on the unit simplex (where $\sum x_i = 1$ and $x_i \geq 0$), the density is given by:

$$f(x_1, \ldots, x_K; \alpha_1, \ldots, \alpha_K) = \frac{1}{B(\alpha)} \prod_{i=1}^{K} x_i^{\alpha_i - 1}$$

where the normalizing constant $B(\alpha)$ is the multivariate beta function:

$$B(\alpha) = \frac{\prod_{i=1}^{K} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{K} \alpha_i)}$$

Simulation Method:

The function utilizes the property that if $Y_1, \ldots, Y_K$ are independent Gamma random variables such that $Y_i \sim Gamma(shape = \alpha_i, rate = 1)$, then:

$$X_i = \frac{Y_i}{\sum_{j=1}^{K} Y_j}$$

The resulting vector $(X_1, \ldots, X_K)$ follows a Dirichlet distribution with parameters α.

Value

A matrix with `n` rows and `length(alpha)` columns. Each row sums to 1, representing a single sample from the Dirichlet distribution.

Examples

```
# Generate 5 samples from a 3-dimensional Dirichlet distribution
set.seed(123)
alpha_params <- c(1, 2, 5)
result <- rdirichlet(n = 5, alpha = alpha_params)
print(result)

# Check that rows sum to 1
rowSums(result)
```

`sim.correlation` *Generate a Random Correlation Matrix via C-Vine Partial Correlations*

Description

This function generates a random $I \times I$ correlation matrix using the C-vine partial correlation parameterization described in Joe & Kurowicka (2026). The method constructs the matrix recursively using partial correlations organized in a C-vine structure, with distributional properties controlled by LKJ concentration and skewness parameters.

Usage

```
sim.correlation(
  I,
  eta = 1,
  skew = 0,
  positive = FALSE,
  permute = TRUE,
  maxiter = 10
)
```

Arguments

I	Dimension of the correlation matrix (must be $I \geq 1$).
eta	LKJ concentration parameter ($\eta > 0$). When $\eta = 1$ and skew $= 0$, the distribution is uniform over correlation matrices. Larger η values concentrate mass near the identity matrix. Critical for positive definiteness: Requires $\eta > (I-2)/2$ to theoretically guarantee positive definiteness (Theorem 1, Joe & Kurowicka 2026). Default is 1.
skew	Skewness parameter ($-1 <$ skew < 1). Controls asymmetry in the partial correlation distribution: • skew > 0: Biased toward positive partial correlations • skew < 0: Biased toward negative partial correlations • skew $= 0$: Symmetric distribution (default)
positive	Logical. If TRUE, restricts partial correlations to $(0, 1)$ and enforces positive definiteness. Default is FALSE.
permute	Logical. If TRUE, applies a random permutation to rows/columns to ensure exchangeability (invariance to variable ordering). Default is TRUE.
maxiter	Integer. Maximum number of generation attempts before numerical adjustment when positive = TRUE. Default is 10.

Details

The algorithm follows four key steps:

1. Partial correlation sampling: For tree level $k = 1, \ldots, I-1$ and node $j = k+1, \ldots, I$, partial correlations $\rho_{k,j|1:(k-1)}$ are sampled as:

$$\alpha_k = \eta + \frac{I-k-1}{2}, \quad a_k = \alpha_k(1+\text{skew}), \quad b_k = \alpha_k(1-\text{skew})$$

   - If positive = FALSE:
   $$\rho_{k,j} \sim 2 \cdot \text{Beta}(a_k, b_k) - 1$$
   - If positive = TRUE:
   $$\rho_{k,j} \sim \text{Beta}(a_k, b_k)$$

2. Recursive matrix construction (C-vine): The correlation matrix $\mathbf{R}$ is built without matrix inversion using backward recursion:

- Tree 1 (raw correlations): $R_{1j} = \rho_{1,j}$ for $j = 2, \ldots, I$
- Trees $l \geq 2$: For pairs (l, j) where $l = 2, \ldots, I-1$ and $j = l+1, \ldots, I$:

$$c \leftarrow \rho_{l,j|1:(l-1)} \text{for } k = l-1 \text{ down to } 1: \quad c \leftarrow c \cdot \sqrt{(1-\rho_{k,l}^2)(1-\rho_{k,j}^2)} + \rho_{k,l} \cdot \rho_{k,j} R_{lj} \leftarrow c$$

This implements the dynamic programming approach from Joe & Kurowicka (2026, Section 2.1).

3. Positive definiteness enforcement (when `positive = TRUE`):
   - Attempt up to `maxiter` generations
   - On failure, project to nearest positive definite correlation matrix using `nearPD` with `corr = TRUE`
   - Final matrix has minimum eigenvalue > 1e-8
4. Exchangeability (optional): If `permute = TRUE`, rows/columns are randomly permuted before returning the matrix.

Value

An $I \times I$ positive definite correlation matrix with unit diagonal.

Note

When `positive = TRUE`, the function guarantees positive definiteness either through direct generation (with retries) or numerical projection. The theoretical guarantee $\eta > (I-2)/2$ is recommended for high dimensions.

References

Joe, H., & Kurowicka, D. (2026). Random correlation matrices generated via partial correlation C-vines. Journal of Multivariate Analysis, 211, 105519. https://doi.org/10.1016/j.jmva.2025.105519

Examples

```
# Default 3x3 correlation matrix
sim.correlation(3)

# 5x5 matrix concentrated near identity (eta=3)
sim.correlation(5, eta = 3)

# Skewed toward positive correlations (no permutation)
sim.correlation(4, skew = 0.7, permute = FALSE)

# Positive partial correlations (enforced positive definiteness)
R <- sim.correlation(6, positive = TRUE)
min(eigen(R, symmetric = TRUE, only.values = TRUE)$values)  # > 1e-8

# High-dimensional case (I=20) with theoretical guarantee
R <- sim.correlation(20, eta = 10)  # eta=10 > (20-2)/2=9
min(eigen(R, symmetric = TRUE, only.values = TRUE)$values)
```

`sim.LCA` *Simulate Data for Latent Class Analysis*

Description

Generates synthetic multivariate categorical data from a latent class model with `L` latent classes. Each observed variable follows a multinomial distribution within classes, with flexible control over class separation via the `IQ` parameter and class size distributions.

Usage

```
sim.LCA(
  N = 1000,
  I = 10,
  L = 3,
  poly.value = 5,
  IQ = "random",
  distribution = "random",
  params = NULL,
  is.sort = TRUE
)
```

Arguments

`N`	Integer; total number of observations to simulate. Must be $> L$. Default: `1000`.
`I`	Integer; number of categorical observed variables. Must be ≥ 1. Default: `10`.
`L`	Integer; number of latent classes. Must be ≥ 2 when `IQ` is numeric. Default: `3`.
`poly.value`	Integer or integer vector; number of categories (levels) for each observed variable. If scalar, all variables share the same number of categories. If vector, must have length `I`. Minimum valid value is `2` when `IQ` is numeric. Default: `5`.
`IQ`	Character or numeric; controls category probability distributions: `"random"` (default) Dirichlet-distributed probabilities ($\alpha = 3$). `Numeric` in $(0.5, 1)$. Forces high discriminative power (see details in section below).
`distribution`	Character; distribution of class sizes. Options: `"random"` (default) or `"uniform"`.
`params`	List with fixed parameters for simulation: `par` $L \times I \times K_{\max}$ array of conditional response probabilities per latent class. `P.Z` Vector of length L with latent class prior probabilities. `Z` Vector of length N containing the latent classes of observations. A fixed observation classes `Z` is applied directly to simulate data only when `P.Z` is `NULL` and `Z` is a `N` length vector.
`is.sort`	A logical value. If `TRUE` (Default), the latent classes will be ordered in descending order according to `P.Z`. All other parameters will be adjusted accordingly based on the reordered latent classes.

Details

Probability Generation:

- Dirichlet Sampling (`IQ="random"`): For each variable-class combination, probabilities are drawn from Dirichlet($\alpha_1 = 3, \ldots, \alpha_k = 3$) where k = poly.value[i].
- High-Discrimination Mode (`IQ=numeric`): For each variable `i`:
  1. Generate special probabilities `par.special` of length `L` containing: IQ, $1 - IQ$, and $L - 2$ values uniformly sampled from $[1 - IQ, IQ]$.
  2. For each class `l`, assign `par.special[l]` to one category, distribute remaining probability $1 -$ par.special[l] uniformly (via Dirichlet) across other categories.
  3. Shuffle category assignments to avoid position bias.

Data Generation:

- Class assignments `Z` are generated first according to `distribution`.
- For each observation `p` and variable `i`:
  1. Retrieve cumulative probabilities for class `Z[p]`
  2. Draw uniform random number $u \sim$ Uniform$(0, 1)$
  3. Assign category `k` where P(category $\leq k - 1) < u \leq P$(category $\leq k$)
- Entire dataset is regenerated if any category of any variable has zero observations.

Critical Constraints:

- When `IQ` is numeric: $0.5 < IQ < 1$ and `min(poly.value) >= 2`
- `N` must be sufficiently large to observe all categories, especially when `IQ` is high or `poly.value` is large. Simulation may fail for small `N`.
- For `distribution="uniform"`, empty classes possible when $N < L$.

Value

A list containing:

response Integer matrix ($N \times I$) of simulated observations. Rows are observations (named `"O1"`, `"O2"`, ...), columns are variables named `"I1"`, `"I2"`, ... Values range from `0` to `poly.value[i]-1`.

par Array ($L \times I \times K$) of true class-specific category probabilities, where $K = \max(poly.value)$ (i.e., the maximum number of categories across variables). Dimensions: classes x variables x categories. Note: For variables with `poly.value[i] < K`, unused category dimensions contain `NA`. Dimension names: `"L1"`, `"L2"`, ... (classes); `"I1"`, `"I2"`, ... (variables); `"poly0"`, `"poly1"`, ... (categories).

Z Integer vector (length N) of true class assignments (1 to L). Named with observation IDs (e.g., `"O1"`).

P.Z Numeric vector (length L) of true class proportions. Named with class labels (e.g., `"L1"`).

poly.value Integer vector (length I) specifying number of categories per variable.

P.Z.Xn Binary matrix ($N \times L$) of true class membership indicators (one-hot encoded). Row `i`, column `l` = 1 if observation `i` belongs to class `l`, else 0. Row/column names match `Z` and class labels.

arguments A list containing all input arguments.

Indicator Quality (IQ) Parameter

Controls the discriminative power of observed variables:

`IQ = "random"` (Default) Category probabilities for each variable-class combination are drawn from a symmetric Dirichlet distribution ($\alpha = 3$), resulting in moderate class separation.

`IQ = numeric` ($0.5 < IQ < 1$) Forces high discriminative power for each variable:

1. For each variable, two categories per class are assigned extreme probabilities: one category gets probability IQ, another gets $1 - IQ$.
2. Remaining categories share the residual probability $1 - IQ - (1 - IQ) = 0$. *Note: This requires* `poly.value >= 2` *for all variables.*
3. Category assignments are randomized within classes to avoid structural patterns.

Higher `IQ` values (closer to 1) yield stronger class separation but increase simulation failure risk.

Class Size Distribution

`"random"` (Default) Class proportions drawn from Dirichlet distribution ($\alpha = 3$ for all classes), ensuring no empty classes. Sizes are rounded to integers with adjustment for exact `N`.

`"uniform"` Equal probability of class membership ($1/L$ per class), sampled with replacement. May produce empty classes if `N` is small relative to `L`.

Response Validation

The simulation enforces a critical constraint: *every category of every observed variable must appear at least once in the dataset*. If initial generation violates this (e.g., a rare category is missing), parameters and responses are regenerated until satisfied. This ensures compatibility with standard LCA estimation.

Examples

```
# Example 1: Default settings (moderate separation, random class sizes)
sim_data <- sim.LCA(N = 30, I = 5, L = 3)

# Example 2: High-discrimination indicators (IQ=0.85), uniform class sizes
sim_high_disc <- sim.LCA(
  N = 30,
  I = 4,
  L = 2,
  poly.value = c(3,4,3,5),  # Variable category counts
  IQ = 0.85,
  distribution = "uniform"
)

# Example 3: Binary indicators (poly.value=2) with high separation
sim_binary <- sim.LCA(N = 300, I = 10, L = 2, poly.value = 2, IQ = 0.9)
```

`sim.LPA` *Simulate Data for Latent Profile Analysis*

Description

Generates synthetic multivariate continuous data from a latent profile model with `L` latent classes. Supports flexible covariance structure constraints (including custom equality constraints) and class size distributions. All covariance matrices are ensured to be positive definite.

Usage

```
sim.LPA(
  N = 1000,
  I = 5,
  L = 2,
  constraint = "VV",
  distribution = "random",
  mean.range = c(-2, 2),
  covs.range = c(0.01, 4),
  params = NULL,
  is.sort = TRUE
)
```

Arguments

`N`	Integer; total number of observations to simulate. Must be $\geq$ `L` (Default = 1000).
`I`	Integer; number of continuous observed variables. Must be ≥ 1 (Default = 5).
`L`	Integer; number of latent profiles (classes). Must be ≥ 1 (Default = 2).
`constraint`	Character string or list specifying covariance constraints. See detailed description below. Default is `"VV"` (fully heterogeneous covariances).
`distribution`	Character; distribution of class sizes. Options: `"random"` (default) or `"uniform"`.
`mean.range`	Numeric vector of length 2; range for sampling class-specific means. Each variable's means are sampled uniformly from `mean.range[1]` to `mean.range[2]`. Default: `c(-4, 4)`.
`covs.range`	Numeric vector of length 2; range for sampling variance parameters (diagonal elements). Must satisfy `covs.range[1] > 0` and `covs.range[2] > covs.range[1]`. Off-diagonal covariances are derived from correlations scaled by these variances. Default: `c(0.01, 4)`.
`params`	List with fixed parameters for simulation: `par` $L \times I \times K_{\max}$ array of conditional response probabilities per latent class. `P.Z` Vector of length L with latent class prior probabilities. `Z` Vector of length N containing the latent classes of observations. A fixed observation classes `Z` is applied directly to simulate data only when `P.Z` is `NULL` and `Z` is a `N` length vector.

`is.sort` A logical value. If `TRUE` (Default), the latent classes will be ordered in descending order according to `P.Z`. All other parameters will be adjusted accordingly based on the reordered latent classes.

Details

Mean Generation: For each variable, $3L$ candidate means are sampled uniformly from `mean.range`. L distinct means are selected without replacement to ensure separation between classes.

Covariance Generation:

- **Positive Definiteness:** All covariance matrices are adjusted using `Matrix::nearPD` and eigenvalue thresholds ($> 10^{-8}$) to guarantee validity. Failed attempts trigger explicit errors.
- **Univariate Case (`I=1`):** Constraints `"UE"` and `"UV"` are enforced automatically. Predefined constraints like `"E0"` map to `"UE"`.
- **VE Constraint:** Requires special handling—base off-diagonal elements are fixed, and diagonals are sampled above a minimum threshold to maintain positive definiteness. May fail if `covs.range` is too narrow.

Class Assignment:

- `"random"`: Uses Dirichlet distribution ($\alpha = 3$) to avoid extremely small classes. Sizes are rounded and adjusted to sum exactly to `N`.
- `"uniform"`: Simple random sampling with equal probability. May produce empty classes if `N` is small.

Data Generation: Observations are simulated using `mvtnorm::rmvnorm` per class. Final data and class labels are shuffled to remove ordering artifacts.

Value

A list containing:

response Numeric matrix ($N \times I$) of simulated observations. Rows are observations, columns are variables named `"V1"`, `"V2"`, ..., or `"UV"` for univariate data.

means Numeric matrix ($L \times I$) of true class-specific means. Row names: `"Class1"`, `"Class2"`, ...; column names match `response`.

covs Array ($I \times I \times L$) of true class-specific covariance matrices. Dimensions: variables x variables x classes. Constrained parameters have identical values across class slices. Dimension names match `response` and class labels.

P.Z.Xn Numeric matrix ($N \times L$) of true class membership probabilities (one-hot encoded). Row `i`, column `l` = 1 if observation `i` belongs to class `l`, else 0. Row names: `"O1"`, `"O2"`, ...; column names: `"Class1"`, `"Class2"`, ...

P.Z Numeric vector (length L) of true class proportions. Named with class labels (e.g., `"Class1"`).

Z Integer vector (length N) of true class assignments (1 to L). Named with observation IDs (e.g., `"O1"`).

constraint Original constraint specification (character string or list) passed to the function.

Covariance Constraints

The `constraint` parameter controls equality constraints on covariance parameters across classes:

Predefined Constraints (Character Strings): `"UE"` **(Univariate only)** Equal variance across all classes.

`"UV"` **(Univariate only)** Varying variances across classes.

`"E0"` Equal variances across classes, zero covariances (diagonal matrix with shared variances).

`"V0"` Varying variances across classes, zero covariances (diagonal matrix with free variances).

`"EE"` Equal full covariance matrix across all classes (homogeneous).

`"EV"` Equal variances but varying covariances (equal diagonal, free off-diagonal).

`"VE"` Varying variances but equal correlations (free diagonal, equal correlation structure).

`"VV"` Varying full covariance matrices across classes (heterogeneous; default).

Custom Constraints (List of integer vectors): Each element specifies a pair of variables whose covariance parameters are constrained equal across classes:

`c(i,i)` Constrains variance of variable `i` to be equal across all classes.

`c(i,j)` Constrains covariance between variables `i` and `j` to be equal across all classes (symmetric: automatically includes `c(j,i)`).

Unconstrained parameters vary freely. The algorithm ensures positive definiteness by:

1. Generating a base positive definite matrix `S0`.
2. Applying constraints via a logical mask.
3. Adjusting unconstrained variances to maintain positive definiteness.

Critical requirements for custom constraints:

At least one variance must be unconstrained if any off-diagonal covariance is unconstrained.

All indices must be between 1 and `I`.

For univariate data (`I=1`), only `list(c(1,1))` is valid.

Class Size Distribution

`"random"` (Default) Class proportions drawn from Dirichlet distribution ($\alpha = 3$ for all classes), ensuring no empty classes. Sizes are rounded to integers with adjustment for exact `N`.

`"uniform"` Equal probability of class membership ($1/L$ per class), sampled with replacement.

Examples

```
# Example 1: Bivariate data, 3 classes, heterogeneous covariances (default)
sim_data <- sim.LPA(N = 500, I = 2, L = 3, constraint = "VV")

# Example 2: Univariate data, equal variances
# 'E0' automatically maps to 'UE' for I=2
sim_uni <- sim.LPA(N = 200, I = 2, L = 2, constraint = "E0")

# Example 3: Custom constraints
# - Equal covariance between V1 and V2 across classes
```

```
# - Equal variance for V3 across classes
sim_custom <- sim.LPA(
  N = 300,
  I = 3,
  L = 4,
  constraint = list(c(1, 2), c(3, 3))
)

# Example 4: VE constraint (varying variances, equal correlations)
sim_ve <- sim.LPA(N = 400, I = 3, L = 3, constraint = "VE")

# Example 5: Uniform class sizes
sim_uniform <- sim.LPA(N = 300, I = 4, L = 5, distribution = "uniform")
```

`sim.LTA` *Simulate Data for Latent Transition Analysis (LTA)*

Description

Simulates longitudinal latent class/profile data where initial class membership and transition probabilities may be influenced by time-varying covariates. Supports both Latent Class Analysis (LCA) for categorical outcomes and Latent Profile Analysis (LPA) for continuous outcomes. Measurement invariance is assumed by default (identical indicator parameters across time).

Usage

```
sim.LTA(
  N = 500,
  I = 5,
  L = 3,
  distribution = "random",
  times = 2,
  type = "LCA",
  rate = NULL,
  constraint = "VV",
  mean.range = c(-2, 2),
  covs.range = c(0.01, 4),
  poly.value = 5,
  IQ = "random",
  params = NULL,
  is.sort = TRUE,
  covariates = NULL,
  beta = NULL,
  gamma = NULL
)
```

Arguments

N	Integer; sample size.
I	Integer; number of observed indicators/items/indicators per time point.
L	Integer; number of latent classes/profiles.
distribution	Character; distribution of initial class probabilities when not using covariates or params. Options: "uniform" (equal probabilities) or "random" (Dirichlet-distributed, default).
times	Integer; number of time points (must be ≥ 1).
type	Character; type of latent model. "LCA" for categorical indicators (default), "LPA" for continuous indicators.
rate	List of matrices or NULL; transition probability matrices for non-covariate mode. Each matrix is $L \times L$ with rows summing to 1. If NULL (default), matrices are generated with 0.7 diagonal probability and uniform off-diagonals. **Ignored when** times=1.
constraint	Character; covariance structure for LPA (type="LPA" only). Options: "VV" (unstructured, default), "VE" (diagonal variance), "EE" (equal variance).
mean.range	Numeric vector; range for randomly generated class means in LPA (default: c(-2, 2)).
covs.range	Numeric vector; range for covariance matrix diagonals in LPA (default: c(0.01, 4)).
poly.value	Integer; number of categories for polytomous LCA indicators (default: 5).
IQ	Character; method for generating indicator discrimination in LCA. "random" (default) or fixed values.
params	List or NULL; pre-specified parameters for reproducibility (see Details).
is.sort	A logical value. If TRUE (Default), the latent classes will be ordered in descending order according to P.Z. All other parameters will be adjusted accordingly based on the reordered latent classes.
covariates	List of matrices or NULL; covariate matrices for each time point. Each matrix must have dimensions $N \times p_t$ and include an intercept column (first column must be all 1s). If NULL, covariate mode is disabled. See Details for automatic coefficient generation.
beta	Matrix or NULL; initial state regression coefficients of dimension $p_1 \times L$. Columns correspond to classes 1 to L (last class L is reference and must be zero). If NULL and covariates are used, coefficients are randomly generated from Uniform$(-1, 1)$.
gamma	List or NULL; transition regression coefficients. Must be a list of length times-1. Each element t is a list of length L (previous state). Each sub-list contains L vectors (next state), where the last vector (reference class) is always $\mathbf{0}$. **Ignored when** times=1. If NULL and covariates are used with times>=2, coefficients are randomly generated from Uniform$(-1, 1)$ for non-reference classes.

Details

Covariate Requirements:

- Covariate matrices must include an intercept (first column = 1). If omitted, the function adds an intercept and issues a warning.
- When `covariates` is provided but `beta` or `gamma` is `NULL`, coefficients are randomly generated from Uniform$(-1, 1)$ (non-reference classes only).
- The reference class (L) always has zero coefficients ($\boldsymbol{\beta}_L = \mathbf{0}$, $\boldsymbol{\gamma}_{l,L} = \mathbf{0}$).

Parameter Compatibility:

- Use `params` to fix indicator parameters (LCA) or class means/covariances (LPA) across simulations.
- In non-covariate mode, `rate` must be a list of $(times - 1)$ valid transition matrices (ignored when `times=1`).
- In covariate mode with `times>=2`, all three (`covariates`, `beta`, `gamma`) must be consistent in dimensions.

Value

A list of class `"sim.LTA"` containing:

`responses` List of length `times`; observed data matrices ($N \times I$).

`Zs` List of length `times`; true latent class memberships ($N \times 1$ vectors).

`P.Zs` List of length `times`; marginal class probabilities at each time.

`par` Indicator parameters for LCA (if `type="LCA"`).

`means` Class means for LPA (if `type="LPA"`).

`covs` Class covariance matrices for LPA (if `type="LPA"`).

`rate` True transition matrices (non-covariate mode only; `NULL` when `times=1`).

`covariates` List of covariate matrices used (covariate mode only).

`beta` True initial state coefficients (covariate mode only).

`gamma` True transition coefficients (covariate mode only; `NULL` when `times=1`).

`call` Function call.

`arguments` Input arguments.

Model Specification

Initial Class Probabilities (with covariates): For observation/participant n at time 1, the probability of belonging to latent class l is:

$$P(Z_{n1} = l \mid \mathbf{X}_{n1}) = \frac{\exp(\boldsymbol{\beta}_l^\top \mathbf{X}_{n1})}{\sum_{k=1}^{L} \exp(\boldsymbol{\beta}_k^\top \mathbf{X}_{n1})}$$

where $\mathbf{X}_{n1} = (X_{n10}, X_{n11}, \ldots, X_{n1M})^\top$ is the covariate vector for observation/participant n at time 1, with $X_{n10} = 1$ (intercept term) and X_{n1m} ($m = 1, \ldots, M$) representing the value of the m-th covariate. The coefficient vector $\boldsymbol{\beta}_l = (\beta_{l0}, \beta_{l1}, \ldots, \beta_{lM})^\top$ corresponds element-wise to $\mathbf{X}_{n1}$, where β_{l0} is the intercept and β_{lm} ($m \geq 1$) are regression coefficients for covariates. Class L is the reference class ($\boldsymbol{\beta}_L = \mathbf{0}$).

Transition Probabilities (with covariates and times>=2): For observation/participant n transitioning from class l at time $t-1$ to class k at time t ($t \geq 2$):

$$P(Z_{nt} = k \mid Z_{n,t-1} = l, \mathbf{X}_{nt}) = \frac{\exp(\boldsymbol{\gamma}_{lkt}^{\top} \mathbf{X}_{nt})}{\sum_{j=1}^{L} \exp(\boldsymbol{\gamma}_{ljt}^{\top} \mathbf{X}_{nt})}$$

where $\mathbf{X}_{nt} = (X_{nt0}, X_{nt1}, \ldots, X_{ntM})^{\top}$ is the covariate vector at time t, with $X_{nt0} = 1$ (intercept) and X_{ntm} ($m = 1, \ldots, M$) as the m-th covariate value. The coefficient vector $\boldsymbol{\gamma}_{lkt} = (\gamma_{lkt0}, \gamma_{lkt1}, \ldots, \gamma_{lktM})^{\top}$ corresponds element-wise to $\mathbf{X}_{nt}$, where γ_{lkt0} is the intercept and γ_{lktm} ($m \geq 1$) are regression coefficients. Class L is the reference class ($\boldsymbol{\gamma}_{lLt} = \mathbf{0}$ for all l).

Without Covariates or When times=1: Initial probabilities follow a multinomial distribution with probabilities $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_L)$. When $times \geq 2$, transitions follow a Markov process with fixed probabilities $\tau_{lk}^{(t)} = P(Z_t = k \mid Z_{t-1} = l)$, where $\sum_{k=1}^{L} \tau_{lk}^{(t)} = 1$ for each l and t.

Examples

```
######################## Example 1: Single time point (times=1) ######################
library(LCPA)
set.seed(123)
sim_single <- sim.LTA(N = 200, I = 4, L = 3, times = 1, type = "LCA")
print(sim_single)

######################## Example 2: LPA without covariates ######################
set.seed(123)
sim_lta <- sim.LTA(N = 200, I = 3, L = 3, times = 3, type = "LPA", constraint = "VE")
print(sim_lta)

################## Example 3: With custom covariates (times>=2) ######################
set.seed(123)
N <- 200 ## sample size

## Covariates at time point T1
covariates.inter <- rep(1, N) # Intercept term is always 1 for each n
covariates.X1 <- rnorm(N)     # Covariate X1 is a continuous variable
covariates.X2 <- rbinom(N, 1, 0.5) # Covariate X2 is a binary variable
covariates.X1.X2 <- covariates.X1 * covariates.X2 # Interaction between covariates X1 and X2
covariates.T1 <- cbind(inter=covariates.inter, X1=covariates.X1,
                X2=covariates.X2, X1.X2=covariates.X1.X2) # Combine into covariates at T1

## Covariates at time point T2
covariates.inter <- rep(1, N) # Intercept term is always 1 for each n
covariates.X1 <- rnorm(N)     # Covariate X1 is a continuous variable
covariates.X2 <- rbinom(N, 1, 0.5) # Covariate X2 is a binary variable
covariates.X1.X2 <- covariates.X1 * covariates.X2 # Interaction between covariates X1 and X2
covariates.T2 <- cbind(inter=covariates.inter, X1=covariates.X1,
                X2=covariates.X2, X1.X2=covariates.X1.X2) # Combine into covariates at T2

covariates <- list(t1=covariates.T1, t2=covariates.T2) # Combine into final covariates list

## Simulate beta coefficients
```

```
# 3x3 matrix (last column is zero because the last category is used as reference)
beta <- matrix(c( 0.8, -0.5, 0.0,
                 -0.3, -0.4, 0.0,
                  0.2,  0.8, 0.0,
                 -0.1,  0.2, 0.0), ncol=3, byrow=TRUE)

## Simulate gamma coefficients (only needed when times>=2)
gamma <- list(
  lapply(1:3, function(l) {
    lapply(1:3, function(k) if(k < 3)
           runif(4, -1.0, 1.0) else c(0, 0, 0, 0)) # Last class as reference
  })
)

## Simulate the data
sim_custom <- sim.LTA(
  N=N, I=4, L=3, times=2, type="LPA",
  covariates=covariates,
  beta=beta,
  gamma=gamma
)

summary(sim_custom)
```

---

summary *S3 Methods: summary*

Description

Generates structured, comprehensive summaries of objects produced by the LCPA package. This generic function dispatches to class-specific methods that extract and organize key information including model configurations, fit statistics, parameter estimates, simulation truths, and diagnostics. Designed for programmatic access and downstream reporting.

Usage

```
## S3 method for class 'LCA'
summary(object, digits = 4, I.max = 5, ...)

## S3 method for class 'LPA'
summary(object, digits = 4, I.max = 5, ...)

## S3 method for class 'LTA'
summary(object, digits = 4, ...)

## S3 method for class 'LCPA'
summary(object, digits = 4, ...)
```

```
## S3 method for class 'sim.LCA'
summary(object, digits = 4, I.max = 5, ...)

## S3 method for class 'sim.LPA'
summary(object, digits = 4, I.max = 5, ...)

## S3 method for class 'sim.LTA'
summary(object, digits = 4, I.max = 5, L.max = 5, ...)

## S3 method for class 'fit.index'
summary(object, digits = 4, ...)

## S3 method for class 'compare.model'
summary(object, digits = 4, ...)

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

Arguments

`object`	An object of one of the following classes: • Model objects: `LCA`, `LPA`, `LCPA`, `LTA` • Simulation objects: `sim.LCA`, `sim.LPA`, `sim.LTA` • Fit/comparison objects: `get.fit.index`, `compare.model` • Standard error objects: `get.SE`
`digits`	Number of decimal places for numeric output (default: 4). Applied universally across all methods.
`I.max`	Maximum number of variables/items to display (`LCA`, `LPA`, `sim.LCA`, `sim.LPA`, `sim.LTA`, `LCPA`, `LTA`, and `compare.model` only; default: 5). Controls verbosity for high-dimensional outputs.
`...`	Additional arguments passed to or from other methods (currently ignored).
`L.max`	Maximum number of latent classes/profiles to display before truncation (`sim.LTA` only; default: 5). Useful when models have many latent groups. Ignored for other classes.

Details

Each method returns a named list with class-specific components optimized for structured access:

`LCA` Returns a `summary.LCA` object with components:

`call` Original function call.

`model.config` List: `latent_classes`, `method`.

`data.info` List: `N`, `I`, `poly.value`, `uniform_categories`.

`fit.stats` List: `LogLik`, `AIC`, `BIC`, `entropy`, `npar`.

`class.probs` Data frame: `Class`, `Count`, `Proportion`.

`item.probs` List of matrices (first `I.max` items) with conditional probabilities per class/category.

convergence List: algorithm, iterations, tolerance, loglik change, hardware (if applicable).

replication List: nrep, best_BIC (if multiple replications performed).

digits, I.max.shown, total.items Metadata for printing/formatting.

LPA Returns a summary.LPA object with components:

call Original function call.

model.config List: latent_profiles, constraint, cov_structure, method.

data.info List: N, I, distribution.

fit.stats List: LogLik, AIC, BIC, entropy, npar.

class.probs Data frame: Profile, Count, Proportion.

class.means Matrix (first I.max variables) of profile-specific means.

convergence List: algorithm, iterations, tolerance, loglik change, hardware (if applicable).

replication List: nrep, best_BIC (if multiple replications performed).

digits, I.max.shown, total.vars Metadata for printing/formatting.

LCPA Returns a summary.LCPA object with components:

call Original function call.

model.config List: latent_classes, model_type, reference_class, covariates_mode, CEP_handling.

data.info List: sample_size, variables.

fit.stats List: LogLik, AIC, BIC, npar.

class.probs Data frame: Class, Probability, Proportion, Frequency.

coefficients Data frame: regression coefficients for non-reference classes (Estimate, Std_Error, z_value, p_value, 95% CI).

reference_class Integer: reference class for multinomial logit.

convergence List: iterations, coveraged, converg_note.

digits, I.max.shown, total.vars, has.covariates Metadata for printing/formatting.

LTA Returns a summary.LTA object with components:

call Original function call.

model.config List: time_points, latent_classes, model_type, reference_class, covariates_mode, CEP_handling, transition_mode.

data.info List: sample_size, variables, time_points.

fit.stats List: LogLik, AIC, BIC, npar.

class.probs List of data frames (per time point): Class, Probability, Proportion, Frequency.

initial_model List: coefficients (data frame), covariate_names, reference_class.

transition_models Named list of data frames: transition coefficients per time interval (From_Class, To_Class, Estimate, Std_Error, etc.).

reference_class Integer: reference destination class for transitions.

convergence List: iterations, coveraged, converg_note.

digits, I.max.shown, total.vars, covariates.timeCross Metadata for printing/formatting.

sim.LCA Returns a summary.sim.LCA object with components:

call Original simulation call.

config List: N, I, L, poly.value, uniform_categories, IQ, distribution.

`class.probs` Data frame: `Class`, `Probability`, `Frequency`.

`item.probs` List of matrices (first `I.max` items) with true conditional probabilities per class/category.

`digits`, `I.max.shown`, `total.vars` Metadata for printing/formatting.

`sim.LPA` Returns a `summary.sim.LPA` object with components:

`call` Original simulation call.

`config` List: `N`, `I`, `L`, `constraint`, `constraint_desc`, `distribution`.

`class.probs` Data frame: `Profile`, `Probability`, `Frequency`.

`class.means` Matrix (first `I.max` variables) of true profile-specific means.

`cov_structure` Character: detailed description of covariance constraints.

`digits`, `I.max.shown`, `total.vars` Metadata for printing/formatting.

`sim.LTA` Returns a `summary.sim.LTA` object with components:

`call` Original simulation call.

`config` List: `N`, `I`, `L`, `times`, `type`, `distribution`, `constraint` (if LPA).

`class.probs` List of data frames (per time point): `Class`, `Probability`, `Frequency`.

`item.probs` Nested list (by time/item) of true conditional probabilities (if `type="LCA"`).

`class.means` List of matrices (by time) of true profile means (if `type="LPA"`).

`transition` List: `mode` ("fixed" or "covariate"), `rate` or `beta`/`gamma` coefficients, `time_points`.

`covariates` List of data frames (per time point) with covariate summaries (Min, Max, Mean), if present.

`digits`, `I.max.shown`, `L.max.shown`, `total.vars`, `total.classes` Metadata for printing/formatting.

`fit.index` Returns a `summary.fit.index` object with components:

`call` Function call that generated the fit indices.

`data.info` List: `N`.

`fit.table` Data frame: `Statistic`, `Value`, `Description` for -2LL, AIC, BIC, SIC, CAIC, AWE, SABIC.

`digits` Numeric: precision used for formatting.

`compare.model` Returns a `summary.compare.model` object with components:

`call` Function call that generated the comparison.

`data.info` List: `N`, `I`, `L` (named vector for two models).

`fit.table` Data frame comparing fit indices for both models.

`model_comparison` Data frame: `Classes`, `npar`, `AvePP`, `Entropy`.

`BF` Numeric: Bayes Factor value (if computed).

`BF_interpretation` Character: interpretive guidance for Bayes Factor.

`lrt_table` Data frame: `Test`, `Statistic`, `DF`, `p-value`, `Sig` (significance markers).

`lrt_objects` List: raw hypothesis test objects for further inspection.

`digits` Numeric: precision used for formatting.

`SE` Returns a `summary.SE` object with components:

`call` Original function call.

`method` Character: "Obs" or "Bootstrap".

`diagnostics` List: method-specific diagnostic info (e.g., n.Bootstrap, hessian_cond_number).

`model_type` Character: "LCA" or "LPA".

`L` Integer: number of latent classes/profiles.

`I` Integer: number of variables/items (NA if unknown).

`nonzero_counts` List: counts of non-zero SEs by parameter type (P.Z, means/par, covs).

`total_PZ` Integer: total number of class probability parameters.

Value

Invisibly returns a structured list containing summary components. The exact structure depends on the class of `object`. All returned objects carry an appropriate S3 class (e.g., `summary.LCA`, `summary.LPA`) for use with corresponding `print` methods.

Methods (by class)

- `summary(LCA)`: Summary method for `LCA` objects
- `summary(LPA)`: Summary method for `LPA` objects
- `summary(LTA)`: Summary method for `LTA` objects
- `summary(LCPA)`: Summary method for `LCPA` objects
- `summary(sim.LCA)`: Summary method for `sim.LCA` objects
- `summary(sim.LPA)`: Summary method for `sim.LPA` objects
- `summary(sim.LTA)`: Summary method for `sim.LTA` objects
- `summary(fit.index)`: Summary method for `fit.index` objects
- `summary(compare.model)`: Summary method for `compare.model` objects
- `summary(SE)`: Summary method for `summary.SE` objects

`update` *S3 Methods: update*

Description

The `update` function provides a unified and convenient interface to refresh or modify existing objects generated by the `LCPA` package. It allows users to re-run model fitting or data simulation with new parameter settings while preserving all other original configurations. Supported classes include: `LCA`, `LPA`, `LCPA`, `LTA`, `sim.LCA`, `sim.LPA`, and `sim.LTA`.

Usage

```
update(x, ...)

## S3 method for class 'LCA'
update(x, ...)

## S3 method for class 'LPA'
update(x, ...)
```

```
## S3 method for class 'LCPA'
update(x, ...)

## S3 method for class 'LTA'
update(x, ...)

## S3 method for class 'sim.LCA'
update(x, ...)

## S3 method for class 'sim.LPA'
update(x, ...)

## S3 method for class 'sim.LTA'
update(x, ...)
```

Arguments

x	An object of one of the following classes: • LCA — Latent Class Analysis model. • LPA — Latent Profile Analysis model. • LCPA — Latent Class Prediction Analysis (with covariates). • LTA — Latent Transition Analysis model. • sim.LCA — Simulated LCA dataset. • sim.LPA — Simulated LPA dataset. • sim.LTA — Simulated LTA dataset.
...	Additional named arguments passed to override or extend the original call. Valid arguments depend on the class of x: LCA response, L, par.ini, method, is.sort, nrep, vis, control.EM, control.Mplus, control.NNE LPA response, L, par.ini, constraint, method, is.sort, nrep, vis, control.EM, control.Mplus, control.NNE LCPA response, L, ref.class, type, covariates, CEP.error, par.ini, params, is.sort, constraint, method, tol, maxiter, nrep, starts, maxiter.wa, vis, control.EM, control.Mplus, control.NNE LTA responses, L, ref.class, type, covariates, CEP.timeCross, CEP.error, covariates.timeCross, par.ini, params, is.sort, constraint, method, tol, maxiter, nrep, starts, maxiter.wa, vis, control.EM, control.Mplus, control.NNE sim.LCA N, I, L, poly.value, IQ, distribution, params, is.sort sim.LPA N, I, L, constraint, distribution, mean.range, covs.range, params, is.sort sim.LTA N, I, L, times, type, rate, constraint, distribution, mean.range, covs.range, poly.value, IQ, covariates, beta, gamma, params, is.sort

Details

Internally, each method extracts the stored `arguments` list from the input object `x`, merges it with user-provided `...` using `modifyList`, then re-invokes the corresponding constructor function (`LCA()`, `LPA()`, `LCPA()`, `LTA()`, `sim.LCA()`, etc.) with the merged argument list.

This ensures that:

- Only explicitly overridden parameters are changed.
- Default values from the original call remain intact.
- Complex nested structures (e.g., control lists) can be partially updated.

Note: If an invalid argument is passed (e.g., `constraint` to `LCA`), it will be silently ignored unless the underlying constructor validates inputs.

Value

An object of the same class as `x`, reconstructed using the original arguments updated with any provided in `....` All unchanged parameters are preserved from the original call.

Methods (by class)

- `update(LCA)`: Update method for `LCA` objects
- `update(LPA)`: Update method for `LPA` objects
- `update(LCPA)`: Update method for `LCPA` objects
- `update(LTA)`: Update method for `LTA` objects
- `update(sim.LCA)`: Update method for `sim.LCA` objects
- `update(sim.LPA)`: Update method for `sim.LPA` objects
- `update(sim.LTA)`: Update method for `sim.LTA` objects

Examples

```
library(LCPA)

# --- Update LCA ---
data <- sim.LCA(N=500, I=5, L=3)
lca.obj <- LCA(data$response, L=3)
lca.updated <- update(lca.obj, method="EM", nrep=5)

# --- Update LPA ---
data2 <- sim.LPA(N=300, I=4, L=2)
lpa.obj <- LPA(data2$response, L=2, constraint="VE")
lpa.updated <- update(lpa.obj, constraint="VV")

# --- Update Simulation Objects ---
sim.obj1 <- sim.LCA(N=1000)
sim.obj1_updated <- update(sim.obj1, N=2000, IQ=0.8)

sim.obj2 <- sim.LPA(I=6)
sim.obj2_updated <- update(sim.obj2, I=8, mean.range=c(-2,2))
```

```
sim.obj3 <- sim.LTA(N=200, I=5, L=2, times=3)
sim.obj3_updated <- update(sim.obj3, N=300, times=4, constraint="ER")
```

Index