Package: LCPA (via r-universe)

October 7, 2026

Type Package

Title A General Framework for Latent Class and Profile Analysis

Version 1.0.4

Date 2026-09-2

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 Provides a unified framework for finite-mixture latent variable models, including latent class analysis (LCA), latent profile analysis (LPA), latent class/profile analysis with covariates, and latent transition analysis (LTA), within one consistent interface. Estimation methods include the expectation-maximization (EM) algorithm; neural network estimation, which requires 'Python' and its dependent libraries; integration with 'Mplus', which requires an installed copy of 'Mplus'; and stochastic EM (SEM) through the optional 'flexmix', 'Rmixmod', and 'RMixtComp' backends. 'flexmix' and the default 'Rmixmod' path perform configurable warm-up trajectories and promote the best candidates to full SEM replications. 'Rmixmod' additionally exposes its native strategy interface, including chained SEM-to-EM estimation, whereas 'RMixtComp' exposes its native SEM and Gibbs controls without the external warm-up stage. Model assessment includes the Akaike information criterion (AIC), Bayesian information criterion (BIC), Schwarz information criterion (SIC), consistent AIC (CAIC), approximate weight of evidence (AWE), sample-size-adjusted BIC (SABIC), entropy, and average posterior probabilities. Model-comparison procedures include the ordinary likelihood-ratio test, the Mplus TECH11 Vuong-Lo-Mendell-Rubin and adjusted Lo-Mendell-Rubin tests, and fixed-replicate or sequential parametric bootstrap likelihood-ratio tests. Standard errors can be estimated by nonparametric bootstrap, numerical observed information, or

analytic observed information based on Louis' identity. Classification-error-adjusted maximum-likelihood and Bolck-Croon-Hagenaars three-step methods support covariates predicting latent membership, initial-status and transition regressions, and latent classes or states predicting continuous and categorical external observed dependent variables. Simulation, posterior-probability, classification-error, extraction, summary, plotting, model-adjustment, and update utilities are also provided for reproducible workflows.

License GPL-3

Depends R (>= 4.1.0)

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

Suggests flexmix, Rmixmod, RMixtComp, RMixtCompUtilities

LinkingTo Rcpp, RcppArmadillo

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' 'three.step.utils.R' 'ZY.utils.R' 'ML.ZY.utils.R' 'BCH.XZ.utils.R' 'install_python_dependencies.R' 'Kmeans.LCA.R' 'Kmeans.LPA.R' 'LCA.R' 'ML.XZ.LCPA.R' 'BCH.XZ.LCPA.R' 'XZ.LCPA.R' 'BCH.ZY.LCPA.R' 'ML.ZY.LCPA.R' 'ZY.LCPA.R' 'LCPA.R' 'logit.R' 'LPA.R' 'ML.XZ.LTA.R' 'BCH.XZ.LTA.R' 'XZ.LTA.R' 'BCH.ZY.LTA.R' 'ML.ZY.LTA.R' 'ZY.LTA.R' 'LTA.R' 'LRT.test.R' 'LRT.test.Bootstrap.R' 'LRT.test.VLMR.R' 'Mplus.LCA.R' 'Mplus.LPA.R' 'normalize.R' 'plotResponse.R' 'Rmixmod.utils.R' 'Rmixmod.LCA.R' 'Rmixmod.LPA.R' 'RMixtComp.utils.R' 'RMixtComp.LCA.R' 'RMixtComp.LPA.R' 'flexmix.utils.R' 'flexmix.LCA.R' 'flexmix.LPA.R' 'RcppExports.R' 'rdirichlet.R' 'S3extract.R' 'S3plot.R' 'S3print.R' 'S3summary.R' 'S3update.R' 'simulation.utils.R' 'sim.correlation.R' 'sim.LCA.R' 'sim.LPA.R' 'sim.LTA.R' 'tools.R' 'utils.R' 'zzz.R'

Config/roxygen2/version 8.1.0

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

Date/Publication 2026-09-06 22:50:10 UTC

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

RemoteRef HEAD

RemoteSha 44e94bb23147684ce84100d28fa4e42d7c295dc1

Contents

adjust.model . . . 4
adjust.response . . . 5
check.response . . . 7
compare.model . . . 8
extract . . . 10
get.AvePP . . . 16
get.CEP . . . 18
get.entropy . . . 21
get.fit.index . . . 23
get.Log.Lik.LCA . . . 24
get.Log.Lik.LPA . . . 25
get.Log.Lik.LTA . . . 26
get.npar.LCA . . . 29
get.npar.LPA . . . 30
get.npar.LTA . . . 32
get.P.Z.Xn.LCA . . . 34
get.P.Z.Xn.LPA . . . 35
get.SE . . . 37
install_python_dependencies . . . 38
Kmeans.LCA . . . 40
Kmeans.LPA . . . 41
LCA . . . 42
LCPA . . . 52
logit . . . 62
LPA . . . 63
LRT.test . . . 73
LRT.test.Bootstrap . . . 75
LRT.test.VLMR . . . 78
LTA . . . 80
normalize . . . 95
plot . . . 97
plotResponse . . . 98
print . . . 99
rdirichlet . . . 103
sim.correlation . . . 104
sim.LCA . . . 107
sim.LPA . . . 110
sim.LTA . . . 113
summary . . . 117
update . . . 122

Index **126**

`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 Python

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 latent class/profile models using multiple fit indices, likelihood ratio tests, and classification metrics. The models may have any class counts, including equal class counts.

Usage

```
compare.model(object1, object2, nrep.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`.
`nrep.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 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 and adjusted LMR tests, see `LRT.test.VLMR`
  - Parametric bootstrap LRT (computationally intensive but robust), see `LRT.test.Bootstrap`
- Computes a Bayes factor approximation from the difference in `SIC` = `-0.5 * BIC`

Important requirements:

- Both models must be of the same type (LCA or LPA)
- The three likelihood-ratio procedures accept any pair of class counts, including equal class counts. The VLMR component requires models fitted to the same observations in the same order; LPA models must also use the same covariance constraint.
- nrep.bootstrap > 0 requires significant computational resources

Value

An object of class compare.model containing:

N Named vector of sample sizes for the ordered models.

I Named vector of indicator counts for the ordered models.

L Named vector of latent class/profile counts for the ordered models.

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 Mplus TECH11 VLMR and adjusted LMR test results

LRT.Bootstrap.obj Bootstrap LRT results (if nrep.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

# 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)
```

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 — One non-missing, non-empty character string specifying the component to extract. Valid choices depend on the class of object; see Details for the complete class-specific listings.

... — Reserved for S3 method compatibility; no additional arguments are used.

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.
- category.levels List of the ordered observed response categories for each indicator.
- 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 Backend model object for method="NNE", method="Mplus", method="flexmix", method="Rmixmod", or method="RMixtComp". The latter three are SEM fits only.
- 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` Backend model object (neural network, Mplus, flexmix SEM, Rmixmod SEM, or RMixtComp SEM).

`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.

`vcov` Variance-covariance matrix of the free Step 3 coefficients.

`information` Observed information matrix for `method.SE="Numeric"` or `"Analytic"`; otherwise `NULL`.

`SE.diagnostics` Diagnostics for the selected standard-error method.

`bound.diagnostics` Indices of Step 3 coefficients at or near an optimization bound.

`iterations` Optimization iterations in Step 3.

`converged` Logical indicator of successful NLopt termination.

`Log.Lik.history` Step 3 log-likelihood history.

`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).

`step1.pool` Logical indicating whether Step 1 used the row-bound responses from all time points.

`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.

`vcov` Variance-covariance matrix of the free Step 3 coefficients.

`information` Observed information matrix for `method.SE="Numeric"` or `"Analytic"`; otherwise `NULL`.

`SE.diagnostics` Diagnostics for the selected standard-error method.

`bound.diagnostics` Indices of Step 3 coefficients at or near an optimization bound.

`iterations` Optimization iterations in Step 3.

`converged` Logical indicator of successful NLopt termination.

`Log.Lik.history` Step 3 log-likelihood history.

`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.

`ref.class` Reference class for the true regression coefficients.

`call` Original simulation function call.

`arguments` Input arguments used in simulation.

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

`N` Sample size used to compute sample-size-dependent indices.

`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` Schwarz information criterion on the log-likelihood scale ($-0.5 \times BIC$).

`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:

`N`, `I`, `L` Named vectors giving sample size, indicator count, and class count for each model.

`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` Mplus TECH11 VLMR and adjusted LMR test results.

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

`call` The matched function call used for comparison.

`arguments` List containing original input arguments (`object1`, `object2`, `nrep.bootstrap`).

`SE` Standard error estimation results. Available components:

`se` List containing standard errors for parameters (components depend on model type).

`vcov` Variance-covariance matrix for `method="Obs"` or `"Louis"`; `NULL` for bootstrap.

`hessian` Observed information matrix for `method="Obs"` or `"Louis"`; `NULL` for bootstrap.

`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 available when method = "Obs" or "Louis". 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 participant n belongs to class l given the observed indicator vector $\mathbf{X}_n$

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,

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

  where N_l is the number of participants assigned to class l, and $\widehat{Z}_n = \arg\max_h p_{nh}$.
- 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.

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

- 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. doi: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 classification error probability (CEP) matrices for bias-corrected three-step latent class/profile and latent transition models with covariates.

Usage

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

Arguments

`P.Z.Xns`	A list of length T, where T is the number of time points. Each element must be an $N \times L$ matrix of posterior class-membership probabilities $\tau_{ntl} = P(Z_{nt} = l \mid \mathbf{X}_{nt})$, with: • rows corresponding to participants ($n = 1, \ldots, N$); • columns corresponding to latent classes ($l = 1, \ldots, L$); • each row expected to sum to 1. All matrices are expected to have identical dimensions and to be ordered chronologically.
`CEP.time.cross`	Logical. If `FALSE`, a separate CEP matrix is estimated for each time point. If `TRUE` (default), posterior-weighted classification information is pooled across all time points to estimate a single time-invariant CEP matrix, which is then returned for every time point.

Details

The CEP matrix describes the relationship between the latent class and the modal class assignment obtained from posterior class-membership probabilities.

For participant n at time t, the modal class assignment is defined as

$$\widehat{Z}_{nt} = \arg\max_{l} \tau_{ntl}.$$

For time point t, the CEP matrix is estimated as

$$\mathrm{CEP}_t(l,k) = P(\widehat{Z}_{nt} = k \mid Z_{nt} = l) = \frac{\sum_{n=1}^{N} \mathbb{1}(\widehat{Z}_{nt} = k)\tau_{ntl}}{\sum_{n=1}^{N} \tau_{ntl}}.$$

Equivalently, defining the posterior-weighted class prevalence as

$$\hat{\pi}_{tl} = \frac{1}{N}\sum_{n=1}^{N} \tau_{ntl},$$

the estimator can be written as

$$\mathrm{CEP}_t(l,k) = \frac{\sum_{n=1}^{N} \mathbb{1}(\widehat{Z}_{nt} = k)\tau_{ntl}}{N\hat{\pi}_{tl}}.$$

The resulting matrix is oriented as

$$\mathrm{CEP}_t[\mathrm{true}, \mathrm{assigned}],$$

so that, in the Step-3 likelihood, the probability associated with an observed modal assignment $\widehat{Z}_{nt}$ under candidate latent class l is obtained from

$$\mathrm{CEP}_t(l, \widehat{Z}_{nt}).$$

In other words, the column corresponding to the participant's observed modal assignment is selected, and the entries across rows give the corresponding classification probabilities under the candidate latent classes.

If `CEP.time.cross = FALSE`, the above estimator is computed separately for each time point.

If `CEP.time.cross = TRUE`, the numerator and denominator are pooled across time points before normalization:

$$\mathrm{CEP}_{\mathrm{pool}}(l,k) = \frac{\sum_{t=1}^{T}\sum_{n=1}^{N} \mathbb{1}(\widehat{Z}_{nt} = k)\tau_{ntl}}{\sum_{t=1}^{T}\sum_{n=1}^{N} \tau_{ntl}}.$$

Therefore, the pooled CEP is not the simple arithmetic mean of the time-specific CEP matrices. Rather, it is equivalent to a class-specific posterior-mass-weighted combination of the time-specific matrices.

Using a common pooled CEP across time assumes that the classification-error mechanism

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

is sufficiently stable across time for a common CEP matrix to be appropriate. This classification-error invariance assumption is conceptually related to, but is not equivalent to, longitudinal measurement invariance.

Value

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

- rows index the latent ("true") class;
- columns index the modal ("assigned" or "predicted") class;
- entry (l, k) estimates $P(\widehat{Z}_{nt} = k \mid Z_{nt} = l)$.

Thus, each row is a conditional probability distribution over modal assignments given a latent class and sums to 1 up to numerical precision. Column sums are not constrained to equal 1.

When `CEP.time.cross = TRUE`, all elements of the returned list contain the same pooled CEP matrix.

List elements are named `"t1"`, `"t2"`, ..., `"tT"`.

Note

- The function assumes complete posterior-probability matrices; missing values are not explicitly handled.
- All elements of `P.Z.Xns` are expected to have the same numbers of individuals and latent classes.
- Posterior-probability rows are expected to sum to 1; this condition is not explicitly checked by the function.
- Modal assignment is obtained using `which.max()`. If two or more classes have exactly equal maximum posterior probabilities, `which.max()` selects the first maximum.
- If a latent class has zero posterior-weighted prevalence, the CEP for that class is undefined and division by zero may occur.

References

Liang, Q., de la Torre, J., & Law, N. (2023). Latent transition cognitive diagnosis model with covariates: A three-step approach. *Journal of Educational and Behavioral Statistics, 48*(6), 690–718. doi:10.3102/10769986231163320

Examples

```
# Simulate posterior probabilities for two time points,
# three latent classes, and 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
)

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

# Pooled CEP matrix used at all time points
cep_pooled <- get.CEP(
  P.Z.Xns,
  CEP.time.cross = TRUE
)

# Under the [true class, assigned class] orientation,
# rows should sum to one
rowSums(cep_time_specific[[1]])
```

`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 participant n belonging to class l

Note

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

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. doi:10.1080/10705511.2013.824786

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

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:

N Number of observations used to fit the model

npar Number of free parameters in the model

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

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

$$\begin{cases} -2 \sum_{n=1}^{N} \log \left\{ \sum_{l=1}^{L} \pi_l \prod_{i=1}^{I} P(X_{ni} = x_{ni} \mid Z_n = l) \right\}, & \text{for LCA,} \\ -2 \sum_{n=1}^{N} \log \left\{ \sum_{l=1}^{L} \pi_l \mathcal{N}(\mathbf{X}_n \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l) \right\}, & \text{for LPA,} \end{cases}$$

where π_l is the prior probability of class l, and $\mathbf{X}_n$ is participant n's indicator vector.

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

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

SIC Schwarz information criterion on the log-likelihood scale: $\text{SIC} = -\frac{1}{2}\text{BIC}$. Equivalent to $\log \mathcal{L} - \frac{npar}{2} \log(N)$. Often used in latent class modeling.

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

AWE Approximate Weight of Evidence: $\text{AWE} = -2 \log \mathcal{L} + npar\,[\log(N) + 1.5]$. 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.

arguments List containing the fitted model as `object`.

call Matched function call.

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, par, P.Z)
```

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
`par`	A 3-dimensional array of dimension $L \times I \times K_{\max}$ containing conditional probabilities, where $par[l, i, q]$ represents $P(X_i = q - 1 \mid Z = l)$ (after internal 0-based re-encoding). Must satisfy: • For each class l and indicator i: $\sum_{q=1}^{K_i} par[l, i, q] = 1$ • Probabilities for non-existent categories (where $q > K_i$) are ignored but must be present in the array

`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 $l = 1, \ldots, L$

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-Conditional Likelihood Contribution: For each participant n and class l, local independence gives

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

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

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

- Total Observed-Data Log-Likelihood: Sum the log marginal contributions across participants:

$$\log \mathcal{L}_{\mathrm{LCA}} = \sum_{n=1}^{N} \log \left\{ \sum_{l=1}^{L} \pi_l \prod_{i=1}^{I} P(X_{ni} = x_{ni} \mid Z_n = l) \right\}.$$

Value

A single numeric value equal to the total observed-data log-likelihood $\log \mathcal{L}_{\mathrm{LCA}}$ defined below.

`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.

Usage

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

Arguments

`response`	A numeric matrix of dimension $N \times I$ containing continuous observations. Rows represent observations, columns represent variables. Missing values are not permitted.
`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.
`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 $l = 1, \ldots, L$

Details

The log-likelihood calculation follows these steps:

- Class-Conditional Likelihood Contribution: For participant n in profile l, the multivariate normal log density is

$$\log \mathcal{N}(\mathbf{X}_n \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l) = -\frac{I}{2}\log(2\pi) - \frac{1}{2}\log|\boldsymbol{\Sigma}_l| - \frac{1}{2}(\mathbf{X}_n - \boldsymbol{\mu}_l)^\top \boldsymbol{\Sigma}_l^{-1}(\mathbf{X}_n - \boldsymbol{\mu}_l).$$

- Participant-Level Marginal Likelihood Contribution: The marginal density for participant n is

$$P(\mathbf{X}_n) = \sum_{l=1}^{L} \pi_l \mathcal{N}(\mathbf{X}_n \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l).$$

- Total Observed-Data Log-Likelihood:

$$\log \mathcal{L}_{\mathrm{LPA}} = \sum_{n=1}^{N} \log \left\{ \sum_{l=1}^{L} \pi_l \mathcal{N}(\mathbf{X}_n \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l) \right\}.$$

Value

A single numeric value equal to the total observed-data log-likelihood $\log \mathcal{L}_{\mathrm{LPA}}$ defined below.

`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,
  ref.class,
  covariates.time.cross = FALSE
)
```

Arguments

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

- `beta`: Matrix of size $(U_1 + 1) \times L$ with coefficients for the initial class membership multinomial logit model (time 1). The coefficient vector for the selected reference class is constrained to the zero vector.
- `gamma`: Nested `list` of transition coefficients. For transition to time t (from time $t-1$ to t, where $t = 2, \ldots, T$):

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

  Coefficients for transitions to the selected reference class are constrained to zero vectors for every origin class.

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

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

where $\widehat{Z}_{nt}$ is the modal class assignment and Z_{nt} is the 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 participant 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 $\widehat{Z}_{nt}$ for participant n at time t.

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

`ref.class` Integer between 1 and `L`; zero-coded reference class in `params$beta` and the destination-specific `params$gamma` coefficients.

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

Details

The log-likelihood calculation follows these steps:

1. Initial class probabilities (time 1): For participant n, compute using multinomial logit with covariates $\boldsymbol{\zeta}_{n1}$:
$$P(Z_{n1} = l \mid \boldsymbol{\zeta}_{n1}) = \frac{\exp(\boldsymbol{\beta}_l^\top \boldsymbol{\zeta}_{n1})}{\sum_{h=1}^{L} \exp(\boldsymbol{\beta}_h^\top \boldsymbol{\zeta}_{n1})}$$
where $\boldsymbol{\beta}_{l_0} = \mathbf{0}$ for reference class l_0 represented in `params`.
2. Transition Probabilities (times $t \geq 2$): For transition from class k at time $t-1$ to class l at time t:
$$P(Z_{nt} = l \mid Z_{n,t-1} = k, \boldsymbol{\zeta}_{nt}) = \frac{\exp(\boldsymbol{\gamma}_{klt}^\top \boldsymbol{\zeta}_{nt})}{\sum_{h=1}^{L} \exp(\boldsymbol{\gamma}_{kht}^\top \boldsymbol{\zeta}_{nt})}$$
where $\boldsymbol{\gamma}_{k,l_0,t} = \mathbf{0}$ for every origin class k and reference destination l_0.
3. Scaled forward recursion: Initialize
$$\alpha_{n1}(l) = P(Z_{n1} = l \mid \boldsymbol{\zeta}_{n1}) \mathrm{CEP}_1(l, \widehat{Z}_{n1})$$
and recursively compute
$$\alpha_{nt}(l) = \mathrm{CEP}_t(l, \widehat{Z}_{nt}) \sum_{k=1}^{L} \alpha_{n,t-1}(k) P(Z_{nt} = l \mid Z_{n,t-1} = k, \boldsymbol{\zeta}_{nt}).$$
This forward recursion marginalizes exactly over all latent paths without explicitly constructing the L^T paths.
4. Analytic score: Optimization uses the corresponding scaled backward recursion to obtain state and transition posterior probabilities for the exact multinomial-logit score.

Value

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

$$\log \mathcal{L}(\boldsymbol{\beta}, \boldsymbol{\gamma}) = \sum_{n=1}^{N} \log \Bigg[\sum_{z_{n1}=1}^{L} \cdots \sum_{z_{nT}=1}^{L} \Big(\prod_{t=1}^{T} \mathrm{CEP}_t(z_{nt}, \widehat{Z}_{nt}) \Big) \cdot P(Z_{n1} = z_{n1} \mid \boldsymbol{\zeta}_{n1}) \cdot \prod_{t=2}^{T} P(Z_{nt} = z_{nt} \mid Z_{n,t-1} = z_{n,t-1}, \boldsymbol{\zeta}_{nt}) \Bigg]$$

where $z_{n1}, \ldots, z_{nT}$ is a latent class path and `Zs[[t]][n]` stores the modal assignment $\widehat{Z}_{nt}$.

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: $L \sum_{i=1}^{I}(K_i - 1)$ 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 $P(X_i = q \mid Z = l)$
- With constraints $\sum_{q=1}^{K_i} P(X_i = q \mid Z = l) = 1$ for each class l
- The L class-specific sum-to-one constraints leave $L(K_i - 1)$ free parameters per variable

Value

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

$$\text{npar} = L \sum_{i=1}^{I}(K_i - 1) + (L - 1)$$

Examples

```
# Example 1: 3 binary variables (K_i=2), 2 latent classes
poly.value <- c(2, 2, 2)  # Three binary variables
L <- 2
npar <- L * sum(poly.value - 1) + (L - 1)  # = 2 * 3 + 1 = 7
get.npar.LCA(poly.value, L)  # Returns 7

# Example 2: Mixed variable types (binary, ternary, quaternary)
```

```
poly.value <- c(2, 3, 4)  # Variables with 2, 3, and 4 categories
L <- 3
npar <- L * sum(poly.value - 1) + (L - 1)  # = 3 * (1+2+3) + 2 = 20
get.npar.LCA(poly.value, L)  # Returns 20

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

`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.time.cross = FALSE)
```

Arguments

`covariates.ncol`
: An integer vector of length T (number of time points). Each element equals $U_t + 1$: U_t observed covariates plus the all-ones intercept column at time t.

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

`covariates.time.cross`
: 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., $U_2 = U_3 = \cdots = U_T$). If `FALSE` (default), each transition has its own set of coefficients.

Details

Parameterization:

Initial status model (time 1): Multinomial logit model with L classes (one class is the reference). Number of free parameters: $(U_1 + 1)(L - 1)$.

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 $U_{t+1}+1$. Total per transition: $L(L-1)(U_{t+1}+1)$ parameters. The constraint `covariates.time.cross` determines whether these parameters are shared across transitions.

Value

Integer representing the total number of free parameters:

$$npar = (U_1 + 1)(L - 1) + \begin{cases} L(L-1)(U_2+1) & \text{if } T > 1 \text{ and time-invariant effects} \\ \sum_{t=2}^{T} L(L-1)(U_t+1) & \text{if } T > 1 \text{ and time-varying effects} \\ 0 & \text{if } T = 1 \end{cases}$$

where:

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

Note

Critical assumptions:

- One latent class is selected as the reference category for all logits; this choice does not change the count.
- When `covariates.time.cross = TRUE`, it is assumed that all time points after the first have identical covariate structures ($U_2 = U_3 = \cdots = U_T$). If violated, the function requires the transition design matrices at times 2 through T to have the same number of columns.
- 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; this determines the shared transition block
covariates.ncol <- c(2, 3)
L <- 2
get.npar.LTA(covariates.ncol, L, covariates.time.cross = TRUE)

# Example 2: Same as above but time-varying coefficients
get.npar.LTA(covariates.ncol, L, covariates.time.cross = 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.time.cross = 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.time.cross = 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, category.levels)
```

Arguments

`response`	Numeric matrix ($N \times I$) of categorical responses.
`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, q]` = $P(X_i = q - 1 \mid Z = l)$ (using 1-based array indexing for the dimension corresponding to category $q - 1$).
`P.Z`	Vector of length L with fixed class prior probabilities (π_l). These values are used directly without re-estimation.
`category.levels`	List of length I. Element i contains the ordered response categories for indicator i as fixed in Step 1. Categories absent from the current sample remain in this mapping.

Details

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

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

where the class probabilities and conditional response probabilities are fixed by the `P.Z` and `par` arguments.

Value

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

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$P.Z,
  category.levels = fit$params$category.levels
)
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:

$$\tau_{nl} = P(Z_n = l \mid \mathbf{X}_n) = \frac{\pi_l \mathcal{N}(\mathbf{X}_n \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}{\sum_{h=1}^{L} \pi_h \mathcal{N}(\mathbf{X}_n \mid \boldsymbol{\mu}_h, \boldsymbol{\Sigma}_h)}$$

where the profile probabilities, means, and covariance matrices are fixed by the `P.Z`, `means`, and `covs` arguments.

Value

Numeric matrix ($N \times L$) of posterior probabilities. Rows sum to 1. Columns are named `"Profile 1"`, `"Profile 2"`, and so on.

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 standard errors (SEs) for parameters estimated by LCA() or LPA(). Three methods are available:

- "Bootstrap": Nonparametric bootstrap with label-switching correction.
- "Obs": Numerical observed information from adaptive central differences of the analytic observed-data score.
- "Louis": Analytic observed information based on Louis' identity.

Usage

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

Arguments

object	An object of class "LCA" or "LPA" returned by LCA() or LPA().
method	Character specifying "Bootstrap", "Obs", or "Louis".
nrep.bootstrap	Integer. Number of successful bootstrap replicates when method = "Bootstrap" (default: 100).
vis	Logical. If TRUE, displays bootstrap progress (default: TRUE).

Details

Class proportions and LCA conditional probabilities are represented by additive log-ratios with the final category as reference. Standard errors for every probability, including the reference probability, are obtained using the full softmax Jacobian.

For LPA, the observed-information parameterization includes only covariance parameters that are free under the fitted constraint; variances are represented on the log scale. This avoids the non-identified Hessian directions produced by differentiating all class-specific covariance elements under equality or zero constraints.

The analytic "Louis" method evaluates the observed information from the conditional complete-data score and Hessian. For both LCA and LPA it accounts for posterior class uncertainty and therefore is not the naive complete-data information matrix. The numerical "Obs" method provides an independent check by differentiating the analytic observed-data score.

Value

A list of class "SE" containing:

se SEs with the same parameter structure and dimnames as the fitted model.

vcov Covariance matrix on the unconstrained estimation scale for "Obs" and "Louis"; NULL for bootstrap.

`hessian` Observed information matrix on the unconstrained estimation scale for `"Obs"` and `"Louis"`; `NULL` for bootstrap.

`diagnostics` Method-specific diagnostics.

`call` Function call that generated the object.

`arguments` Input arguments.

References

Louis, T. A. (1982). Finding the observed information matrix when using the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological), 44*(2), 226–233. doi:10.1111/j.25176161.1982.tb01203.x

McLachlan, G. J., & Peel, D. (2000). *Finite mixture models*. John Wiley & Sons.

Examples

```
set.seed(123)
lca.data <- sim.LCA(N = 300, I = 4, L = 2, poly.value = 3)
lca.fit <- LCA(lca.data$response, L = 2, nrep = 3, vis = FALSE)
se.louis <- get.SE(lca.fit, method = "Louis")

lpa.data <- sim.LPA(N = 300, I = 3, L = 2, constraint = "V0")
lpa.fit <- LPA(lpa.data$response, L = 2, constraint = "V0", nrep = 3, vis = FALSE)
se.louis.lpa <- get.SE(lpa.fit, method = "Louis")
```

---

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

# Run dependency installer
deps <- install_python_dependencies()

# Check which were missing
print(deps)
```

`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, starts = 1)
```

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 $2 \leq L < N$.
`starts`	Integer specifying the number of random starts for K-means algorithm (default: 1). 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 $P(X_i = q \mid Z = l)$ for all indicators i and response categories q.
- Posterior probabilities: Constructs hard-classification matrix where $P(Z_n = l \mid \mathbf{X}_n) = 1$ 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

This function is primarily designed as an initialization method for `LCA()` and not 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, starts = 5)

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

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

Description

Performs hard clustering with K-means and converts the resulting partition into initial means, covariance matrices, profile proportions, and hard posterior assignments for Latent Profile Analysis (LPA).

Usage

```
Kmeans.LPA(response, L, constraint = "VV", starts = 1)
```

Arguments

`response`	A numeric matrix or data frame of dimension $N \times I$ containing continuous indicators. Missing and non-finite values are not allowed. The indicators must be standardized before input using `scale` or `normalize`; no additional scaling is performed internally.
`L`	Positive integer specifying the number of latent profiles. It must be smaller than the number of observations.
`constraint`	Covariance structure used to construct the initial profile covariance matrices. Named options are `"UE"`, `"UV"`, `"E0"`, `"V0"`, `"EE"`, `"VV"`, `"VE"`, and `"EV"`. A custom list of index pairs may be supplied to constrain selected variance or covariance elements equal across profiles, using the same semantics as `LPA`.
`starts`	Positive integer specifying the number of internal K-means random starts (default: 1). The solution with the lowest within-cluster sum of squares is retained.

Details

The function performs four operations:

- Runs K-means on the supplied standardized indicators using Lloyd's algorithm.
- Uses the cluster centers as initial profile means and cluster proportions as initial profile probabilities.
- Computes within-cluster maximum-likelihood covariance matrices, then applies the requested named or custom equality constraints.

Value

A list containing:

`params` A list with:

`means` An $L \times I$ matrix of initial profile means.

`covs` An $I \times I \times L$ array of constrained, positive-definite initial covariance matrices.

`P.Z` A numeric vector of length L containing initial profile proportions.

`P.Z.Xn` An $N \times L$ hard-assignment matrix whose rows contain one 1 and $L - 1$ zeros.

Note

This function constructs initialization values only; it does not fit a final LPA model. When called directly with `starts > 1`, K-means selects one solution from those internal starts. The package-level `LPA(..., par.ini = "kmeans")` workflow instead calls `Kmeans.LPA(..., starts = 1)` separately for every outer start.

Examples

```
set.seed(123)
response <- scale(matrix(rnorm(300), ncol = 3))
initial <- Kmeans.LPA(response, L = 2, constraint = "V0")
initial$params$means
initial$params$P.Z
```

LCA *Fit Latent Class Analysis Models*

Description

This function estimates parameters of a Latent Class Analysis (LCA; Hagenaars & McCutcheon, 2002) model using the Expectation-Maximization (EM) algorithm, stochastic EM (SEM) through `flexmix` or `RMixtComp`, native EM/CEM/SEM strategies through `Rmixmod`, Neural Network Estimation (NNE), or Mplus. It supports flexible initialization strategies and provides comprehensive model diagnostics.

Usage

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

Arguments

`response`	A numeric matrix of dimension $N \times I$, where N is the number of participants and I is the number of observed categorical indicators. Each column must contain nominal-scale discrete responses (e.g., integers representing categories).
`L`	Integer specifying the number of latent classes (default: 2).
`category.levels`	Optional list fixing the ordered response categories for each indicator. If `NULL`, the mapping is determined once from the supplied full data. LCA estimation requires every mapped category to occur in the supplied data.
`method`	Character string specifying estimation algorithm: • `"EM"`: Expectation-Maximization algorithm (default). • `"NNE"`: Neural Network Estimation (experimental), using feed-forward layers, optional transformer attention, gradient optimization, and simulated annealing. See `install_python_dependencies`. • `"Mplus"`: Calls external Mplus software for estimation. Uses Mplus defaults for optimization unless overridden by `control.Mplus`. • `"flexmix"`: Stochastic EM (SEM) through `flexmix`, using LCPA's warm-up and promoted-replication scheme. Requires `flexmix`. • `"Rmixmod"`: LCPA warm-up plus SEM, or a native Rmixmod EM, CEM, or SEM strategy. Requires the `Rmixmod` package. • `"RMixtComp"`: Stochastic EM (SEM) estimation through `RMixtComp`. Requires `RMixtComp` and `RMixtCompUtilities`. No non-SEM RMixtComp algorithm is exposed.
`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, 2000).
- 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.

For methods that expose this initialization interface, par.ini is used only to construct the starts warm-up initializations. It does not initialize the nrep refinement runs, which continue directly from the selected warm-up states. With par.ini = "kmeans", each outer warm-up start performs exactly one K-means run. Thus, starts is the number of separately initialized K-means outputs passed to warm-up training. Backends such as Mplus and Rmixmod may instead use their native initialization mechanisms and are not required to implement K-means initialization. If "kmeans" is requested for a method that does not support it, par.ini is automatically changed to "random".

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.

starts — Positive integer. Number of warm-up analyses to run (default: 100). Each analysis is initialized by the selected method and trained for at most maxiter.warmup iterations, producing exactly starts warm-up solutions. For methods supporting par.ini = "kmeans", each outer warm-up start performs exactly one K-means run, so starts is also the number of K-means outputs.

maxiter.warmup — Positive integer. Maximum number of training iterations for each of the starts warm-up analyses (default: 20). This limit applies only to warm-up and does not limit the subsequent refinement phase.

nrep — Positive integer not exceeding starts. Number of refinement analyses (default: 20). The nrep warm-up solutions with the largest log-likelihoods are continued from their saved states until the full-training stopping rule is met; no new initialization occurs in this phase. The refined solution with the largest log-likelihood is returned as the final result. These three staged-training arguments are not used by method = "RMixtComp" or by method = "Rmixmod" with control.Rmixmod$path = "Rmixmod"; those paths use their documented native controls instead.

vis — Logical. If TRUE, displays carriage-return-updated Warm and Rep lines when the backend exposes those stages, followed by one final fit-summary line (default: TRUE). Each stage occupies one console line and ends with one newline.

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 under which Mplus writes intermediate files, including model input, data, output, and saved posterior probabilities. The effective default is "". A non-empty path is created recursively when necessary and must be writable. Within it, the function creates a unique timestamped subdirectory named "Mplus_LCA_YYYY-MM-DD_HH-MM-SS" to isolate all files from the current run. If files.path = "", that timestamped subdirectory is created directly under R's current working directory, getwd(). Explicit NULL is invalid.

files.clean Logical. If TRUE (default), all intermediate files and the temporary working directory created for the run are deleted on successful completion or error exit via on.exit(). If FALSE, the complete timestamped working directory is retained under files.path, or under getwd() when files.path = "", for inspection and debugging.

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 Positive offset used in the NNE objective (default: 1e-8).

lambda Coefficient of the NNE parameter penalty (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).

patience.early Maximum consecutive iterations without improvement before early stopping (default: 100).

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: 0.80).

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.

`control.flexmix`

List of control parameters for flexmix SEM estimation:

`maxiter` Number of SEM iterations in every promoted full run (default: 1000).

`minprior` Minimum component prior accepted by flexmix (default: 0, so LCPA does not intentionally remove requested classes).

`tol` Relative likelihood-change threshold used by flexmix (default: 0, which enforces the fixed SEM iteration count).

`control.Rmixmod`

List of control parameters for Rmixmod estimation:

`path` Execution path: `"LCPA"` (default) preserves the package's `starts`/`maxiter.warmup`/`nrep` warm-up plus pure SEM procedure; `"Rmixmod"` delegates one native EM, CEM, SEM, or combined strategy and ignores those three arguments.

`algorithm`, `nrep`, `method.init`, `starts`, `maxiter.init`, `maxiter`, `tol.init`, `tol`, `par.ini`, `labels.i` Package-standard controls translated to the corresponding arguments of `Rmixmod::mixmodStrategy()` when `path="Rmixmod"`. Unspecified arguments retain the installed Rmixmod version's defaults. Under `path="LCPA"`, only `maxiter` is used, with 1000 iterations when omitted. `algorithm` accepts `"EM"`, `"CEM"`, and `"SEM"`, including ordered combinations supported by Rmixmod.

`strategy` Optional pre-built Rmixmod `Strategy` object for `path="Rmixmod"`. When supplied, it takes precedence over the individual strategy controls.

`control.RMixtComp`

List of control parameters for RMixtComp SEM estimation:

`maxiter.burnin` Number of native SEM burn-in iterations (default: 50).

`maxiter` Number of recorded post-burn-in SEM iterations (default: 50).

`maxiter.gibbs.burnin`, `maxiter.gibbs` Numbers of burn-in and recorded iterations in RMixtComp's subsequent fixed-parameter Gibbs stage (defaults: 50 and 50).

`n.init.per.class` Number of observations per class used by RMixtComp's native parameter initialization (default: 50).

`maxattempts.sem` Maximum number of SEM attempts (default: 20).

`confidence.level`, `stable.ratio`, `n.stable` Native RMixtComp SEM controls (defaults: 0.95, 0.99, and 20).

`criterion` RMixtComp model-selection criterion: `"BIC"` (default) or `"ICL"`.

`nrep` Number of native RMixtComp SEM runs for the requested number of classes; RMixtComp retains the run with the largest observed likelihood (default: 1).

`ncores` Number of cores used by RMixtComp to parallelize `nrep`; must not exceed `nrep` (default: 1).

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.

`category.levels` Fixed ordered response categories for each indicator.

`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 when applicable; for native Rmixmod and RMixtComp paths, the selected native fit's BIC.

`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` List of item-specific conditional probability matrices with classes in rows and observed category labels in columns.

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

`Log.Lik.nrep` Vector of log-likelihoods from each replication run. For native Rmixmod and RMixtComp paths, this is the selected native fit's scalar log-likelihood.

`model` Backend model object for `method="NNE"`, `method="Mplus"`, `method="flexmix"`, `method="Rmixmod"`, or `method="RMixtComp"`, when supplied by that backend.

`call` Matched function call.

`arguments` A list containing all effective input arguments.

Random-number reproducibility

Except for `method = "NNE"`, which intentionally uses its fixed backend seed, every stochastic estimator is driven from R's current random-number generator. The user only needs to call `set.seed()` immediately before `LCA()` to reproduce EM, K-means, flexmix, Rmixmod, RMixtComp, and Mplus estimation. LCPA automatically passes an R-derived seed to backends with independent random streams; no backend-specific seed setting is required.

Notation

Write the response matrix as $\mathbf{X} = (X_{ni})_{N\times I}$, where $n = 1, 2, \ldots, N$ indexes participants and $i = 1, 2, \ldots, I$ indexes observed indicators. The response vector for participant n is $\mathbf{X}_n = (X_{n1}, \ldots, X_{nI})^\top$. The latent class variable is $Z_n \in \{1, 2, \ldots, L\}$, and $l = 1, 2, \ldots, L$ indexes a particular latent class. Response categories for indicator i are indexed by q.

Under local independence, the observed-data log-likelihood is

$$\log \mathcal{L}_{\mathrm{LCA}} = \sum_{n=1}^{N} \log \left\{ \sum_{l=1}^{L} \pi_l \prod_{i=1}^{I} P(X_{ni} = x_{ni} \mid Z_n = l) \right\}.$$

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:

$$\tau_{nl} = P(Z_n = l \mid \mathbf{X}_n) = \frac{\pi_l \prod_{i=1}^{I} P(X_{ni} = x_{ni} \mid Z_n = l)}{\sum_{h=1}^{L} \pi_h \prod_{i=1}^{I} P(X_{ni} = x_{ni} \mid Z_n = h)}.$$

  where x_{ni} is the standardized (0-based) response of participant n to indicator i (see `adjust.response`).
- M-step: Update parameters by maximizing expected complete-data log-likelihood:
  - Class probabilities: $\pi_l^{\text{new}} = \frac{1}{N} \sum_{n=1}^{N} \tau_{nl}$
  - Conditional probabilities: $P(X_i = q \mid Z = l)^{\text{new}} = \frac{\sum_{n:x_{ni}=q} \tau_{nl}}{\sum_{n=1}^{N} \tau_{nl}}$
- 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.

Profile Parameter Estimation Global conditional probability parameters ($P(X_i = q \mid Z = l)$) 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

Mplus

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

Workflow:

Working Directory Setup Creates a timestamped `"Mplus_LCA_YYYY-MM-DD_HH-MM-SS"` directory under `control.Mplus$files.path`, or under the current working directory when that path is empty, 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$files.clean = 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
  `STSEED` Random-start seed drawn from R's current random-number generator
  `STITERATIONS = maxiter.warmup` 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)

flexmix Stochastic EM

With `method = "flexmix"`, each SEM iteration performs one stochastic classification draw from the current posterior probabilities before the M-step. LCPA runs exactly `starts` short trajectories of `maxiter.warmup` iterations, promotes the best `nrep` trajectories by observed log-likelihood, continues each for `control.flexmix$maxiter` SEM iterations, and retains the largest-likelihood final state across the promoted runs. This selection compares the final state returned by each flexmix SEM run; flexmix's `classify = "SEM"` does not retain the largest-likelihood state visited within a run. Binary indicators use `flexmix::FLXMCmvbinary()`. When any indicator is polytomous, LCPA supplies one joint categorical flexmix model driver whose M-step uses the exact weighted-frequency solution instead of repeatedly optimizing intercept-only multinomial regressions. The stochastic classification and iteration loop remain those of flexmix. Setting `control.flexmix$tol = 0` prevents likelihood-based early termination. Only SEM is exposed; `par.ini` is not used by this backend.

Rmixmod stochastic strategies

When `method = "Rmixmod"`, the model is estimated in `Rmixmod::mixmodCluster()` using the unconstrained multinomial model `"Binary_pk_Ekjh"`. Indicators are converted to factors after the package-standard zero-based recoding.

With `path="LCPA"`, the function generates exactly `starts` random balanced partitions and runs exactly `maxiter.warmup` consecutive stochastic E-S-M iterations for every warm-up start. The best `nrep` warm-up classifications are then passed directly to independent Rmixmod SEM runs, and the finite solution with the largest final log-likelihood is retained. Rmixmod's internal `"smallEM"` and `"SEMMax"` initialization searches are not used; in particular, `"SEMMax"` is an initialization search rather than one consecutive SEM trajectory for each user-level start. `par.ini` is not used for this

method. SEM stops after `control.Rmixmod$maxiter` iterations; an epsilon convergence criterion is not defined for SEM in Rmixmod. Unlike flexmix's `classify = "SEM"`, Rmixmod retains the largest-likelihood parameter state visited within each SEM run, so equal iteration counts do not imply identical final-state selection.

With `path="Rmixmod"`, LCPA translates all non-`NULL` package-standard strategy controls to `Rmixmod::mixmodStrategy()` and runs one native `mixmodCluster()` call. Its `algorithm` may contain `"EM"`, `"CEM"`, `"SEM"`, or an ordered combination of these algorithms. LCPA does not add its outer `starts`, `maxiter.warmup`, or `nrep`; `control.Rmixmod$nrep` controls complete strategy repetitions. The published strategy of Mulder et al. (2015) uses 200 SEM iterations followed by EM with a relative likelihood-change tolerance of `1e-5`. For reproducibility across Rmixmod versions, the example and simulation scripts explicitly pin the contemporaneous documented defaults: `smallEM`, 50 initialization tries, 5 initialization iterations, `tol.init=0.001`, and a 200-iteration EM limit.

RMixtComp Stochastic EM

When `method = "RMixtComp"`, LCPA calls `RMixtComp::mixtCompLearn()` in classic, non-hierarchical learning mode with a multinomial model for every indicator. This integration exposes only RMixtComp's stochastic EM (SEM) algorithm; it does not introduce any other RMixtComp estimation algorithm. LCPA does not add its own `starts`, `maxiter.warmup`, or `nrep` stages to this backend. Each native run performs RMixtComp initialization, SEM burn-in, recorded SEM iterations, and then the fixed-parameter Gibbs burn-in and recorded Gibbs iterations. One stochastic S-step is performed per SEM iteration, between the E-step and M-step. Native repetition and parallelization are controlled only by `control.RMixtComp$nrep` and `control.RMixtComp$ncores`. RMixtComp's native random stream is independent of R; LCPA therefore passes it one integer drawn from R's current random stream. The user only needs an external `set.seed()` for reproducibility. The default algorithm controls reproduce `RMixtCompUtilities::createAlgo()` defaults (version 4.1.4 or later). `par.ini` is not used by this backend.

References

Biernacki, C. (2015). MixtComp software: Model-based clustering/imputation with mixed data, missing data and uncertain data. *MISSDATA 2015*. https://inria.hal.science/hal-01253393

Hagenaars, J. A., & McCutcheon, A. L. (Eds.). (2002). *Applied latent class analysis*. Cambridge University Press.

Leisch, F. (2004). FlexMix: A general framework for finite mixture models and latent class regression in R. *Journal of Statistical Software, 11*(8), 1–18. doi:10.18637/jss.v011.i08

McLachlan, G. J., & Peel, D. (2000). *Finite mixture models*. John Wiley & Sons.

Mulder, V. L., Lacoste, M., Martin, M. P., Richer-de-Forges, A., & Arrouays, D. (2015). Understanding large-extent controls of soil organic carbon storage in relation to soil depth and soil-landscape systems. *Global Biogeochemical Cycles, 29*(8), 1210–1229. doi:10.1002/2015GB005178

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
# An empty 'files.path' creates a timestamped subfolder
# (e.g., "Mplus_LCA_YYYY-MM-DD_HH-MM-SS") under the current working directory
# to store all temporary Mplus files (.inp, .dat, .out, etc.).
## Not run:
fit.mplus <- LCA(response, L = 2, method = "Mplus", nrep = 3)

## 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)

# Fit 2-class model with flexmix SEM and LCPA warm-up/replication
# need flexmix
## Not run:
  fit.flexmix <- LCA(response, L = 2, method = "flexmix",
                     nrep = 2, starts = 5, maxiter.warmup = 5,
                     control.flexmix = list(maxiter = 50))

## End(Not run)

# Fit 2-class model with the published Rmixmod SEM-to-EM strategy
# need Rmixmod
## Not run:
  fit.rmixmod <- LCA(response, L = 2, method = "Rmixmod",
                     control.Rmixmod = list(path = "Rmixmod",
                                            algorithm = c("SEM", "EM"),
                                            nrep = 1,
                                            method.init = "smallEM",
                                            starts = 50,
                                            maxiter.init = 5,
                                            tol.init = 0.001,
                                            maxiter = c(200, 200),
                                            tol = c(NA, 1e-5)))

## End(Not run)
```

LCPA *Three-step latent class/profile analysis*

Description

Fits one of two independent auxiliary-variable paths. `type.analysis = "XZ"` estimates the effect of observed covariates on latent class/profile membership. `type.analysis = "ZY"` estimates class/profile-specific distributions of external observed dependent variables; these variables are never treated as indicators.

Usage

```
LCPA(
  response,
  L = 2,
  type.analysis = c("XZ", "ZY"),
  type.model = c("LCA", "LPA"),
  covariates = NULL,
  ref.class = L,
  dependent.variables = NULL,
  family = "gaussian",
  method.model = "EM",
  control.model = NULL,
  control.EM = NULL,
  control.Mplus = NULL,
  control.NNE = NULL,
  control.flexmix = NULL,
  control.Rmixmod = NULL,
  control.RMixtComp = NULL,
  method.3step = NULL,
  CEP.error = TRUE,
  method.regression = "Analytic",
  maxiter = 5000,
  tol = 1e-04,
  lower = -10,
  upper = 10,
  method.SE = "Bootstrap",
  nrep.bootstrap = 100,
  vis = TRUE
)
```

Arguments

`response`	An $N \times I$ numeric matrix or data frame containing the Step 1 latent class/profile indicators. For `type.model = "LCA"`, each column is a categorical indicator and follows the same category-mapping requirements as the `response` argument of `LCA()`. For `type.model = "LPA"`, all columns are continuous, missing values

are not allowed, and the indicators should be standardized with `scale()` or `normalize()` before analysis, exactly as required by `LPA()`.

`L` Integer number of latent classes/profiles in the Step 1 measurement model (default: 2; must be at least 2). It has the same meaning as `L` in `LCA()` or `LPA()`.

`type.analysis` Character string selecting the independent Step 3 path:

- `"XZ"`: estimate the effect of observed covariates $\boldsymbol{\zeta}$ on latent class/profile membership Z; `covariates` is used and `dependent.variables` is ignored.
- `"ZY"`: estimate class/profile-specific distributions of external observed dependent variables $\mathbf{Y}$; `dependent.variables` and `family` are used and `covariates` and `ref.class` are ignored.

The default is `"XZ"`.

`type.model` Character string selecting the Step 1 measurement model: `"LCA"` for categorical indicators or `"LPA"` for continuous indicators (default: `"LCA"`). Its meaning, response-data requirements, and fitted measurement parameters are the same as in `LCA()` and `LPA()`, respectively.

`covariates` Used only when `type.analysis = "XZ"`. An $N \times (U + 1)$ numeric matrix/data frame representing $\boldsymbol{\zeta}_n = (1, \zeta_{n1}, \ldots, \zeta_{nU})^\top$. Its first column is an unstandardized all-ones intercept; standardize the U observed covariates before analysis and construct interactions from the standardized variables. If `NULL` (default), the function automatically creates an $N \times 1$ all-ones design, so an intercept-only class-membership model remains estimable. The argument is ignored for `"ZY"`; that path automatically uses latent-class-specific intercepts as the dependent-variable predictors.

`ref.class` Integer from 1 to `L` selecting the reference category in the `X -> Z` multinomial regression (default: `L`). Its coefficient vector is fixed to zero. If `control.model$is.sort = TRUE`, the value refers to the class/profile position after Step 1 sorting. This argument is not used for `"ZY"` because that path estimates a separate dependent-variable distribution for every class.

`dependent.variables` Used only when `type.analysis = "ZY"`. An observed dependent-variable vector, or an $N \times V$ matrix/data frame representing $\mathbf{Y}_n = (Y_{n1}, \ldots, Y_{nV})^\top$, with one dependent variable per column and one participant per row. These are external dependent variables (for example, depression or anxiety), not Step 1 latent class/profile indicators. Missing dependent-variable values are allowed and are omitted separately for each dependent variable; the indicator data in `response` remain subject to the requirements of `LCA()` or `LPA()`. Before calling `LCPA()`, standardize every continuous dependent-variable column assigned `family = "gaussian"` over the N participants, preferably with `scale()`, so that its observed sample mean is 0 and sample standard deviation is 1. Do not standardize columns assigned `family = "categorical"`; retain their original category values.

`family` Used only when `type.analysis = "ZY"`. Either one character string applied to every dependent variable or a character vector with one value per dependent-variable column. `"gaussian"` (default) estimates class-specific means for a standardized numeric continuous dependent variable; `"categorical"` estimates class-specific probabilities for the observed categories. Missing values are excluded separately for each dependent variable.

`method.model` Character string selecting the Step 1 parameter estimator. It has exactly the same meaning and available values as the `method` argument of `LCA()` or `LPA()`: `"EM"` (default), `"NNE"`, `"Mplus"`, `"flexmix"`, `"Rmixmod"`, or `"RMixtComp"`. The corresponding `control.*` argument is passed to the selected Step 1 function. It is not used to estimate the Step 3 regression/dependent-variable model.

`control.model` Optional named list of common Step 1 measurement-model settings. If `NULL` (default), all settings below use their defaults. Supply only the elements to override:

- `params` Optional fixed Step 1 parameter list (default: `NULL`). When supplied, Step 1 fitting is skipped. For LCA it must contain `par`, `P.Z`, and `category.levels`; for LPA it must contain `means`, `covs`, and `P.Z`, with the definitions and dimensions returned by `LCA()` and `LPA()`.
- `par.ini` Initialization used when `params = NULL` (default: `"random"`). It accepts `"random"`, `"kmeans"`, or the model-specific parameter-list forms documented for `LCA()` and `LPA()`.
- `constraint` LPA covariance structure (default: `"VV"`). It accepts `"UE"`, `"UV"`, `"E0"`, `"V0"`, `"EE"`, `"VV"`, `"VE"`, `"EV"`, or a custom equality list as documented for `LPA()`. It is ignored when `type.model = "LCA"`.
- `is.sort` Logical (default: `TRUE`). Order classes/profiles by decreasing Step 1 prior probability and consistently permute Step 1, CEP, and Step 3 class-specific results.
- `starts` Positive integer number of independently initialized Step 1 warm-up analyses (default: 100).
- `maxiter.warmup` Positive integer maximum number of Step 1 iterations per warm-up analysis (default: 20).
- `nrep` Positive integer not exceeding `starts` (default: 20). The best warm-up states are continued to final Step 1 fits.

Element names must be unique; unknown or unnamed elements are rejected.

`control.EM` Optional Step 1 control list for the EM estimator in `LCA()` or `LPA()`. `maxiter` sets the maximum number of EM iterations, `tol` sets the convergence tolerance, and the LPA covariance-floor element bounds small covariance eigenvalues away from zero. See `control.EM` in those functions for the complete list.

`control.Mplus` Optional list passed to the selected `LCA()`/`LPA()` Mplus backend. It has the same `maxiter`, `tol`, `files.path`, `files.clean`, and LPA covariance-floor semantics documented there.

`control.NNE` Optional list passed to the selected `LCA()`/`LPA()` NNE backend. Network architecture, attention, annealing, optimizer, plotting, and device fields have exactly the meanings documented for `control.NNE` in those functions.

`control.flexmix` Optional list passed to the selected `LCA()`/`LPA()` flexmix SEM backend. `maxiter`, `minprior`, `tol`, and the LPA covariance floor have the same meanings as in `LCA()`/`LPA()`.

`control.Rmixmod` Optional list passed to the selected `LCA()`/`LPA()` Rmixmod backend. The `path`, native strategy, algorithm, initialization, iteration, tolerance, and replication fields retain the meanings and restrictions documented in those functions.

`control.RMixtComp`
: Optional list passed to the selected `LCA()`/`LPA()` RMixtComp SEM backend. Burn-in, SEM/Gibbs iterations, stability, initialization, criterion, replication, and core controls have the same meanings and restrictions documented in those functions.

`method.3step`
: Character string selecting the Step 2–3 correction:

  - `"ML"`: maximize a likelihood that treats the modal Step 1 class as measured with error through the CEP matrix.
  - `"BCH"`: invert the CEP matrix and estimate Step 3 with BCH pseudo-weights, leaving the Step 1 class/profile definition fixed.

  If `NULL` (default), `"ML"` is selected for `"XZ"` and `"BCH"` for `"ZY"`. Both methods are available for both paths. ML/CEP estimates a classification-error-corrected likelihood, whereas BCH estimates inverse-CEP-weighted score equations. BCH is generally preferred for distal dependent variables, and ML is generally preferred for class-membership regression. Vermunt (2010) develops both BCH and ML corrections for covariates predicting class membership, including the BCH-XZ specification. Bakk, Tekle, and Vermunt (2013) develop the bias-adjusted ML formulation for class membership predicting a distal dependent variable; Nylund-Gibson, Grimm, and Masyn (2019) provide a worked manual ML three-step distal-outcome analysis.

`CEP.error`
: Logical. If `TRUE` (default and recommended), Step 2 estimates the classification-error probability matrix $\mathrm{CEP}(l, k) = P(\widehat{Z} = k \mid Z = l)$ using `get.CEP()`, and Step 3 applies the selected correction. If `FALSE`, an identity CEP matrix is used, reducing the analysis to naive modal-class assignment without classification-error correction.

`method.regression`
: Character string controlling Step 3 point estimation (default: `"Analytic"`). For ML-XZ, BCH-XZ, and ML-ZY, `"Analytic"` supplies the exact analytic gradient to a numerical optimizer; these models do not generally have closed-form coefficient estimates. For BCH-ZY, `"Analytic"` directly evaluates the weighted closed-form class means or categorical probabilities. `"Numeric"` optimizes the same likelihood without the supplied gradient or numerically solves the same BCH estimating equations. This choice is independent of `method.SE`.

`maxiter`
: Positive integer giving the maximum number of Step 3 `"XZ"` optimization iterations (default: 5000). It is distinct from Step 1 `control.*$maxiter` and `control.model$maxiter.warmup`, and is ignored for `"ZY"`.

`tol`
: Positive finite convergence tolerance for the Step 3 `"XZ"` optimization (default: 10^{-4}). It is separate from the Step 1 tolerance in `control.EM`, `control.Mplus`, or another backend control and is ignored for `"ZY"`, whose dependent-variable estimators use their own equations.

`lower, upper`
: Finite lower and upper bounds applied to every `"XZ"` multinomial regression coefficient (defaults: -10 and 10). They are ignored for `"ZY"`. Inspect the returned bound diagnostics when an estimate reaches a limit.

`method.SE`
: Character string selecting Step 3 uncertainty estimation:

  - `"Analytic"`: for ML-XZ, use the analytic observed-information bread and an empirical sandwich meat that includes the influence of estimating the

CEP matrix; for ML-ZY, use Louis observed information; for BCH, use the analytic estimating-equation bread and empirical sandwich meat.

- `"Numeric"`: replace the analytic information/bread with a numerical Hessian or Jacobian while retaining the corresponding empirical meat and Step 3 estimand.
- `"Bootstrap"` (default): resample individuals, keep the Step 1 measurement parameters fixed, recompute posterior assignments and CEP or BCH weights, and re-estimate Step 3.

`nrep.bootstrap` Integer number of nonparametric bootstrap replications used only when `method.SE = "Bootstrap"` (default: 100; minimum: 2). Only successful Step 3 replications contribute to the empirical covariance; larger values such as 500–1000 are advisable for final publication-level inference when computationally feasible.

`vis` Logical (default: `TRUE`). If enabled, display the Step 1 measurement-model progress, Step 2 posterior/CEP preparation, and Step 3 regression or distal-dependent-variable estimation progress. Each Step 2 and Step 3 heading identifies the selected `X -> Z` or `Z -> Y` path. For `Z -> Y`, the output also gives the number and family of dependent-variable models, convergence and iteration information, and the selected standard-error or bootstrap progress.

Details

The notation distinguishes the Step 1 indicators from the Step 3 auxiliary variables. Write the indicator matrix as $\mathbf{X} = (X_{ni})_{N\times I}$, where $n = 1, 2, \ldots, N$ indexes participants and $i = 1, 2, \ldots, I$ indexes observed indicators. Participant n's indicator vector is $\mathbf{X}_n = (X_{n1}, \ldots, X_{nI})^\top$, and $Z_n \in \{1, 2, \ldots, L\}$, with $l = 1, 2, \ldots, L$ indexing latent classes/profiles.

The covariate vector is $\boldsymbol{\zeta}_n = (1, \zeta_{n1}, \ldots, \zeta_{nU})^\top$, where $u = 1, 2, \ldots, U$ indexes the U observed covariates and the leading 1 is the intercept. The dependent-variable vector is $\mathbf{Y}_n = (Y_{n1}, \ldots, Y_{nV})^\top$, where $v = 1, 2, \ldots, V$ indexes the V external observed dependent variables. Neither $\boldsymbol{\zeta}_n$ nor $\mathbf{Y}_n$ is part of the indicator vector $\mathbf{X}_n$. `"XZ"` is the function-interface label for the covariate-to-latent path; the formulas use $\boldsymbol{\zeta}$ for its covariates because $\mathbf{X}$ is reserved for the LCA/LPA indicator data. `type.analysis = "XZ"` uses $\boldsymbol{\zeta}_n$ but not $\mathbf{Y}_n$; `type.analysis = "ZY"` uses $\mathbf{Y}_n$ but not $\boldsymbol{\zeta}_n$. These are separate Step 3 analyses rather than a jointly estimated mediation model. Run both analyses when both the covariate-to-class and class-to-dependent-variable associations are required. For a ZY analysis, standardization applies to the continuous dependent variables in $\mathbf{Y}_n$, not to the Step 1 indicator matrix $\mathbf{X}$. Each Gaussian dependent variable must be transformed before model fitting to have observed sample mean 0 and sample standard deviation 1. Consequently, its class/profile-specific estimates and standard errors are expressed in observed-standard-deviation units. Categorical dependent variables retain their original category values.

Value

An object of class `"LCPA"`. The selected result is available from `analysis$XZ` or `analysis$ZY`; posterior probabilities, modal assignments, and CEP matrices use the same list structure as `LTA()`. For `type.analysis = "ZY"`, `dependent.variables$t1` contains one fitted model per observed dependent variable. A Gaussian model reports class/profile-specific `estimate`, `se`, `variance`, `variance.se`, their covariance matrices, omnibus Wald tests for both means and variances, group weight masses, observations, omitted values, iterations, and convergence. A categorical model reports class/profile-by-category `estimate` and `se` matrices, their covariance matrix, an omnibus

Wald test of equality of the conditional category distributions, group weight masses, observations, omitted values, iterations, and convergence.

Methodology overview

The cross-sectional three-step analysis proceeds as follows.

Step 1 – Unconditional measurement model. Fit an unconditional `LCA()` or `LPA()` to `response`. Let $\pi_l = P(Z_n = l)$. For LCA, the Step 1 observed-data log-likelihood is

$$\log \mathcal{L}_{\mathrm{LCA}} = \sum_{n=1}^{N} \log \left\{ \sum_{l=1}^{L} \pi_l \prod_{i=1}^{I} P(X_{ni} = x_{ni} \mid Z_n = l) \right\}.$$

For LPA, it is

$$\log \mathcal{L}_{\mathrm{LPA}} = \sum_{n=1}^{N} \log \left\{ \sum_{l=1}^{L} \pi_l \mathcal{N}(\mathbf{X}_n \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l) \right\}.$$

These are the likelihoods defined in `get.Log.Lik.LCA()` and `get.Log.Lik.LPA()`. Bayes' theorem gives

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

for LCA and

$$\tau_{nl} = P(Z_n = l \mid \mathbf{X}_n) = \frac{\pi_l \mathcal{N}(\mathbf{X}_n \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}{\sum_{h=1}^{L} \pi_h \mathcal{N}(\mathbf{X}_n \mid \boldsymbol{\mu}_h, \boldsymbol{\Sigma}_h)}$$

for LPA. The modal assignment is $\widehat{Z}_n = \arg\max_l \tau_{nl}$. `method.model` selects the estimator of this measurement model, and `control.model` supplies its initialization, covariance-constraint, sorting, and replication settings. If `control.model$params` is supplied, those fixed measurement parameters are used to calculate τ_{nl}.

Step 2 – Classification-error probabilities. The $L \times L$ CEP matrix has rows indexed by latent class/profile l and columns indexed by modal assignment k, so $\mathrm{CEP}(l, k) = P(\widehat{Z}_n = k \mid Z_n = l)$. The modal assignment, posterior-weight estimator, matrix orientation, and pooling rules are defined in `get.CEP()`. With `CEP.error = FALSE`, CEP is replaced by the identity matrix and Step 3 becomes an uncorrected modal-assignment analysis. The BCH and ML corrections for this classification error follow Bolck, Croon, and Hagenaars (2004) and Vermunt (2010).

Step 3A – Covariates predicting latent membership (XZ). With reference class `ref.class` denoted by l_0, the multinomial-logit model is

$$P(Z_n = l \mid \boldsymbol{\zeta}_n) = \frac{\exp(\boldsymbol{\zeta}_n^{\top} \boldsymbol{\beta}_l)}{1 + \sum_{h \neq l_0} \exp(\boldsymbol{\zeta}_n^{\top} \boldsymbol{\beta}_h)}, \quad l \neq l_0,$$

with $\boldsymbol{\beta}_{l_0} = 0$. If `covariates = NULL`, $\boldsymbol{\zeta}_n = 1$ and the model contains class-specific intercepts only. Vermunt's (2010) ML/CEP estimator maximizes

$$\ell_{\mathrm{ML}}(\boldsymbol{\beta}) = \sum_{n=1}^{N} \log \left\{ \sum_{l=1}^{L} \mathrm{CEP}(l, \widehat{Z}_n) P(Z_n = l \mid \boldsymbol{\zeta}_n) \right\}.$$

For each non-reference class/profile $l \neq l_0$, the first derivative of the observed-data log-likelihood with respect to the coefficient vector $\boldsymbol{\beta}_l$ is

$$\frac{\partial \ell_{\mathrm{ML}}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}_l} = \sum_{n=1}^{N} \boldsymbol{\zeta}_n P(Z_n = l \mid \boldsymbol{\zeta}_n) \left\{ \frac{\mathrm{CEP}(l, \widehat{Z}_n)}{\sum_{h=1}^{L} \mathrm{CEP}(h, \widehat{Z}_n) P(Z_n = h \mid \boldsymbol{\zeta}_n)} - 1 \right\}.$$

This derivative is an $(U+1) \times 1$ vector: its entries correspond to the intercept and the U covariate coefficients in $\boldsymbol{\beta}_l$. At an interior maximum, the ML estimates jointly satisfy $\partial \ell_{\mathrm{ML}} / \partial \boldsymbol{\beta}_l = \mathbf{0}$ for every $l \neq l_0$; `lower` and `upper` define the permitted coefficient range.

Vermunt's (2010) BCH-XZ estimator instead solves, for class $l \neq l_0$,

$$\sum_{n=1}^{N} \boldsymbol{\zeta}_n \left\{ (\mathrm{CEP}^{-1})_{\widehat{Z}_n, l} - P(Z_n = l \mid \boldsymbol{\zeta}_n) \sum_{h=1}^{L} (\mathrm{CEP}^{-1})_{\widehat{Z}_n, h} \right\} = \mathbf{0}.$$

This is a BCH estimating equation, not the derivative of the ML corrected likelihood above. Its $U+1$ equations correspond to the intercept and covariate coefficients in $\boldsymbol{\beta}_l$. Thus ML and BCH estimate the same multinomial-logit parameters but use different corrections for modal-classification error.

Step 3B – Latent membership predicting dependent variables (ZY). The bias-adjusted ML three-step formulation follows Bakk, Tekle, and Vermunt (2013) and Nylund-Gibson, Grimm, and Masyn (2019), with the BCH secondary-model formulation described by Asparouhov and Muthén (2014b). The model estimates the conditional distribution of Y_{nv} given $Z_n = l$ separately for $v = 1, \ldots, V$. For `family = "gaussian"`, $Y_{nv} \mid Z_n = l \sim N(\mu_{lv}, \sigma^2_{lv})$. For `family = "categorical"`, $P(Y_{nv} = q \mid Z_n = l) = p_{lvq}$, where q indexes the observed categories of dependent variable v. No design matrix is required because the model contains a separate intercept for every class/profile.

ML/CEP maximizes

$$\ell_{\mathrm{ML},v} = \begin{cases} \sum_{n=1}^{N} \log \left\{ \sum_{l=1}^{L} \mathrm{CEP}(l, \widehat{Z}_n) \pi_l \mathcal{N}(Y_{nv} \mid \mu_{lv}, \sigma^2_{lv}) \right\}, & \text{for a Gaussian dependent variable,} \\ \sum_{n=1}^{N} \log \left\{ \sum_{l=1}^{L} \mathrm{CEP}(l, \widehat{Z}_n) \pi_l \prod_q p_{lvq}^{\mathbb{1}(Y_{nv}=q)} \right\}, & \text{for a categorical dependent variable,} \end{cases}$$

The reported class-shift rate compares the Step 3 modal class with $\widehat{Z}_n$ from Step 1.

BCH Gaussian means solve

$$\sum_{n=1}^{N} (\mathrm{CEP}^{-1})_{\widehat{Z}_n, l} (Y_{nv} - \mu_{lv}) = 0,$$

and the corresponding Gaussian variances solve

$$\sum_{n=1}^{N} (\mathrm{CEP}^{-1})_{\widehat{Z}_n, l} \{ (Y_{nv} - \mu_{lv})^2 - \sigma^2_{lv} \} = 0.$$

and categorical probabilities solve

$$\sum_{n=1}^{N} (\mathrm{CEP}^{-1})_{\widehat{Z}_n, l} \{ \mathbb{1}(Y_{nv} = q) - p_{lvq} \} = 0.$$

Parameter estimation and uncertainty

With `method.regression = "Analytic"`, ML-XZ, BCH-XZ, and ML-ZY use their exact scores or gradients within numerical optimization; BCH-ZY evaluates the closed-form weighted estimates shown above. `"Numeric"` evaluates the same likelihoods without supplied gradients or minimizes the squared BCH-ZY estimating equations numerically.

For ML-XZ and BCH, the covariance has sandwich form $A^{-1}BA^{-\top}$. In ML-XZ, A is the observed information and B is formed from individual likelihood scores plus the influence function of the estimated CEP matrix. In BCH, A is the estimating-equation Jacobian and B is the empirical covariance of individual estimating-function contributions. For BCH-ZY Gaussian models, the mean and variance equations are stacked so their sandwich covariance includes the covariance between $\widehat{\mu}_{lv}$ and $\widehat{\sigma}^2_{lv}$. ML-ZY uses the inverse Louis observed-information matrix; the variance standard error follows by applying the delta method to the fitted log-standard-deviation parameter. `method.SE = "Numeric"` evaluates the required Hessian or Jacobian numerically. `"Bootstrap"` resamples individuals, recalculates posterior assignments and CEP/BCH weights, and re-estimates Step 3 while holding the Step 1 measurement parameters fixed. For a Gaussian dependent variable, `estimate` and `se` report μ_{lv} and its standard error, whereas `variance` and `variance.se` report σ^2_{lv} and its standard error. Separate omnibus Wald tests assess equality of the conditional means and equality of the conditional variances across classes/profiles. For a categorical dependent variable, `estimate` and `se` report every class/profile-specific category probability and its standard error; the omnibus Wald test assesses equality of the complete conditional category distributions.

Method selection

ML is the default for XZ because its likelihood directly represents the error-prone modal assignment through CEP. BCH is the default for ZY because its weights are calculated without using $\mathbf{Y}$, so the Step 1 class/profile definition is not changed by the dependent variable. ML-ZY provides a corrected-likelihood sensitivity analysis and reports class shifts. Use a dedicated DCAT procedure when the DCAT estimand is required.

References

Asparouhov, T., & Muthén, B. (2014a). Auxiliary variables in mixture modeling: Three-step approaches using Mplus. *Structural Equation Modeling: A Multidisciplinary Journal, 21*(3), 329–341. doi:10.1080/10705511.2014.915181

Asparouhov, T., & Muthén, B. (2014b). *Auxiliary variables in mixture modeling: Using the BCH method in Mplus to estimate a distal outcome model and an arbitrary secondary model* (Mplus Web Note No. 21, Version 2). https://www.statmodel.com/examples/webnotes/webnote21.pdf

Bakk, Z., Tekle, F. B., & Vermunt, J. K. (2013). Estimating the association between latent class membership and external variables using bias-adjusted three-step approaches. *Sociological Methodology, 43*(1), 272–311. doi:10.1177/0081175012470644

Bolck, A., Croon, M., & Hagenaars, J. (2004). Estimating latent structure models with categorical variables: One-step versus three-step estimators. *Political Analysis, 12*(1), 3–27. doi:10.1093/pan/mph001

Nylund-Gibson, K., Grimm, R. P., & Masyn, K. E. (2019). Prediction from latent classes: A demonstration of different approaches to include distal outcomes in mixture models. *Structural Equation Modeling: A Multidisciplinary Journal, 26*(6), 967–985. doi:10.1080/10705511.2019.1590146

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

See Also

LCA(), LPA(), LTA(), get.CEP()

Examples

```
library(LCPA)

set.seed(1245)
N <- 2000
L <- 3
I <- 6

# Two observed covariates plus the required intercept
covariates <- cbind(
  Intercept = 1,
  Zeta.1 = as.numeric(scale(rnorm(N))),
  Zeta.2 = rbinom(N, 1, 0.5)
)
beta <- matrix(c(
   0.70,  0.30, 0,
   0.40, -0.20, 0,
  -0.30,  0.30, 0
), ncol = L, byrow = TRUE)
rownames(beta) <- colnames(covariates)

data.LCPA <- sim.LTA(
  N = N, I = I, L = L, times = 1, type = "LPA",
  constraint = "VE", mean.range = c(-3, 3),
  covs.range = c(0.4, 0.8),
  covariates = list(covariates), ref.class = 3,
  beta = beta, is.sort = TRUE
)
control.model <- list(
  constraint = "VE", is.sort = TRUE,
  starts = 10, maxiter.warmup = 10, nrep = 3
)

# Covariates predicting latent profiles: XZ analysis
fit.LCPA.XZ <- LCPA(
  response = data.LCPA$responses[[1]], L = L,
  type.analysis = "XZ", type.model = "LPA",
  covariates = covariates, ref.class = 3,
  method.model = "EM", control.model = control.model,
  method.3step = "ML", method.regression = "Analytic",
  method.SE = "Analytic", maxiter = 500, vis = TRUE
)
round(cbind(
  "True Class 1" = beta[, 1],
  "Estimate Class 1" = fit.LCPA.XZ$beta[, 1],
```

```
  "True Class 2" = beta[, 2],
  "Estimate Class 2" = fit.LCPA.XZ$beta[, 2]
), 3)

# Latent profiles predicting two dependent variables: ZY analysis
true.mean <- rbind(
  "Class 1" = c(Depression = 8, Anxiety = 12),
  "Class 2" = c(Depression = 10, Anxiety = 10),
  "Class 3" = c(Depression = 13, Anxiety = 8)
)
dependent.variables <- scale(
  true.mean[data.LCPA$Zs[[1]], ] +
    matrix(rnorm(N * 2, sd = 1.5), N, 2)
)
true.mean.standardized <- sweep(
  sweep(true.mean, 2, attr(dependent.variables, "scaled:center"), "-"),
  2, attr(dependent.variables, "scaled:scale"), "/"
)
true.variance.standardized <-
  (1.5 / attr(dependent.variables, "scaled:scale"))^2
dependent.variables <- as.data.frame(dependent.variables)
fit.LCPA.ZY <- LCPA(
  response = data.LCPA$responses[[1]], L = L,
  type.analysis = "ZY", type.model = "LPA",
  dependent.variables = dependent.variables,
  family = "gaussian",
  method.model = "EM", control.model = control.model,
  method.3step = "BCH", method.regression = "Analytic",
  method.SE = "Analytic", vis = TRUE
)
round(cbind(
  True.Depression = true.mean.standardized[, "Depression"],
  Estimate.Depression =
    fit.LCPA.ZY$dependent.variables$t1$Depression$estimate,
  True.Anxiety = true.mean.standardized[, "Anxiety"],
  Estimate.Anxiety =
    fit.LCPA.ZY$dependent.variables$t1$Anxiety$estimate
), 3)
round(cbind(
  True.Variance.Depression = rep(
    true.variance.standardized["Depression"], L
  ),
  Estimate.Variance.Depression =
    fit.LCPA.ZY$dependent.variables$t1$Depression$variance,
  True.Variance.Anxiety = rep(
    true.variance.standardized["Anxiety"], L
  ),
  Estimate.Variance.Anxiety =
    fit.LCPA.ZY$dependent.variables$t1$Anxiety$variance
), 3)
```

`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. This function returns the logistic/sigmoid transformation $1/(1 + \exp(-x))$.

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 the Expectation-Maximization (EM) algorithm, stochastic EM (SEM) through `flexmix` or `RMixtComp`, native EM/CEM/SEM strategies through `Rmixmod`, Neural Network Estimation (NNE), or external Mplus software.

Usage

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

Arguments

`response`	A numeric matrix of dimension $N \times I$, where N is the number of participants and I is the number of continuous observed indicators. 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).
`constraint`	Character string specifying covariance structure constraints: `"VV"` Varying variances and varying covariances across profiles (heterogeneous full covariance; Default). `"VE"` Varying variances but equal covariances 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).

"UE" Univariate response only: equal variance across profiles.

"UV" Univariate response only: varying variances across profiles.

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.

For method = "flexmix", all eight named structures and custom constraint lists are supported through LCPA's joint Gaussian M-step driver inside the flexmix SEM loop. For method = "Rmixmod", only "V0", "EE", and "VV" are supported; custom constraint lists are not supported. For method = "RMixtComp", only the locally independent "V0" structure is supported; custom constraint lists are not supported. For method = "Mplus", at least two indicators are required; the supported named structures are "E0", "V0", "EE", "VE", "EV", and "VV", together with custom constraint lists.

method Character string specifying estimation algorithm:

- "EM": Expectation-Maximization algorithm (Default).
- "NNE": Neural Network Estimation (experimental), using feed-forward layers, optional transformer attention, gradient optimization, and simulated annealing. See install_python_dependencies.
- "Mplus": Calls external Mplus software for estimation. Uses Mplus defaults for optimization unless overridden by control.Mplus.
- "flexmix": Stochastic EM (SEM) through flexmix, using LCPA's warm-up and promoted-replication scheme. Requires flexmix.
- "Rmixmod": LCPA warm-up plus SEM, or a native Rmixmod EM, CEM, or SEM strategy. Requires the Rmixmod package.
- "RMixtComp": Stochastic EM (SEM) estimation through RMixtComp. Requires RMixtComp and RMixtCompUtilities. No non-SEM RMixtComp algorithm is exposed.

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, 2000).
- 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.

For methods that expose this initialization interface, `par.ini` is used only to construct the `starts` warm-up initializations. It does not initialize the `nrep` refinement runs, which continue directly from the selected warm-up states. With `par.ini = "kmeans"`, each EM warm-up start calls `Kmeans.LPA` exactly once with one internal K-means start; NNE applies the same single-start K-means rule in its Python backend. Thus, `starts` is the number of separately initialized K-means outputs passed to warm-up training. Backends such as Mplus and Rmixmod may instead use their native initialization mechanisms and are not required to implement K-means initialization. If `"kmeans"` is requested for a method that does not support it, `par.ini` is automatically changed to `"random"`.

`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.

`starts` Positive integer. Number of warm-up analyses to run (default: 100). Each analysis is initialized by the selected method and trained for at most `maxiter.warmup` iterations, producing exactly `starts` warm-up solutions. With `par.ini = "kmeans"`, each EM warm-up start calls `Kmeans.LPA` exactly once with `starts = 1`; NNE applies the equivalent single-start rule in Python. The public `starts` argument is therefore also the number of K-means outputs.

`maxiter.warmup` Positive integer. Maximum number of training iterations for each of the `starts` warm-up analyses (default: 20). This limit applies only to warm-up and does not limit the subsequent refinement phase.

`nrep` Positive integer not exceeding `starts`. Number of refinement analyses (default: 20). The `nrep` warm-up solutions with the largest log-likelihoods are continued from their saved states until the full-training stopping rule is met; no new initialization occurs in this phase. The refined solution with the largest log-likelihood is returned as the final result. These three staged-training arguments are not used by `method = "RMixtComp"` or by `method = "Rmixmod"` with `control.Rmixmod$path = "Rmixmod"`; those paths use their documented native controls instead.

`vis` Logical. If `TRUE`, displays carriage-return-updated `Warm` and `Rep` lines when the backend exposes those stages, followed by one final fit-summary line (default: `TRUE`). Each stage occupies one console line and ends with one newline.

`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 under which Mplus writes intermediate files, including model input, data, output, and saved posterior probabilities. The effective default is `""`. A non-empty path is created recursively when necessary and must be writable. Within it, the function creates a unique timestamped subdirectory named `"Mplus_LPA_YYYY-MM-DD_HH-MM-SS"`

to isolate all files from the current run. If `files.path = ""`, that timestamped subdirectory is created directly under R's current working directory, `getwd()`. Explicit `NULL` is invalid.

`files.clean` Logical. If `TRUE` (default), all intermediate files and the temporary working directory created for the run are deleted on successful completion or error exit via `on.exit()`. If `FALSE`, the complete timestamped working directory is retained under `files.path`, or under `getwd()` when `files.path = ""`, for inspection and debugging.

`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` Positive offset used in the NNE objective (default: 1e-8).

`lambda` Coefficient of the NNE parameter penalty (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).

`patience.early` Maximum consecutive iterations without improvement before early stopping (default: 100).

`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: 0.80).

`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.

`control.flexmix`

List of control parameters for flexmix SEM estimation:

`maxiter` Number of SEM iterations in every promoted full run (default: 1000).

`minprior` Minimum component prior accepted by flexmix (default: 0, so LCPA does not intentionally remove requested profiles).

`tol` Relative likelihood-change threshold used by flexmix (default: 0, which enforces the fixed SEM iteration count).

`control.Rmixmod`

List of control parameters for Rmixmod estimation:

`path` Execution path: `"LCPA"` (default) preserves the package's `starts`/`maxiter.warmup`/`nrep` warm-up plus pure SEM procedure; `"Rmixmod"` delegates one native EM, CEM, SEM, or combined strategy and ignores those three arguments.

`algorithm`, `nrep`, `method.init`, `starts`, `maxiter.init`, `maxiter`, `tol.init`, `tol`, `par.ini`, `labels.i` Package-standard controls translated to the corresponding arguments of `Rmixmod::mixmodStrategy()` when `path="Rmixmod"`. Unspecified arguments retain the installed Rmixmod version's defaults. Under `path="LCPA"`, only `maxiter` is used, with 1000 iterations when omitted. `algorithm` accepts `"EM"`, `"CEM"`, and `"SEM"`, including ordered combinations supported by Rmixmod.

`strategy` Optional pre-built Rmixmod `Strategy` object for `path="Rmixmod"`. When supplied, it takes precedence over the individual strategy controls.

`control.RMixtComp`

List of control parameters for RMixtComp SEM estimation:

`maxiter.burnin` Number of native SEM burn-in iterations (default: 50).

`maxiter` Number of recorded post-burn-in SEM iterations (default: 50).

`maxiter.gibbs.burnin`, `maxiter.gibbs` Numbers of burn-in and recorded iterations in RMixtComp's subsequent fixed-parameter Gibbs stage (defaults: 50 and 50).

`n.init.per.class` Number of observations per class used by RMixtComp's native parameter initialization (default: 50).

`maxattempts.sem` Maximum number of SEM attempts (default: 20).

`confidence.level`, `stable.ratio`, `n.stable` Native RMixtComp SEM controls (defaults: 0.95, 0.99, and 20).

`criterion` RMixtComp model-selection criterion: `"BIC"` (default) or `"ICL"`.

`nrep` Number of native RMixtComp SEM runs for the requested number of profiles; RMixtComp retains the run with the largest observed likelihood (default: 1).

`ncores` Number of cores used by RMixtComp to parallelize `nrep`; must not exceed `nrep` (default: 1).

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 when applicable; for native Rmixmod and RMixtComp paths, the selected native fit's BIC.

`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. For native Rmixmod and RMixtComp paths, this is the selected native fit's scalar log-likelihood.

`model` The optimal model object:

- For `method="NNE"`: Trained neural network model.
- For `method="Mplus"`: Estimated Mplus model.
- For `method="flexmix"`: Selected `flexmix` SEM object.
- For `method="Rmixmod"`: Selected `MixmodCluster` object.
- For `method="RMixtComp"`: Selected `MixtCompLearn` SEM object.

`call` Matched function call.

`arguments` A list containing all effective input arguments.

Random-number reproducibility

Except for `method = "NNE"`, which intentionally uses its fixed backend seed, every stochastic estimator is driven from R's current random-number generator. The user only needs to call `set.seed()` immediately before `LPA()` to reproduce EM, K-means, flexmix, Rmixmod, RMixtComp, and Mplus estimation. LCPA automatically passes an R-derived seed to backends with independent random streams; no backend-specific seed setting is required.

Notation

Write the response matrix as $\mathbf{X} = (X_{ni})_{N\times I}$, where $n = 1, 2, \ldots, N$ indexes participants and $i = 1, 2, \ldots, I$ indexes observed indicators. The response vector for participant n is $\mathbf{X}_n = (X_{n1}, \ldots, X_{nI})^\top$. The latent profile variable is $Z_n \in \{1, 2, \ldots, L\}$, and $l = 1, 2, \ldots, L$ indexes a particular latent profile.

With profile mean $\boldsymbol{\mu}_l$ and covariance $\boldsymbol{\Sigma}_l$, the observed-data log-likelihood is

$$\log \mathcal{L}_{\text{LPA}} = \sum_{n=1}^{N} \log \left\{ \sum_{l=1}^{L} \pi_l \mathcal{N}(\mathbf{X}_n \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l) \right\}.$$

EM Algorithm

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

$$\log \mathcal{L}_{\text{LPA}} = \sum_{n=1}^{N} \log \left\{ \sum_{l=1}^{L} \pi_l \mathcal{N}(\mathbf{X}_n \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l) \right\}.$$

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 participant n and class l:

$$\tau_{nl} = P(Z_n = l \mid \mathbf{X}_n) = \frac{\pi_l \mathcal{N}(\mathbf{X}_n \mid \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}{\sum_{h=1}^{L} \pi_h \mathcal{N}(\mathbf{X}_n \mid \boldsymbol{\mu}_h, \boldsymbol{\Sigma}_h)}.$$

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.

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"` / `"EV"` Hybrid constraints (e.g., `"VE"`: varying variances, equal covariances). The covariance part of the standard EM Q-function is maximized numerically under the exact equality constraints.

  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 estimated by maximizing the covariance part of the standard EM Q-function under the specified equalities.

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 indicators $\mathbf{X}_n \in \mathbb{R}^I$ are standardized (mean-centered and scaled to unit variance) during training. No encoding is required.

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.

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 one covariance matrix $\boldsymbol{\Sigma}_l$ for each profile.

Constraint handling:

- Covariance constraints (`constraint`) are enforced after activation; variances or covariances marked for equality are shared across profiles.
- 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:

Working Directory Setup Creates a timestamped `"Mplus_LPA_YYYY-MM-DD_HH-MM-SS"` directory under `control.Mplus$files.path`, or under the current working directory when that path is empty, 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$files.clean = 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
  `STSEED` Random-start seed drawn from R's current random-number generator
  `STITERATIONS = maxiter.warmup` 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 restrictions are encoded directly in the generated Mplus model syntax before estimation; they are not imposed by post-estimation averaging.
- The named structures are translated as follows:

  - "E0": variances carry common equality labels across profiles and all covariances are fixed to zero.
  - "V0": variances are freely estimated within profile and all covariances are fixed to zero.
  - "EE": variances and covariances carry common equality labels across profiles.
  - "VE": variances vary across profiles while covariances carry common equality labels.
  - "EV": variances carry common equality labels while covariances vary across profiles.
  - "VV": all variances and covariances vary across profiles.
- For a custom list, c(i, i) assigns a common cross-profile label to the variance of variable i, whereas c(i, j) assigns one to the covariance between variables i and j. Unlisted parameters remain profile-specific, and covariance symmetry is represented by a single Mplus WITH parameter for each variable pair.
- "UE" and "UV" are not available through this Mplus backend because it requires at least two indicators. They remain available for the univariate EM and flexmix paths.

flexmix Stochastic EM

With method = "flexmix", each SEM iteration performs one stochastic classification draw from the current posterior probabilities before the M-step. LCPA runs exactly starts short trajectories of maxiter.warmup iterations, promotes the best nrep trajectories by observed log-likelihood, continues each for control.flexmix$maxiter SEM iterations, and retains the largest-likelihood final state across the promoted runs. This selection compares the final state returned by each flexmix SEM run; flexmix's classify = "SEM" does not retain the largest-likelihood state visited within a run. The stochastic classification and iteration control remain those of flexmix. LCPA supplies a joint Gaussian M-step driver so that "UE", "UV", "E0", "V0", "EE", "VV", "VE", "EV", and custom equality lists obey the same covariance definitions used by LCPA's EM estimator. Setting control.flexmix$tol = 0 prevents likelihood-based early termination. Only SEM is exposed; par.ini is not used.

Rmixmod stochastic strategies

When method = "Rmixmod", estimation uses Rmixmod::mixmodCluster(). The covariance constraints map to Rmixmod Gaussian models as follows: "V0" to "Gaussian_pk_Lk_Bk", "EE" to "Gaussian_pk_L_C", and "VV" to "Gaussian_pk_Lk_Ck".

With path="LCPA", the function generates exactly starts random balanced partitions and runs exactly maxiter.warmup consecutive stochastic E-S-M iterations for every warm-up start. The best nrep warm-up parameter sets are then passed directly to independent Rmixmod SEM runs, and the finite solution with the largest final log-likelihood is retained. Rmixmod's internal "smallEM" and "SEMMax" initialization searches are not used; in particular, "SEMMax" is an initialization search rather than one consecutive SEM trajectory for each user-level start. par.ini is not used for this method. SEM stops after control.Rmixmod$maxiter iterations; an epsilon convergence criterion is not defined for SEM in Rmixmod. Unlike flexmix's classify = "SEM", Rmixmod retains the largest-likelihood parameter state visited within each SEM run, so equal iteration counts do not imply identical final-state selection. Custom covariance-constraint lists are not accepted by the Rmixmod backend.

With path="Rmixmod", LCPA translates all non-NULL package-standard strategy controls to Rmixmod::mixmodStrategy() and runs one native mixmodCluster() call. Its algorithm may contain "EM", "CEM", "SEM", or an ordered combination of these algorithms. LCPA does not add its outer starts, maxiter.warmup,

or nrep; control.Rmixmod$nrep controls complete strategy repetitions. The published strategy of Mulder et al. (2015) uses 200 SEM iterations followed by EM with a relative likelihood-change tolerance of 1e-5. For reproducibility across Rmixmod versions, the example and simulation scripts explicitly pin the contemporaneous documented defaults: smallEM, 50 initialization tries, 5 initialization iterations, tol.init=0.001, and a 200-iteration EM limit.

RMixtComp Stochastic EM

When method = "RMixtComp", LCPA calls RMixtComp::mixtCompLearn() in classic, non-hierarchical learning mode with a univariate Gaussian model for every indicator. This integration exposes only RMixtComp's stochastic EM (SEM) algorithm; it does not introduce any other RMixtComp estimation algorithm. Conditional independence implies diagonal, class-varying covariance matrices, so only constraint = "V0" is accepted. Custom covariance-constraint lists are not accepted by the RMixtComp backend. LCPA does not add its own starts, maxiter.warmup, or nrep stages to this backend. Each native run performs RMixtComp initialization, SEM burn-in, recorded SEM iterations, and then the fixed-parameter Gibbs burn-in and recorded Gibbs iterations. One stochastic S-step is performed per SEM iteration, between the E-step and M-step. Native repetition and parallelization are controlled only by control.RMixtComp$nrep and control.RMixtComp$ncores. RMixtComp's native random stream is independent of R, so LCPA passes it one integer drawn from R's current random stream. The user only needs an external set.seed() for reproducibility. The default algorithm controls reproduce RMixtCompUtilities::createAlgo() defaults (version 4.1.4 or later). par.ini is not used by this backend.

References

Biernacki, C. (2015). MixtComp software: Model-based clustering/imputation with mixed data, missing data and uncertain data. *MISSDATA 2015*. https://inria.hal.science/hal-01253393

Leisch, F. (2004). FlexMix: A general framework for finite mixture models and latent class regression in R. *Journal of Statistical Software, 11*(8), 1–18. doi:10.18637/jss.v011.i08

McLachlan, G. J., & Peel, D. (2000). *Finite mixture models*. John Wiley & Sons.

Mulder, V. L., Lacoste, M., Martin, M. P., Richer-de-Forges, A., & Arrouays, D. (2015). Understanding large-extent controls of soil organic carbon storage in relation to soil depth and soil-landscape systems. *Global Biogeochemical Cycles, 29*(8), 1210–1229. doi:10.1002/2015GB005178

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.
# An empty 'files.path' instead uses the current working directory.
# 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)

# Fit an EE model with flexmix SEM and the joint constrained M-step
# need flexmix
## Not run:
  fit_flexmix <- LPA(response, L = 2, constraint = "EE", method = "flexmix",
                     nrep = 2, starts = 5, maxiter.warmup = 5,
                     control.flexmix = list(maxiter = 50))

## End(Not run)

# Fit 2-profile model with the published Rmixmod SEM-to-EM strategy
# need Rmixmod
## Not run:
  fit_rmixmod <- LPA(response, L = 2, constraint = "VV", method = "Rmixmod",
                     control.Rmixmod = list(path = "Rmixmod",
                                            algorithm = c("SEM", "EM"),
                                            nrep = 1,
                                            method.init = "smallEM",
                                            starts = 50,
                                            maxiter.init = 5,
                                            tol.init = 0.001,
                                            maxiter = c(200, 200),
                                            tol = c(NA, 1e-5)))

## End(Not run)
```

LRT.test *Likelihood Ratio Test*

Description

Conducts a likelihood ratio test to compare the fit of two LCA or LPA models with any numbers of latent classes/profiles, including equal class counts. 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 LCA or LPA model. When both models have the same number of free parameters, this is treated as the null model.
`object2`	Fitted LCA or LPA model of the same type. When both models have the same number of free parameters, this is treated as the alternative 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. Models may have any class counts; they do not need to differ by exactly one class. If both models have the same number of free parameters, the likelihood-ratio statistic is returned but the chi-square p-value is `NA` because its reference distribution has zero degrees of freedom.

Value

An object of class "htest" containing:

- statistic: Standard likelihood ratio 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 latent class analysis (LCA) or latent profile analysis (LPA) models with any numbers of latent classes/profiles, including equal class counts. Implements both fixed-replicate and sequential stopping procedures for computational efficiency.

Usage

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

Arguments

object1	Fitted model object. Must be of class "LCA" or "LPA". When both models have the same number of free parameters, this is treated as the null model.
object2	Fitted model object of the same class as object1. When both models have the same number of free parameters, this is treated as the alternative model.
nrep.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. The default of 100 replicates follows 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 nrep.bootstrap replicates (traditional fixed bootstrap).

Details

Models may have any class counts; they do not need to differ by exactly one class. Models are ordered by their numbers of free parameters. If these are equal, the supplied order is retained and bootstrap samples are generated under `object1`.

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. Class labels are sampled with replacement from a pool constructed by rounding $NP(Z = l)$ and adjusting the largest class to length N.
- Model Re-fitting: For each simulated dataset, both the null and alternative models are re-estimated. Each refit uses `update()` to preserve the supplied model's fitting controls while replacing `response`, setting `par.ini = "random"` and `is.sort = FALSE`, and suppressing progress output. If that refit errors, the same update is retried with `method = "EM"`.
- 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 `nrep.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 null model). This vector can be used for diagnostic purposes, such as inspecting the empirical null distribution or computing alternative p-value estimates.
- `nrep.bootstrap.requested`: Maximum number of bootstrap replicates requested.
- `nrep.bootstrap.completed`: Actual number of bootstrap replicates completed.
- `negative.LRT`: Number of bootstrap LRT statistics below zero.
- `negative.LRT.proportion`: Proportion of bootstrap LRT statistics below zero.

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. doi:10.1080/10705510701575396

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

Description

Computes the Mplus TECH11 implementation of the Vuong-Lo-Mendell-Rubin likelihood ratio test (VLMR; Vermunt, 2024) and the Lo-Mendell-Rubin adjusted test (aLMR; Lo et al., 2001) for LCA or LPA models with any numbers of latent classes/profiles, including equal class counts.

Usage

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

Arguments

`object1`	Fitted `LCA` or `LPA` model. When both models have the same number of free parameters, this is treated as the null model.
`object2`	Fitted model of the same type and fitted to the same data. When both models have the same number of free parameters, this is treated as the alternative model.

Details

This function reports two related tests. The unadjusted VLMR is based on the likelihood-ratio test for nested or overlapping models developed by Vuong (1989), which Lo, Mendell, and Rubin (2001) applied to the comparison of K- and $K+1$-component mixture models. The adjusted LMR test (aLMR) is the modification proposed by Lo et al. (2001). The robust reference distribution follows the Mplus TECH11 reconstruction described by Vermunt (2024) and is calculated from observation-level scores and observed-information matrices as follows.

Let H_0 denote the null model and H_1 the alternative model, ordered by parameter count as described above. For model $r \in \{0, 1\}$, let ℓ_{rn} be participant n's log-likelihood contribution, s_{rn} its score vector, and $I_r = -\sum_{n=1}^{N} \partial^2 \ell_{rn} / \partial\theta_r \partial\theta_r^{\mathsf{T}}$ the observed information matrix. Define

$$B_r = \sum_{n=1}^{N} s_{rn} s_{rn}^{\mathsf{T}}, \qquad B_{10} = \sum_{n=1}^{N} s_{1n} s_{0n}^{\mathsf{T}},$$

and the robust sandwich covariance matrix

$$R_r = I_r^{-1} B_r I_r^{-1}.$$

Following the Mplus implementation identified by Vermunt (2024), the reference-distribution weights are the eigenvalues of

$$W_{\text{Mplus}} = \begin{bmatrix} B_1 R_1 & B_{10} R_0 \\ -B_{10}^{\mathsf{T}} R_1 & -B_0 R_0 \end{bmatrix}.$$

If these eigenvalues are $\lambda_1, \ldots, \lambda_m$, the unadjusted statistic and its estimated null distribution are

$$\text{LR} = 2\{\ell(\widehat{\theta}_1) - \ell(\widehat{\theta}_0)\}, \qquad Q = \sum_{j=1}^{m} \lambda_j \chi^2_{1,j}.$$

Thus, the VLMR p-value is $\Pr(Q \geq \mathrm{LR})$. It is evaluated using Imhof's (1961) method, retaining negative eigenvalue weights. The reported mean and standard deviation of the reference distribution are $\sum_j \lambda_j$ and $\{2\sum_j \lambda_j^2\}^{1/2}$, respectively (Vermunt, 2024).

The aLMR adjustment is specifically Equation (15) of Lo et al. (2001):

$$\mathrm{aLMR} = \frac{\mathrm{LR}}{c}, \qquad c = 1 + \frac{1}{(p-q)\log(N)},$$

where $p - q$ is the difference in the numbers of free parameters. The adjusted statistic uses the same weighted chi-square reference distribution as the unadjusted VLMR statistic (Vermunt, 2024). Consequently, `p.value` is the unadjusted VLMR result derived from Vuong (1989) and Lo et al. (2001), whereas `adjusted.p.value` is the aLMR result based on Equation (15) of Lo et al. (2001). Models may have any class counts; they do not need to differ by exactly one class. When their numbers of free parameters are equal, the unadjusted VLMR is computed, but the aLMR correction factor, adjusted statistic, and adjusted p-value are `NA` because Equation (15) contains $p - q$ in the denominator.

The result matches the TECH11 calculation conditional on the two supplied maximum-likelihood solutions. Local maxima, singular information matrices, and boundary solutions can invalidate the comparison.

Value

An object of classes `"VLMR"` and `"htest"` containing:

- `statistic`: Unadjusted VLMR likelihood ratio statistic.
- `p.value`: VLMR p-value from the Mplus weighted chi-square distribution.
- `adjusted.statistic`: LMR statistic adjusted using Equation (15) of Lo et al. (2001).
- `adjusted.p.value`: Adjusted LMR p-value from the same weighted chi-square distribution.
- `correction.factor`: Correction factor from Equation (15) of Lo et al. (2001).
- `distribution`: Mean, standard deviation, and eigenvalue weights of the estimated VLMR reference distribution.
- `diagnostics`: Score, information-matrix, eigenvalue, and Imhof integration diagnostics.

References

Imhof, J. P. (1961). Computing the distribution of quadratic forms in normal variables. *Biometrika, 48*(3–4), 419–426. doi:10.1093/biomet/48.34.419

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

Vermunt, J. K. (2024). The Vuong-Lo-Mendell-Rubin test for latent class and latent profile analysis: A note on the different implementations in Mplus and LatentGOLD. *Methodology, 20*(1), 72–83. doi:10.5964/meth.12467

Vuong, Q. H. (1989). Likelihood ratio tests for model selection and non-nested hypotheses. *Econometrica, 57*(2), 307–333. doi:10.2307/1912557

LTA *Latent Transition Analysis*

Description

Estimates initial latent-state probabilities and conditional transition matrices across repeated LCA/LPA measurements. `type.analysis = "XZ"` estimates covariate effects on initial status and transitions; `"ZY"` estimates external observed dependent-variable distributions conditional on a latent state at a selected time or on the complete longitudinal latent-state path. These auxiliary variables are not included among the indicators.

Usage

```
LTA(
  responses,
  L = 2,
  type.analysis = c("XZ", "ZY"),
  type.model = c("LCA", "LPA"),
  covariates = NULL,
  covariates.time.cross = FALSE,
  ref.class = L,
  dependent.variables = NULL,
  family = "gaussian",
  dependent.variable.structure = c("State", "Path"),
  dependent.variable.time = NULL,
  dependent.variable.time.cross = FALSE,
  method.model = "EM",
  step1.pool = FALSE,
  control.model = NULL,
  control.EM = NULL,
  control.Mplus = NULL,
  control.NNE = NULL,
  control.Rmixmod = NULL,
  method.3step = NULL,
  CEP.error = TRUE,
  CEP.time.cross = FALSE,
  method.regression = "Analytic",
  maxiter = 5000,
  tol = 1e-04,
  lower = -10,
  upper = 10,
  method.SE = "Bootstrap",
  nrep.bootstrap = 100,
  vis = TRUE
)
```

Arguments

`responses` A non-empty list of $N \times I$ numeric response matrices/data frames, one per time point. All elements must contain the same participants in the same row order and have identical numbers of rows and indicators. For `type.model = "LCA"`, columns are categorical and follow the same requirements as `response` in `LCA()`. For `type.model = "LPA"`, columns are continuous, contain no missing values, and should be standardized with `scale()` or `normalize()` as for `LPA()`.

`L` Integer number of latent classes/profiles at every time point (default: 2; must be at least 2). It has the same meaning as `L` in `LCA()`/`LPA()` and is held constant over time.

`type.analysis` Character string selecting the analysis path:

- `"XZ"`: estimate covariate effects on initial latent status and subsequent latent transitions; `covariates` is used and `dependent.variables` is ignored.
- `"ZY"`: estimate distributions of external observed dependent variables conditional on a selected latent state or on the complete latent-state path; `dependent.variables`, `family`, `dependent.variable.structure`, `dependent.variable.time`, and `dependent.variable.time.cross` are used, while `covariates`, `covariates.time.cross`, and `ref.class` are ignored.

The default is `"XZ"`.

`type.model` Character string selecting the common Step 1 measurement model: `"LCA"` for categorical indicators or `"LPA"` for continuous indicators (default: `"LCA"`). Its meaning, response-data requirements, and fitted parameters are the same as in `LCA()` and `LPA()`, respectively.

`covariates` Used only when `type.analysis = "XZ"`. A list with one $N \times (U_t + 1)$ numeric matrix/data frame per time point, representing $\boldsymbol{\zeta}_{nt} = (1, \zeta_{nt1}, \ldots, \zeta_{ntU_t})^\top$. Each first column must be an unstandardized all-ones intercept; standardize all other covariates before analysis. The first element models initial status and later elements model transitions into their corresponding time points. If `NULL` (default), an all-ones design is created at every time point, giving an intercept-only initial/transition model. This argument is ignored for `"ZY"`, which uses implicit latent-state-specific dependent-variable intercepts.

`covariates.time.cross`
Logical used only for `"XZ"` (default: `FALSE`). If `TRUE`, constrain the transition-regression coefficients to be equal at times 2, ..., T; those transition design matrices must have identical column structures. The initial-state coefficients remain a separate block. If `FALSE`, estimate time-specific transition effects. Ignored for `"ZY"`.

`ref.class` Integer from 1 to `L` selecting the reference destination class in the `X -> Z` initial-state and transition multinomial regressions (default: `L`). The corresponding coefficient vectors are fixed to zero. If `control.model$is.sort = TRUE`, it refers to the common class order established after Step 1 sorting. It is not used for `"ZY"`.

`dependent.variables`
Used only when `type.analysis = "ZY"`. Supply either one dependent-variable vector/matrix/data frame or a list having the same length as `responses`. Each

supplied object must have N rows; at time t, its V_t columns represent $\mathbf{Y}_{nt} = (Y_{nt1}, \ldots, Y_{ntV_t})^\top$. They are external observed dependent variables, not latent-class indicators. Elements of a longitudinal list may be NULL at unmeasured time points. Missing values are allowed and are omitted separately by dependent variable. Before calling LTA(), standardize every continuous dependent-variable column assigned family = "gaussian", preferably with scale(), so that its observed sample mean is 0 and sample standard deviation is 1. Do not standardize columns assigned family = "categorical"; retain their original category values.

family — Used only for "ZY". "gaussian" (default) estimates class-specific means for standardized numeric continuous dependent variables; "categorical" estimates class-specific probabilities for observed categories. Supply one value for all dependent-variable columns, one value per column, or a list aligned with responses whose selected elements follow either rule. For categorical BCH analyses, the reported probabilities are class-specific conditional distributions of the dependent variable.

dependent.variable.structure — Used only for "ZY". "State" (default) estimates the distribution of each dependent variable conditional on the latent state at its selected time. "Path" estimates the distribution conditional on each complete latent-state path $(Z_{n1}, \ldots, Z_{nT})$, giving L^T path-specific groups.

dependent.variable.time — Used only for "ZY". Integer time-point index/indices. For a longitudinal dependent.variables list, NULL (default) selects every non-NULL element; an explicit vector selects distinct non-NULL elements. When dependent.variables is a single vector/matrix/data frame, exactly one time must be selected and NULL assigns it to the final time point.

dependent.variable.time.cross — Logical used only for "ZY" (default: FALSE). If FALSE, fit a separate class-specific dependent-variable model at each selected time. If TRUE, stack the selected records and constrain their class-specific dependent-variable parameters to equality over time. Pooled times must have identical dependent-variable column names and families. BCH analytic/numeric inference uses a participant-cluster sandwich; for pooled ML, use method.SE = "Bootstrap" to retain within-participant dependence.

method.model — Character string selecting the Step 1 parameter estimator. It has the same meaning as the method argument of LCA() or LPA(). Longitudinal three-step models support "EM" (default), "NNE", "Mplus", and "Rmixmod"; the corresponding control.* list is passed to the Step 1 function. It does not control Step 3 estimation.

step1.pool — Logical (default: FALSE). If FALSE, estimate the common Step 1 measurement model from responses[[1]] and apply it at all times. If TRUE, row-bind all response matrices, estimate one common model from the pooled records, and then split posterior probabilities back by time. This argument is ignored when fixed control.model$params are supplied.

control.model — Optional named list of common Step 1 measurement-model settings. If NULL (default), all settings below use their defaults. Supply only the elements to override:

params Optional fixed Step 1 parameter list (default: NULL). When supplied, Step 1 fitting is skipped. For LCA it must contain par, P.Z, and category.levels; for LPA it must contain means, covs, and P.Z, with the definitions and dimensions returned by LCA() and LPA().

par.ini Initialization used when params = NULL (default: "random"). It accepts "random", "kmeans", or the model-specific parameter-list forms documented for LCA() and LPA().

constraint LPA covariance structure (default: "VV"). It accepts "UE", "UV", "E0", "V0", "EE", "VV", "VE", "EV", or a custom equality list as documented for LPA(). It is ignored when type.model = "LCA".

is.sort Logical (default: TRUE). Order classes/profiles by decreasing Step 1 prior probability and consistently permute Step 1, CEP, and Step 3 class-specific results.

starts Positive integer number of independently initialized Step 1 warm-up analyses (default: 100).

maxiter.warmup Positive integer maximum number of Step 1 iterations per warm-up analysis (default: 20).

nrep Positive integer not exceeding starts (default: 20). The best warm-up states are continued to final Step 1 fits.

Element names must be unique; unknown or unnamed elements are rejected.

control.EM Optional Step 1 control list for the EM estimator in LCA() or LPA(). maxiter sets the maximum number of EM iterations, tol sets the convergence tolerance, and the LPA covariance-floor element bounds small covariance eigenvalues away from zero. See control.EM in those functions for the complete list.

control.Mplus Optional list passed to the selected LCA()/LPA() Mplus backend. maxiter, tol, files.path, files.clean, and the LPA covariance-floor option have the meanings documented there.

control.NNE Optional list passed to the selected LCA()/LPA() NNE backend. Its network, attention, annealing, optimizer, plotting, and device fields have exactly the meanings documented for those functions.

control.Rmixmod

Optional list passed to the selected LCA()/LPA() Rmixmod backend. Its package-staged/native path, algorithm, strategy, initialization, iteration, tolerance, and replication fields retain the meanings and restrictions documented there.

method.3step Character string selecting the Step 2–3 correction:

- "ML": use the classification-error-corrected likelihood. For "XZ", the full latent path is evaluated by forward–backward recursion rather than independent pairwise regressions.
- "BCH": invert each CEP matrix and fit Step 3 using BCH pseudo-weights. For transitions, joint weights are formed under conditional independence of time-specific classification errors.

If NULL (default), "ML" is selected for "XZ" and "BCH" for "ZY". Both choices are available for both paths. The longitudinal use of ML three-step and BCH auxiliary-variable models is demonstrated by Nylund-Gibson, Arch, and Carter

(2026). For the underlying cross-sectional corrections, BCH-XZ follows Vermunt (2010), whereas ML-ZY follows Bakk, Tekle, and Vermunt (2013) and Nylund-Gibson, Grimm, and Masyn (2019).

`CEP.error` Logical. If `TRUE` (default and recommended), estimate CEP matrices with `get.CEP()` and apply the selected ML or BCH classification-error correction. If `FALSE`, use identity matrices, reducing Step 3 to naive modal-state assignment.

`CEP.time.cross` Logical. If `FALSE` (default), estimate a separate CEP matrix at every time. If `TRUE`, use one posterior-mass-weighted common CEP matrix across time, which assumes invariant measurement and classification error; measurement invariance is not tested internally. The option applies to both `"XZ"` and `"ZY"`. The estimator and pooling rule are documented in `get.CEP()`.

`method.regression`
Character string controlling Step 3 point estimation (default: `"Analytic"`). For ML-XZ, BCH-XZ, and ML-ZY State/Path models, `"Analytic"` supplies the exact analytic gradient to a numerical optimizer; these models do not generally have closed-form coefficient estimates. For BCH-ZY, `"Analytic"` directly evaluates the weighted closed-form class-specific means or categorical probabilities, separately or under the requested cross-time equality constraint. `"Numeric"` optimizes the same likelihood without the supplied gradient or numerically solves the same BCH estimating equations for either the selected-state or complete-path groups. This choice is independent of `method.SE`.

`maxiter` Positive integer giving the maximum number of Step 3 `"XZ"` optimization iterations (default: 5000). It is distinct from Step 1 `control.*$maxiter` and `control.model$maxiter.warmup`, and is ignored for `"ZY"`.

`tol` Positive finite convergence tolerance for the Step 3 `"XZ"` optimization (default: 10^{-4}). It is separate from the Step 1 backend tolerance and is ignored for `"ZY"`.

`lower, upper` Finite lower and upper bounds applied to each `"XZ"` initial-state or transition-regression coefficient (defaults: -10 and 10). They are ignored for `"ZY"`.

`method.SE` Character string selecting Step 3 uncertainty estimation:

- `"Analytic"`: for ML-XZ, use the analytic observed-information bread and a participant-level sandwich meat that includes CEP estimation; for ML-ZY, use Louis observed information; for BCH, use the analytic estimating-equation bread and participant-cluster sandwich meat.
- `"Numeric"`: replace the analytic information/bread with a numerical Hessian or Jacobian while retaining the corresponding participant-level meat and Step 3 estimand.
- `"Bootstrap"` (default): jointly resample participants over all times, keep Step 1 measurement parameters fixed, recompute posterior assignments and CEP/BCH weights, and re-estimate Step 3.

`nrep.bootstrap` Integer number of bootstrap replications used only when `method.SE = "Bootstrap"` (default: 100; minimum: 2). Only successful Step 3 fits enter the empirical covariance; consider 500–1000 replications for final publication-level inference when feasible.

`vis` Logical (default: `TRUE`). If enabled, display Step 1 measurement estimation, time-specific Step 2 posterior/CEP preparation, and Step 3 transition-regression

or distal-dependent-variable progress. Each Step 2 and Step 3 heading identifies the selected X -> Z or Z -> Y path. For Z -> Y, the output also gives the number and family of dependent-variable models, convergence and iteration information, and the selected standard-error or bootstrap progress.

Details

At time $t = 1, 2, \ldots, T$, write the indicator matrix as $\mathbf{X}_t = (X_{nti})_{N \times I}$, where $n = 1, 2, \ldots, N$ indexes participants and $i = 1, 2, \ldots, I$ indexes observed indicators. Participant n's indicator vector is $\mathbf{X}_{nt} = (X_{nt1}, \ldots, X_{ntI})^\top$, and $Z_{nt} \in \{1, 2, \ldots, L\}$, with $l = 1, 2, \ldots, L$ indexing latent states.

The covariate vector is $\boldsymbol{\zeta}_{nt} = (1, \zeta_{nt1}, \ldots, \zeta_{ntU_t})^\top$, where $u = 1, 2, \ldots, U_t$ indexes the observed covariates available at time t, and $U = \max_t U_t$. The leading 1 is the intercept. The dependent-variable vector is $\mathbf{Y}_{nt} = (Y_{nt1}, \ldots, Y_{ntV_t})^\top$, where $v = 1, 2, \ldots, V_t$ indexes external observed dependent variables and $V = \max_t V_t$. Neither $\boldsymbol{\zeta}_{nt}$ nor $\mathbf{Y}_{nt}$ is part of $\mathbf{X}_{nt}$. "XZ" is the function-interface label for the covariate-to-latent path; the formulas use $\boldsymbol{\zeta}$ for its covariates because $\mathbf{X}$ is reserved for the LCA/LPA indicator data. type.analysis = "XZ" uses $\boldsymbol{\zeta}_{nt}$ but not $\mathbf{Y}_{nt}$; type.analysis = "ZY" uses $\mathbf{Y}_{nt}$ but not $\boldsymbol{\zeta}_{nt}$. These are separate Step 3 analyses rather than one jointly estimated longitudinal mediation model. For a ZY analysis, standardization applies to the continuous dependent variables in $\mathbf{Y}_{nt}$, not to the Step 1 indicator matrices $\mathbf{X}_t$. When dependent.variable.time.cross = FALSE, standardize each Gaussian dependent-variable column separately within its supplied time point. When dependent.variable.time.cross = TRUE, apply one common centering and scaling transformation to the corresponding Gaussian column pooled over all selected time points. The resulting estimates and standard errors are expressed in observed-standard-deviation units. Categorical dependent variables retain their original category values.

Value

An object of class "LTA", organized by analysis stage:

Analysis identity type.analysis, type.model, method.3step, dependent.variable.structure, and analysis. analysis$type records the selected path; exactly one of analysis$XZ and analysis$ZY is non-NULL. The same selected result is also available from the top-level XZ or ZY component.

Initial-state and transition model When type.analysis = "XZ", beta[, l] is $\boldsymbol{\beta}_l$, and gamma[[t - 1]][[k]][[l]] is $\boldsymbol{\gamma}_{klt}$, the coefficient vector defining row k, column l of $\mathbf{P}_{nt}$. beta.se, gamma.se, z statistics, p-values, vcov, information, and SE.diagnostics provide the corresponding inference. Log.Lik, AIC, BIC, iterations, converged, Log.Lik.history, and bound.diagnostics describe estimation of the initial-state and transition likelihood. These fields are available both in XZ and at the top level; ZY is NULL.

Measurement model and classification params contains the common fitted or supplied LCA/LPA measurement parameters. P.Z.Xns, P.Zs, and Zs are time-indexed posterior probabilities, marginal latent-state proportions, and modal assignments. CEP is the corresponding list of time-specific or constrained classification-error matrices, and step1.pool records the measurement-model data source.

ZY Step 3 result When type.analysis = "ZY", dependent.variables is nested by selected measurement time and observed dependent variable. Each fitted variable contains its family, state- or path-specific estimate and se, convergence status, and method-specific diagnostics.

Gaussian models additionally contain `variance`, `variance.se`, `variance.vcov`, `mean.variance.vcov`, and separate `omnibus` and `omnibus.variance` Wald tests. Categorical models contain class-by-category probability `estimate` and `se` matrices and an omnibus test of their complete conditional distributions. `vcov`, `SE.diagnostics`, `classification`, `case.weights` where applicable, and `diagnostics` provide the associated inference and correction information. With `dependent.variable.structure = "Path"`, `latent.paths` gives the $L^T \times T$ matrix that maps each reported path name to its time-specific latent states. ML path models also report their fitted initial-state and transition parameters within each dependent-variable model. `XZ` is `NULL`, and `ZY` points to `dependent.variables`.

Reproducibility `call` stores the matched call and `arguments` stores the normalized public arguments, including `step1.pool` and the expanded `control.model` defaults.

Latent transition model

For `type.analysis = "XZ"`, the central model is the evolution of $Z_{n1}, Z_{n2}, \ldots, Z_{nT}$. Let `ref.class` be the reference destination state l_0. The initial-state probabilities are

$$\pi_{n1l} = P(Z_{n1} = l \mid \boldsymbol{\zeta}_{n1}) = \frac{\exp(\boldsymbol{\zeta}_{n1}^\top \boldsymbol{\beta}_l)}{\sum_{h=1}^{L} \exp(\boldsymbol{\zeta}_{n1}^\top \boldsymbol{\beta}_h)},$$
$$\boldsymbol{\beta}_{l_0} = 0.$$

For each $t = 2, \ldots, T$, participant n's conditional transition matrix is

$$\mathbf{P}_{nt} = [P(Z_{nt} = l \mid Z_{n,t-1} = k, \boldsymbol{\zeta}_{nt})]_{k,l=1}^{L}.$$

Its row k gives the destination-state distribution conditional on origin state k; its column l corresponds to destination state l. Every row therefore sums to 1. The (k, l) entry is

$$P(Z_{nt} = l \mid Z_{n,t-1} = k, \boldsymbol{\zeta}_{nt}) = \frac{\exp(\boldsymbol{\zeta}_{nt}^\top \boldsymbol{\gamma}_{klt})}{\sum_{h=1}^{L} \exp(\boldsymbol{\zeta}_{nt}^\top \boldsymbol{\gamma}_{kht})}, \qquad \boldsymbol{\gamma}_{k,l_0,t} = 0.$$

Equivalently, for $l \neq l_0$,

$$\log \frac{P(Z_{nt} = l \mid Z_{n,t-1} = k, \boldsymbol{\zeta}_{nt})}{P(Z_{nt} = l_0 \mid Z_{n,t-1} = k, \boldsymbol{\zeta}_{nt})} = \boldsymbol{\zeta}_{nt}^\top \boldsymbol{\gamma}_{klt}.$$

Thus, γ_{klt0} is the intercept for the transition from origin state k to destination state l, relative to destination l_0; γ_{kltu} is the corresponding change in log relative transition odds for a one-unit increase in covariate u.

When `covariates = NULL`, every $\boldsymbol{\zeta}_{nt}$ contains only the intercept, so $\mathbf{P}_{nt}$ is a population transition matrix rather than a participant-specific matrix. When `covariates.time.cross = TRUE`, $\boldsymbol{\gamma}_{kl2} = \cdots = \boldsymbol{\gamma}_{klT}$; the coefficient structure is time-invariant, although transition probabilities may still differ between participants or times when their covariate values differ.

Conditional on the covariates, the probability of one latent-state path is

$$\begin{aligned} &P(Z_{n1} = z_1, \ldots, Z_{nT} = z_T \\ &\quad \mid \boldsymbol{\zeta}_{n1}, \ldots, \boldsymbol{\zeta}_{nT}) \\ &= P(Z_{n1} = z_1 \mid \boldsymbol{\zeta}_{n1}) \\ &\quad \times \prod_{t=2}^{T} P(Z_{nt} = z_t \mid Z_{n,t-1} = z_{t-1}, \boldsymbol{\zeta}_{nt}). \end{aligned}$$

If the latent states were observed, the transition-model log-likelihood would be

$$\begin{aligned}\ell_{\text{path}} = &\sum_{n=1}^{N}\sum_{l=1}^{L} \mathbb{1}(Z_{n1} = l)\log \pi_{n1l} \\ &+ \sum_{n=1}^{N}\sum_{t=2}^{T}\sum_{k=1}^{L}\sum_{l=1}^{L} \mathbb{1}(Z_{n,t-1} = k, Z_{nt} = l) \\ &\quad \times \log P(Z_{nt} = l \mid Z_{n,t-1} = k, \boldsymbol{\zeta}_{nt}).\end{aligned}$$

Corrected observed transition likelihood

The measurement and classification stages follow the same three-step principles as `LCPA()`. A common unconditional `LCA()` or `LPA()` measurement model is fitted at the first time point or to the pooled time points according to `step1.pool`; `control.model$params` may instead supply fixed common measurement parameters. This assumes measurement invariance over the analyzed times. At each time, posterior classification produces $\widehat{Z}_{nt}$ and an $L \times L$ matrix with entries $\text{CEP}_t(l, k) = P(\widehat{Z}_{nt} = k \mid Z_{nt} = l)$. See `get.CEP()` for its estimator and orientation. This longitudinal three-step construction follows the latent Markov/LTA developments of Bartolucci, Montanari, and Pandolfi (2015), Di Mari, Oberski, and Vermunt (2016), and Liang, de la Torre, and Law (2023).

The bias-adjusted ML approach of Di Mari et al. (2016) combines those classification-error probabilities with the complete transition model and sums over every possible latent-state path:

$$\begin{aligned}\ell_{\text{ML}}(\boldsymbol{\beta}, \boldsymbol{\gamma}) = \sum_{n=1}^{N} \log \Bigg[&\sum_{z_1=1}^{L} \cdots \sum_{z_T=1}^{L} \text{CEP}_1(z_1, \widehat{Z}_{n1}) \\ &\times P(Z_{n1} = z_1 \mid \boldsymbol{\zeta}_{n1}) \\ &\times \prod_{t=2}^{T} \Big\{ \text{CEP}_t(z_t, \widehat{Z}_{nt}) \\ &\quad \times P(Z_{nt} = z_t \mid Z_{n,t-1} = z_{t-1}, \boldsymbol{\zeta}_{nt}) \Big\} \Bigg].\end{aligned}$$

The CEP term is the conditional probability of the observed modal state given the corresponding latent state. With `CEP.error = FALSE`, every CEP matrix is the identity matrix and the likelihood reduces to the likelihood of the modal state sequence.

The likelihood is evaluated by the forward recursion

$$\alpha_{n1}(l) = \text{CEP}_1(l, \widehat{Z}_{n1})\pi_{n1l},$$

$$\alpha_{nt}(l) = \text{CEP}_t(l, \widehat{Z}_{nt}) \sum_{k=1}^{L} \alpha_{n,t-1}(k) P(Z_{nt} = l \mid Z_{n,t-1} = k, \boldsymbol{\zeta}_{nt}),$$

so that

$$\ell_{\text{ML}} = \sum_{n=1}^{N} \log \left\{ \sum_{l=1}^{L} \alpha_{nT}(l) \right\}.$$

A backward recursion supplies the posterior initial-state and adjacent-state probabilities used by the analytic score.

Optimization of the transition model

The free parameter vector contains $\boldsymbol{\beta}_l$ and $\boldsymbol{\gamma}_{klt}$ for $l \neq l_0$; all reference-state coefficient vectors are fixed at zero. ML minimizes $-\ell_{\mathrm{ML}}(\boldsymbol{\beta}, \boldsymbol{\gamma})$ subject to `lower` and `upper`. All latent-state posterior probabilities in the score below are conditional on the complete modal-state sequence and the supplied covariates. For $l \neq l_0$, the first derivative with respect to the initial-state coefficient vector $\boldsymbol{\beta}_l$ is

$$\frac{\partial \ell_{\mathrm{ML}}(\boldsymbol{\beta}, \boldsymbol{\gamma})}{\partial \boldsymbol{\beta}_l} = \sum_{n=1}^{N} \boldsymbol{\zeta}_{n1} \left\{ P(Z_{n1} = l \mid \widehat{Z}_{n1}, \ldots, \widehat{Z}_{nT}) - \pi_{n1l} \right\}.$$

This is an $(U_1 + 1) \times 1$ derivative vector whose entries correspond to the intercept and the U_1 covariate coefficients in $\boldsymbol{\beta}_l$. Let $\xi_{ntkl} = P(Z_{n,t-1} = k, Z_{nt} = l \mid \widehat{Z}_{n1}, \ldots, \widehat{Z}_{nT})$. Here k indexes the origin state at time $t-1$, l indexes the destination state at time t, and t identifies the transition from time $t-1$ to time t. The first derivative with respect to $\boldsymbol{\gamma}_{klt}$ is

$$\frac{\partial \ell_{\mathrm{ML}}(\boldsymbol{\beta}, \boldsymbol{\gamma})}{\partial \boldsymbol{\gamma}_{klt}} = \sum_{n=1}^{N} \boldsymbol{\zeta}_{nt} \left\{ \xi_{ntkl} - P(Z_{nt} = l \mid Z_{n,t-1} = k, \boldsymbol{\zeta}_{nt}) \sum_{h=1}^{L} \xi_{ntkh} \right\}, \qquad l \neq l_0.$$

This is an $(U_t + 1) \times 1$ derivative vector whose entries correspond to the intercept and the U_t covariate coefficients in $\boldsymbol{\gamma}_{klt}$. At an interior maximum, these derivatives and the derivatives with respect to $\boldsymbol{\beta}_l$ jointly equal $\mathbf{0}$. If `covariates.time.cross = TRUE`, the derivatives are summed over $t = 2, \ldots, T$ for the transition coefficient vector shared across time. With `method.regression = "Analytic"`, the exact forward–backward score is supplied to box-constrained L-BFGS optimization. With `"Numeric"`, the same corrected log-likelihood is optimized using its numerical gradient. `tol` and `maxiter` control this Step 3 optimization.

With `method.3step = "BCH"`, the modified inverse-CEP weighting of Vermunt (2010) and Bakk, Tekle, and Vermunt (2013) extends to multiple latent states when the time-specific classification errors are conditionally independent given the latent-state path. Bakk, Tekle, and Vermunt (2013) describe the multiple-latent-variable extension, and Nylund-Gibson, Arch, and Carter (2026) apply BCH to auxiliary-variable analysis in LTA. The resulting criterion is

$$\begin{aligned}\ell_{\mathrm{BCH}} = &\sum_{n=1}^{N} \sum_{l=1}^{L} (\mathrm{CEP}_1^{-1})_{\widehat{Z}_{n1},l} \log P(Z_{n1} = l \mid \boldsymbol{\zeta}_{n1}) \\ &+ \sum_{n=1}^{N} \sum_{t=2}^{T} \sum_{k=1}^{L} \sum_{l=1}^{L} (\mathrm{CEP}_{t-1}^{-1})_{\widehat{Z}_{n,t-1},k} (\mathrm{CEP}_t^{-1})_{\widehat{Z}_{nt},l} \log P(Z_{nt} = l \mid Z_{n,t-1} = k, \boldsymbol{\zeta}_{nt}).\end{aligned}$$

It decomposes into one initial-state multinomial model and origin-state-specific transition multinomial models. Analytic estimation supplies their exact scores to L-BFGS-B; numeric estimation optimizes the same criterion without a supplied score. The adjacent-state BCH correction assumes that classification errors are conditionally independent across times given the latent-state path.

Dependent variables at selected times

`type.analysis = "ZY"` relates the external observed $\mathbf{Y}_{nt}$ to the latent states without treating any Y_{ntv} as an indicator in $\mathbf{X}_{nt}$. The Gaussian and categorical conditional distributions have the same definitions as in `LCPA()`; see Nylund-Gibson et al. (2014), Asparouhov and Muthén (2014a), and Nylund-Gibson, Arch, and Carter (2026) for longitudinal auxiliary-variable models. The longitudinal model offers the following two structures. Here t denotes a selected dependent-variable

time, $r = 1, 2, \ldots, T$ indexes all measurement times, and q indexes the observed categories of a categorical dependent variable.

With `dependent.variable.structure = "State"`, each selected Y_{ntv} is conditional only on the latent state at the same selected time. Then

$$Y_{ntv} \mid Z_{nt} = l \sim \mathcal{N}(\mu_{tlv}, \sigma^2_{tlv})$$

for a Gaussian dependent variable, or

$$P(Y_{ntv} = q \mid Z_{nt} = l) = p_{tlvq}$$

for a categorical dependent variable. Following the manual ML distal-outcome formulation of Bakk, Tekle, and Vermunt (2013) and Nylund-Gibson, Grimm, and Masyn (2019), the LTA State model sums over all complete latent-state paths while constraining paths with the same state at selected time t to share the same dependent-variable parameters. For a Gaussian dependent variable,

$$\begin{aligned}\ell_{\mathrm{ML},tv} = \sum_{n=1}^{N} \log \Bigg[\sum_{z_1=1}^{L} \cdots \sum_{z_T=1}^{L} \left\{ \prod_{r=1}^{T} \mathrm{CEP}_r(z_r, \widehat{Z}_{nr}) \right\} \\ \times P(Z_{n1} = z_1) \prod_{r=2}^{T} P(Z_{nr} = z_r \mid Z_{n,r-1} = z_{r-1}) \\ \times \mathcal{N}(Y_{ntv} \mid \mu_{t,z_t,v}, \sigma^2_{t,z_t,v}) \Bigg].\end{aligned}$$

For a categorical dependent variable, the normal density is replaced by $\prod_q p_{t,z_t,vq}^{\mathbb{1}(Y_{ntv}=q)}$. Selecting $t = T$ therefore estimates the commonly used final-state model $\mathbf{Y}_{nT} \mid Z_{nT}$.

With `dependent.variable.structure = "Path"`, each dependent-variable distribution is conditional on the complete latent-state path, where every $z_r \in \{1, 2, \ldots, L\}$:

$$Y_{ntv} \mid Z_{n1} = z_1, \ldots, Z_{nT} = z_T \sim \mathcal{N}(\mu_{t,z_1 \ldots z_T,v}, \sigma^2_{t,z_1 \ldots z_T,v}),$$

with an analogous path-specific categorical probability $p_{t,z_1 \ldots z_T,vq}$. This gives L^T dependent-variable distributions for all latent-state combinations or transition patterns. Nylund-Gibson, Arch, and Carter (2026) describe distal analyses by transition pattern in LTA, and Bakk, Tekle, and Vermunt (2013) provide the multiple-latent-variable correction underlying this construction. ML estimates an intercept-only initial-state distribution, time-specific first-order transition matrices, and the path-specific dependent-variable distributions jointly. For a Gaussian dependent variable,

$$\begin{aligned}\ell_{\mathrm{ML},tv} = \sum_{n=1}^{N} \log \Bigg[\sum_{z_1=1}^{L} \cdots \sum_{z_T=1}^{L} \left\{ \prod_{r=1}^{T} \mathrm{CEP}_r(z_r, \widehat{Z}_{nr}) \right\} \\ \times P(Z_{n1} = z_1) \prod_{r=2}^{T} P(Z_{nr} = z_r \mid Z_{n,r-1} = z_{r-1}) \\ \times \mathcal{N}(Y_{ntv} \mid \mu_{t,z_1 \ldots z_T,v}, \sigma^2_{t,z_1 \ldots z_T,v}) \Bigg].\end{aligned}$$

For a categorical dependent variable, the final normal density is replaced by $\prod_q p_{t,z_1 \ldots z_T,vq}^{\mathbb{1}(Y_{ntv}=q)}$.

BCH uses the inverse CEP entries directly. In the State model, summing the complete-path BCH weights over every state other than the state at selected time t gives the selected-time equations used in the BCH distal-dependent-variable construction of Asparouhov and Muthén (2014b). The Gaussian mean and variance satisfy

$$\sum_{n=1}^{N}(\mathrm{CEP}_t^{-1})_{\widehat{Z}_{nt},l}(Y_{ntv}-\mu_{tlv})=0,$$

$$\sum_{n=1}^{N}(\mathrm{CEP}_t^{-1})_{\widehat{Z}_{nt},l}\{(Y_{ntv}-\mu_{tlv})^2-\sigma_{tlv}^2\}=0,$$

and the categorical probability satisfies

$$\sum_{n=1}^{N}(\mathrm{CEP}_t^{-1})_{\widehat{Z}_{nt},l}\{\mathbb{1}(Y_{ntv}=q)-p_{tlvq}\}=0.$$

In the Path model, the corresponding equations are

$$\sum_{n=1}^{N}\left\{\prod_{r=1}^{T}(\mathrm{CEP}_r^{-1})_{\widehat{Z}_{nr},z_r}\right\}(Y_{ntv}-\mu_{t,z_1\ldots z_T,v})=0,$$

$$\sum_{n=1}^{N}\left\{\prod_{r=1}^{T}(\mathrm{CEP}_r^{-1})_{\widehat{Z}_{nr},z_r}\right\}\{(Y_{ntv}-\mu_{t,z_1\ldots z_T,v})^2-\sigma_{t,z_1\ldots z_T,v}^2\}=0,$$

$$\sum_{n=1}^{N}\left\{\prod_{r=1}^{T}(\mathrm{CEP}_r^{-1})_{\widehat{Z}_{nr},z_r}\right\}\{\mathbb{1}(Y_{ntv}=q)-p_{t,z_1\ldots z_T,vq}\}=0.$$

The products across time assume conditionally independent classification errors given the latent-state path. When `dependent.variable.time.cross = TRUE`, the selected times share the same state- or path-specific dependent-variable parameters.

Uncertainty estimation

For ML-XZ and BCH-XZ, `method.SE = "Analytic"` uses the analytic information or estimating-equation derivative with participant-level score contributions. `"Numeric"` evaluates the corresponding Hessian or Jacobian numerically. `"Bootstrap"` resamples participants jointly across all times, recalculates the posterior classifications and CEP matrices, and re-estimates the initial-state and transition model while holding the common measurement parameters fixed. The ML-XZ covariance also includes the sampling influence of the estimated CEP matrices. State-ZY uncertainty estimation follows `LCPA()`. Path-ML uses Louis observed information for the jointly estimated initial-state, transition, and dependent-variable parameters. State-ML imposes equality of the dependent-variable parameters across complete paths sharing the selected state; Path-ML leaves them path-specific. Path-BCH uses the estimating-equation sandwich formed from the complete-path BCH products. The ZY bootstrap resamples each participant's complete longitudinal record. Gaussian ZY models report the state- or path-specific conditional means and variances, their respective standard errors and covariance matrices, and separate omnibus Wald tests for equality of means and variances. Categorical ZY models report every state- or path-specific category probability, its standard error and covariance matrix, and an omnibus Wald test of equality of the complete conditional category distributions.

References

Asparouhov, T., & Muthén, B. (2014a). Auxiliary variables in mixture modeling: Three-step approaches using Mplus. *Structural Equation Modeling: A Multidisciplinary Journal, 21*(3), 329–341. doi:10.1080/10705511.2014.915181

Asparouhov, T., & Muthén, B. (2014b). *Auxiliary variables in mixture modeling: Using the BCH method in Mplus to estimate a distal outcome model and an arbitrary secondary model* (Mplus Web Note No. 21, Version 2). https://www.statmodel.com/examples/webnotes/webnote21.pdf

Bakk, Z., Tekle, F. B., & Vermunt, J. K. (2013). Estimating the association between latent class membership and external variables using bias-adjusted three-step approaches. *Sociological Methodology, 43*(1), 272–311. doi:10.1177/0081175012470644

Bartolucci, F., Montanari, G. E., & Pandolfi, S. (2015). Three-step estimation of latent Markov models with covariates. *Computational Statistics & Data Analysis, 83*, 287–301. doi:10.1016/j.csda.2014.10.017

Di Mari, R., Oberski, D. L., & Vermunt, J. K. (2016). Bias-adjusted three-step latent Markov modeling with covariates. *Structural Equation Modeling: A Multidisciplinary Journal, 23*(5), 649–660. doi:10.1080/10705511.2016.1191015

Liang, Q., de la Torre, J., & Law, N. (2023). Latent transition cognitive diagnosis model with covariates: A three-step approach. *Journal of Educational and Behavioral Statistics, 48*(6), 690–718. doi:10.3102/10769986231163320

Nylund-Gibson, K., Grimm, R., Quirk, M., & Furlong, M. (2014). A latent transition mixture model using the three-step specification. *Structural Equation Modeling: A Multidisciplinary Journal, 21*(3), 439–454. doi:10.1080/10705511.2014.915375

Nylund-Gibson, K., Grimm, R. P., & Masyn, K. E. (2019). Prediction from latent classes: A demonstration of different approaches to include distal outcomes in mixture models. *Structural Equation Modeling: A Multidisciplinary Journal, 26*(6), 967–985. doi:10.1080/10705511.2019.1590146

Nylund-Gibson, K., Arch, D. A. N., & Carter, D. (2026). Latent transition analysis with auxiliary variables: A demonstration of the ML 3-step and BCH in Mplus. *The Quantitative Methods for Psychology, 22*(1), 1–8. doi:10.20982/tqmp.22.1.p001

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

See Also

LCA(), LPA(), LCPA(), get.CEP()

Examples

```
library(LCPA)

set.seed(1245)
N <- 2000
L <- 3
I <- 6
times <- 2

# Two observed covariates plus the required intercept at both times
```

```
zeta.1 <- as.numeric(scale(rnorm(N)))
zeta.2 <- rbinom(N, 1, 0.5)
covariates <- list(
  cbind(Intercept = 1, Zeta.1 = zeta.1, Zeta.2 = zeta.2),
  cbind(Intercept = 1, Zeta.1 = zeta.1, Zeta.2 = zeta.2)
)
beta <- matrix(c(
   0.70,  0.30, 0,
   0.40, -0.20, 0,
  -0.30,  0.30, 0
), ncol = L, byrow = TRUE)
rownames(beta) <- colnames(covariates[[1]])
gamma <- list(list(
  list(c( 0.90,  0.25, -0.15), c( 0.20, -0.15,  0.20), c(0, 0, 0)),
  list(c( 0.10,  0.20,  0.10), c( 0.80, -0.20, -0.15), c(0, 0, 0)),
  list(c(-0.10,  0.15, -0.10), c( 0.20,  0.10,  0.15), c(0, 0, 0))
))

data.LTA <- sim.LTA(
  N = N, I = I, L = L, times = times, type = "LCA",
  poly.value = 2, IQ = 0.95, distribution = "uniform",
  covariates = covariates, ref.class = 3,
  beta = beta, gamma = gamma, is.sort = TRUE
)
control.model <- list(
  is.sort = TRUE, starts = 10, maxiter.warmup = 10, nrep = 3
)

# Covariates predicting initial states and transitions: XZ analysis
fit.LTA.XZ <- LTA(
  responses = data.LTA$responses, L = L,
  type.analysis = "XZ", type.model = "LCA",
  covariates = covariates, ref.class = 3,
  method.model = "EM", control.model = control.model,
  method.3step = "ML", method.regression = "Analytic",
  method.SE = "Analytic", maxiter = 500, vis = TRUE
)
round(cbind(
  "True Class 1" = beta[, 1],
  "Estimate Class 1" = fit.LTA.XZ$beta[, 1],
  "True Class 2" = beta[, 2],
  "Estimate Class 2" = fit.LTA.XZ$beta[, 2]
), 3)
gamma.matrix <- function(value) {
  do.call(rbind, lapply(seq_len(L), function(k) {
    do.call(rbind, value[[1]][[k]][seq_len(L - 1)])
  }))
}
gamma.true <- gamma.matrix(gamma)
gamma.estimate <- gamma.matrix(fit.LTA.XZ$gamma)
rownames(gamma.true) <- rownames(gamma.estimate) <- paste0(
  "Origin.", rep(seq_len(L), each = L - 1),
  ".To.", rep(seq_len(L - 1), times = L)
```

```
)
colnames(gamma.true) <- paste0("True.", colnames(covariates[[2]]))
colnames(gamma.estimate) <- paste0("Estimate.", colnames(covariates[[2]]))
round(cbind(gamma.true, gamma.estimate), 3)

# Final latent state predicting two dependent variables: ZY-State
true.state.mean <- rbind(
  "Class 1" = c(Depression = 8, Anxiety = 12),
  "Class 2" = c(Depression = 10, Anxiety = 10),
  "Class 3" = c(Depression = 13, Anxiety = 8)
)
dependent.variables <- scale(
  true.state.mean[data.LTA$Zs[[times]], ] +
    matrix(rnorm(N * 2, sd = 1.5), N, 2)
)
true.state.mean.standardized <- sweep(
  sweep(true.state.mean, 2, attr(dependent.variables, "scaled:center"), "-"),
  2, attr(dependent.variables, "scaled:scale"), "/"
)
true.state.variance.standardized <-
  (1.5 / attr(dependent.variables, "scaled:scale"))^2
dependent.variables <- as.data.frame(dependent.variables)
fit.LTA.ZY <- LTA(
  responses = data.LTA$responses, L = L,
  type.analysis = "ZY", type.model = "LCA",
  dependent.variables = dependent.variables,
  family = "gaussian", dependent.variable.structure = "State",
  dependent.variable.time = times,
  method.model = "EM", control.model = control.model,
  method.3step = "BCH", method.regression = "Analytic",
  method.SE = "Analytic", vis = TRUE
)
round(cbind(
  True.Depression = true.state.mean.standardized[, "Depression"],
  Estimate.Depression =
    fit.LTA.ZY$dependent.variables$t2$Depression$estimate,
  True.Anxiety = true.state.mean.standardized[, "Anxiety"],
  Estimate.Anxiety =
    fit.LTA.ZY$dependent.variables$t2$Anxiety$estimate
), 3)
round(cbind(
  True.Variance.Depression = rep(
    true.state.variance.standardized["Depression"], L
  ),
  Estimate.Variance.Depression =
    fit.LTA.ZY$dependent.variables$t2$Depression$variance,
  True.Variance.Anxiety = rep(
    true.state.variance.standardized["Anxiety"], L
  ),
  Estimate.Variance.Anxiety =
    fit.LTA.ZY$dependent.variables$t2$Anxiety$variance
), 3)
```

```
# Complete latent-state paths predicting two dependent variables: ZY-Path
paths <- expand.grid(t1 = seq_len(L), t2 = seq_len(L))
true.path.mean <- cbind(
  Wellbeing = 16 - paths$t1 - 2 * paths$t2,
  Stress = 5 + paths$t2 + 1.5 * (paths$t1 != paths$t2)
)
rownames(true.path.mean) <- paste("Path", paste(paths$t1, paths$t2, sep = "-"))
path.index <- data.LTA$Zs[[1]] + L * (data.LTA$Zs[[2]] - 1L)
dependent.variables.path <- scale(
  true.path.mean[path.index, ] +
    matrix(rnorm(N * 2, sd = 1.5), N, 2)
)
true.path.mean.standardized <- sweep(
  sweep(true.path.mean, 2,
    attr(dependent.variables.path, "scaled:center"), "-"),
  2, attr(dependent.variables.path, "scaled:scale"), "/"
)
true.path.variance.standardized <-
  (1.5 / attr(dependent.variables.path, "scaled:scale"))^2
dependent.variables.path <- as.data.frame(dependent.variables.path)
fit.LTA.ZY.path <- LTA(
  responses = data.LTA$responses, L = L,
  type.analysis = "ZY", type.model = "LCA",
  dependent.variables = dependent.variables.path,
  family = "gaussian", dependent.variable.structure = "Path",
  dependent.variable.time = times,
  method.model = "EM", control.model = control.model,
  method.3step = "BCH", method.regression = "Analytic",
  method.SE = "Analytic", vis = TRUE
)
round(cbind(
  True.Wellbeing = true.path.mean.standardized[, "Wellbeing"],
  Estimate.Wellbeing =
    fit.LTA.ZY.path$dependent.variables$t2$Wellbeing$estimate,
  True.Stress = true.path.mean.standardized[, "Stress"],
  Estimate.Stress =
    fit.LTA.ZY.path$dependent.variables$t2$Stress$estimate
), 3)
round(cbind(
  True.Variance.Wellbeing = rep(
    true.path.variance.standardized["Wellbeing"], L^times
  ),
  Estimate.Variance.Wellbeing =
    fit.LTA.ZY.path$dependent.variables$t2$Wellbeing$variance,
  True.Variance.Stress = rep(
    true.path.variance.standardized["Stress"], L^times
  ),
  Estimate.Variance.Stress =
    fit.LTA.ZY.path$dependent.variables$t2$Stress$variance
), 3)
```

`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, y = NULL, ncol = 2, ...)

## S3 method for class 'LPA'
plot(x, y = NULL, ncol = 2, ...)
```

Arguments

x	An object of one of the following classes: • Model objects: LCA, LPA.
y	Reserved for compatibility with the base plot generic. It must be NULL; specify ncol by name.
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 plot.LCA and 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, and summary.SE
...	Reserved for S3 method compatibility; no additional arguments are used.
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, print.summary.sim.LPA, print.summary.sim.LTA, and 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, print.summary.sim.LPA, and print.summary.sim.LTA.
L.max	Maximum number of latent classes/profiles to display before truncation (default: varies, e.g., 3). Used by print.SE and 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)
- Likelihood fit statistics for models for which they are defined
- Class/profile prior probabilities and frequencies
- Item-response probabilities (LCA) or profile means (LPA)
- For XZ: regression coefficients with significance markers and 95% CIs
- For Gaussian ZY: conditional means and variances with standard errors, 95% CIs, and omnibus equality tests
- For categorical ZY: conditional category probabilities with standard errors, 95% CIs, and an omnibus distributional test
- 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 are preferred for information criteria
- 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())
- Method-specific diagnostics

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.

`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 $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_K)$ on the unit simplex, where $\sum_{q=1}^{K} \omega_q = 1$ and $\omega_q \geq 0$, the density is

$$f(\boldsymbol{\omega};\boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})}\prod_{q=1}^{K}\omega_q^{\alpha_q-1}.$$

where the normalizing constant $B(\alpha)$ is the multivariate beta function:

$$B(\boldsymbol{\alpha}) = \frac{\prod_{q=1}^{K}\Gamma(\alpha_q)}{\Gamma(\sum_{q=1}^{K}\alpha_q)}.$$

Simulation method:

The function uses independent Gamma random variables $G_q \sim \mathrm{Gamma}(\mathrm{shape}=\alpha_q, \mathrm{rate}=1)$:

$$\omega_q = \frac{G_q}{\sum_{h=1}^{K} G_h}, \qquad q = 1,\ldots,K.$$

The resulting vector $\boldsymbol{\omega}$ follows a Dirichlet distribution with parameter vector $\boldsymbol{\alpha}$.

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,
  maxattempts = 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.
maxattempts	Integer. Maximum number of matrix draws 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 (when `positive = TRUE`): The returned matrix is positive definite.
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

The theoretical condition $\eta > (I-2)/2$ is recommended for positive-definite matrices in high dimensions.

References

Joe, H., & Kurowicka, D. (2026). Random correlation matrices generated via partial correlation C-vines. *Journal of Multivariate Analysis, 211*, 105519. doi: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)  # > 0

# 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 in the final class order. P.Z Vector of length L with latent class prior probabilities in the final class order. Z Vector of length N containing fixed class assignments in the final class order. When supplied, Z takes precedence over P.Z.
is.sort	A logical value. If TRUE (default), internally generated class probabilities are ordered decreasingly before class-specific parameters and observations are generated. Supplied parameters are already defined in this final order and are never reordered; supplied P.Z or Z must therefore follow decreasing class proportions.

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 $P(\text{category} \leq k - 1) < u \leq P(\text{category} \leq k)$
- 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: `"Class 1"`, `"Class 2"`, and so on for classes; `"I1"`, `"I2"`, and so on for indicators; `"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 `"Class 1"`, `"Class 2"`, and so on.

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.

call The matched simulation call.

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. Across the L classes, construct L focal probabilities containing IQ, $1 - IQ$, and, when $L > 2$, values sampled from $[1 - IQ, IQ]$.
2. For each class, assign its focal probability to one category and distribute the remaining mass over the other categories using a symmetric Dirichlet draw.
3. Randomize category positions within each class.

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(-2, 2)`.
`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: `means` $L \times I$ matrix of class-specific means in the final class order. `covs` $I \times I \times L$ array of class-specific covariance matrices in the final class order. `P.Z` Vector of length L with latent class prior probabilities in the final class order.

`Z` Vector of length N containing fixed class assignments in the final class order. When supplied, `Z` takes precedence over `P.Z`.

`is.sort` A logical value. If `TRUE` (default), internally generated class probabilities are ordered decreasingly before class-specific parameters and observations are generated. Supplied parameters are already defined in this final order and are never reordered; supplied `P.Z` or `Z` must therefore follow decreasing class proportions.

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:

- Univariate case (`I=1`): Constraints `"UE"` and `"UV"` are enforced automatically. Predefined constraints like `"E0"` map to `"UE"`.

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 `"V"` for univariate data.

means Numeric matrix ($L \times I$) of true class-specific means. Row names: `"Profile 1"`, `"Profile 2"`, and so on; 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: `"Profile 1"`, `"Profile 2"`, and so on.

P.Z Numeric vector (length L) of true class proportions. Named `"Profile 1"`, `"Profile 2"`, and so on.

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.

call Matched simulation call.

arguments List of the effective simulation arguments.

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, and the returned covariance matrices are positive definite. 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=1
sim_uni <- sim.LPA(N = 200, I = 1, 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,
  times = 2,
  type = "LCA",
  distribution = "random",
  constraint = "VV",
  poly.value = 5,
  IQ = "random",
  mean.range = c(-2, 2),
  covs.range = c(0.01, 4),
  params = NULL,
  is.sort = TRUE,
  rate = NULL,
  covariates = NULL,
  ref.class = L,
  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.
times	Integer; number of time points (must be ≥ 1).
type	Character; type of latent model. "LCA" for categorical indicators (default), "LPA" for continuous indicators.
distribution	Character; distribution of initial class probabilities when not using covariates or params. Options: "uniform" (equal probabilities) or "random" (Dirichlet-distributed, default).
constraint	Character; covariance structure for LPA (type="LPA" only). Options: "UE" and "UV" for univariate responses; "E0", "V0", "EE", "EV", "VE", and "VV" for multivariate responses. See sim.LPA. The default is "VV".
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.
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)).
params	List or NULL; pre-specified measurement and initial-class parameters in the final first-time-point class order (see Details).
is.sort	A logical value. If TRUE (default), internally generated first-time-point classes are established in decreasing model-implied probability order. Supplied parameters already use this final order and are never reordered; a supplied beta, params$P.Z, or params$Z must therefore imply decreasing first-time-point class proportions. If FALSE, the supplied or generated order is retained. No later time point is reordered.
rate	List of matrices or NULL; transition probability matrices in the final class order 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.
covariates	List of matrices or NULL; covariate matrices for each time point. Each matrix must have dimensions $N \times (U_t+1)$ and include an intercept column (first column must be all 1s). If NULL, covariate mode is disabled. See Details for automatic coefficient generation.
ref.class	Integer between 1 and L; reference class in the class order established at the first time point. When is.sort=TRUE, this is the position after ordering the first-time-point classes by decreasing P.Z. The same class order is retained at every later time point.
beta	Matrix or NULL; initial state regression coefficients of dimension $(U_1 + 1) \times L$ in the final first-time-point class order. Column ref.class must be zero. Supplied coefficients determine the logits and are returned unchanged. 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). The supplied coefficients refer to the class order established

at the first time point and are returned unchanged. 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).
- All supplied simulation parameters use the final first-time-point class order. They are never reordered or reparameterized. Internally generated initial coefficients are ordered and reparameterized before they are returned. The same order is used for every transition.

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"`).

`poly.value` Category counts for LCA (if `type="LCA"`).

`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`).

`ref.class` Reference class of the returned `beta` and `gamma` coefficients.

`call` Function call.

`arguments` Input arguments.

Model Specification

Initial Class Probabilities (with covariates): For participant n at time 1, the probability of belonging to latent class l is

$$P(Z_{n1} = l \mid \boldsymbol{\zeta}_{n1}) = \frac{\exp(\boldsymbol{\beta}_l^\top \boldsymbol{\zeta}_{n1})}{\sum_{h=1}^{L} \exp(\boldsymbol{\beta}_h^\top \boldsymbol{\zeta}_{n1})}.$$

Here $\boldsymbol{\zeta}_{n1} = (1, \zeta_{n11}, \ldots, \zeta_{n1U_1})^\top$; the leading 1 is the intercept, and $u = 1, \ldots, U_1$ indexes observed covariates. The coefficient vector $\boldsymbol{\beta}_l = (\beta_{l0}, \beta_{l1}, \ldots, \beta_{lU_1})^\top$ has the corresponding intercept and slopes. The class selected by `ref.class` is the reference class and has a zero coefficient vector.

Transition Probabilities (with covariates and times>=2): For participant n transitioning from class k at time $t-1$ to class l at time t ($t \geq 2$),

$$P(Z_{nt} = l \mid Z_{n,t-1} = k, \boldsymbol{\zeta}_{nt}) = \frac{\exp(\boldsymbol{\gamma}_{klt}^\top \boldsymbol{\zeta}_{nt})}{\sum_{h=1}^{L} \exp(\boldsymbol{\gamma}_{kht}^\top \boldsymbol{\zeta}_{nt})}.$$

Here $\boldsymbol{\zeta}_{nt} = (1, \zeta_{nt1}, \ldots, \zeta_{ntU_t})^\top$, and $\boldsymbol{\gamma}_{klt} = (\gamma_{klt0}, \gamma_{klt1}, \ldots, \gamma_{kltU_t})^\top$ contains the corresponding intercept and slopes. The destination class selected by `ref.class` has a zero coefficient vector for every origin class.

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 $\tau_{kl}^{(t)} = P(Z_{nt} = l \mid Z_{n,t-1} = k)$, where $\sum_{l=1}^{L} \tau_{kl}^{(t)} = 1$ for each origin class k and time 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( 1.3,  0.5, 0.0,
                  0.1,  0.4, 0.0,
                 -0.6, -0.8, 0.0,
                 -0.3, -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",
  ref.class=3,
  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 for `LCA`, `LPA`, `sim.LCA`, `sim.LPA`, and `sim.LTA` summaries (default: 5).
`...`	Reserved for S3 method compatibility; no additional arguments are used.
`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 class-specific list designed both for its corresponding print method and for programmatic access:

`LCA` A `summary.LCA` object with:

- `call` Original fitting call.
- `model.config` Number of latent classes and estimation method.
- `data.info` Sample size, item count, number of categories per item, and whether category counts are uniform.
- `fit.stats` Log-likelihood, AIC, BIC, entropy, and number of free parameters.
- `class.probs` Data frame containing class labels, modal-assignment counts, and estimated class proportions.
- `item.probs` Conditional response-probability matrices for the first `I.max` items.
- `convergence` Backend-specific algorithm, iteration, tolerance, initialization, and diagnostic information.
- `replication` Replication count and best BIC when replication selection applies; otherwise `NULL`.
- `digits`, `I.max.shown`, `total.items` Formatting and truncation metadata.

`LPA` A `summary.LPA` object with:

- `call` Original fitting call.
- `model.config` Number of profiles, requested covariance constraint, its expanded description, and estimation method.
- `data.info` Sample size, variable count, and multivariate-normal distribution label.
- `fit.stats` Log-likelihood, AIC, BIC, entropy, and number of free parameters.
- `class.probs` Data frame containing profile labels, modal-assignment counts, and estimated profile proportions.
- `class.means` Profile-specific means for the first `I.max` variables.
- `convergence` Backend-specific algorithm, iteration, tolerance, initialization, and diagnostic information.
- `replication` Replication count and best BIC when replication selection applies; otherwise `NULL`.
- `digits`, `I.max.shown`, `total.vars` Formatting and truncation metadata.

`LCPA` A `summary.LCPA` object with:

- `call` Original fitting call.
- `model.config` Analysis path, number of classes/profiles, model and three-step methods, Step 1 source, dependent-variable structure where applicable, and classification-error handling.
- `data.info` Sample size and number of response variables.
- `fit.stats` For XZ, log-likelihood, AIC, BIC, and number of free parameters.
- `class.probs` Data frame containing class probabilities, proportions, and modal-assignment frequencies.
- `coefficients` For XZ, the non-reference-class coefficient table with estimates, standard errors, 95 percent confidence limits, z statistics, and two-sided p-values.

dependent.variables For ZY, fitted conditional distributions nested by model and dependent variable. Gaussian entries contain class/profile-specific means and variances, their standard errors and covariance matrices, and separate omnibus Wald tests. Categorical entries contain class/profile-specific category probabilities, standard errors, covariance matrices, and an omnibus test of equality of the conditional distributions.

covariates.names, ref.class Displayed covariate names and the multinomial-logit reference class.

convergence Overall and model-specific Step 3 convergence and iteration information.

digits, vars.to.show, total.vars, has.covariates Formatting and covariate metadata.

LTA A summary.LTA object with:

call Original fitting call.

model.config Number of time points and classes, model type, Step 1 source, reference class, covariate mode, classification-error handling, and transition mode.

data.info Sample size, response-variable count, and number of time points.

fit.stats For XZ, log-likelihood, AIC, BIC, and number of free parameters.

class.probs Time-indexed data frames containing class probabilities, proportions, and modal-assignment frequencies.

initial.model For XZ, the initial-status coefficient table, covariate names, and reference class.

transition.models For XZ, time-invariant or time-indexed transition coefficient tables with origin class, destination class, covariate, estimate, standard error, confidence limits, z statistic, and p-value.

dependent.variables For ZY, state- or path-specific Gaussian means and variances or categorical probabilities, together with their standard errors, covariance matrices, confidence-interval inputs, group masses, and omnibus Wald tests.

convergence Overall and model-specific Step 3 convergence and iteration information.

digits, total.vars, covariates.time.cross, ref.class Formatting, covariate, and reference-class metadata.

sim.LCA A summary.sim.LCA object with:

call Original simulation call.

config Sample size, item count, class count, category counts, category-count uniformity, item quality, and generating distribution.

class.probs True class probabilities and realized frequencies.

item.probs True conditional response probabilities for the first I.max items.

digits, I.max.shown, total.vars Formatting and truncation metadata.

sim.LPA A summary.sim.LPA object with:

call Original simulation call.

config Sample size, variable count, profile count, constraint specification and description, and generating distribution.

class.probs True profile probabilities and realized frequencies.

class.means True profile means for the first I.max variables.

constraint Expanded description of the covariance constraint.

digits, I.max.shown, total.vars Formatting and truncation metadata.

`sim.LTA` A `summary.sim.LTA` object with:

- `call` Original simulation call.
- `config` Sample size, variable count, class count, time points, model type, generating distribution, coefficient reference class, and LPA constraint when applicable.
- `class.probs` Time-indexed true class probabilities and realized frequencies.
- `item.probs`, `class.means` Truncated true measurement parameters for LCA or LPA simulations, respectively.
- `transition` Fixed-rate or covariate-dependent transition specification, including beta/gamma parameters and time indices when present.
- `covariates` Time-indexed covariate summaries containing minima, maxima, and means, or `NULL`.
- `digits`, `I.max.shown`, `L.max.shown`, `total.vars`, `total.classes` Formatting and truncation metadata.

`fit.index` A `summary.fit.index` object with:

- `call` Call that produced the fit-index object.
- `data.info` List containing the sample size `N`.
- `fit.table` Data frame with `Statistic`, `Value`, and `Description` columns for `npar`, `Log.Lik`, `-2LL`, AIC, BIC, SIC, CAIC, AWE, and SABIC.
- `digits` Requested numeric precision.

`compare.model` A `summary.compare.model` object with:

- `call` Call that produced the model comparison.
- `data.info` Lists the named model-specific sample sizes and indicator counts and the two class counts.
- `fit.table` Side-by-side table of class count, parameter count, log-likelihood, -2LL, AIC, BIC, SIC, CAIC, AWE, and SABIC.
- `model.comparison` Data frame comparing class counts, parameter counts, diagonal average posterior probabilities, and entropy.
- `BF`, `BF.interpretation` Bayes factor computed from SIC and its evidence label.
- `LRT.table` Separate rows for the standard LRT, VLMR, adjusted LMR, and BLRT when available, with statistics, degrees of freedom, p-values, and significance symbols.
- `LRT.objects` Named list containing the unmodified hypothesis-test objects used to build `LRT.table`.
- `digits` Requested numeric precision.

`SE` A `summary.SE` object with:

- `call` Call that produced the standard-error object.
- `method` Selected `"Bootstrap"`, `"Obs"`, or `"Louis"` method.
- `diagnostics` Complete method-specific diagnostic list from `get.SE()`.
- `type` `"LCA"`, `"LPA"`, or `"Unknown"`, inferred from the standard-error components.
- `L`, `I` Number of classes/profiles and variables/items.
- `nonzero.counts` Counts of nonzero standard errors for `P.Z` and, as applicable, `par`, `means`, and `covs`.
- `total.P.Z` Total number of class-proportion standard errors.

Value

A structured list whose S3 class identifies the corresponding summary method, such as summary.LCA or summary.LPA. Every summary method returns its object visibly, so an interactive call to summary() dispatches automatically to the corresponding print method.

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(object, ...)

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

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

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

## S3 method for class 'LTA'
update(object, ...)
```

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

## S3 method for class 'sim.LPA'
update(object, ...)

## S3 method for class 'sim.LTA'
update(object, ...)
```

Arguments

`object` 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.

`...` Named arguments that override values in the original call. Every argument must be named, unique, and present in the formal arguments of the corresponding constructor. Valid arguments depend on the class of `object`:

`LCA` `response`, `L`, `category.levels`, `method`, `par.ini`, `is.sort`, `starts`, `maxiter.warmup`, `nrep`, `vis`, `control.EM`, `control.Mplus`, `control.NNE`, `control.flexmix`, `control.Rmixmod`, `control.RMixtComp`

`LPA` `response`, `L`, `constraint`, `method`, `par.ini`, `is.sort`, `starts`, `maxiter.warmup`, `nrep`, `vis`, `control.EM`, `control.Mplus`, `control.NNE`, `control.flexmix`, `control.Rmixmod`, `control.RMixtComp`

`LCPA` `response`, `L`, `type.analysis`, `type.model`, `covariates`, `ref.class`, `dependent.variables`, `family`, `method.model`, `control.model`, `control.EM`, `control.Mplus`, `control.NNE`, `control.flexmix`, `control.Rmixmod`, `control.RMixtComp`, `method.3step`, `CEP.error`, `method.regression`, `maxiter`, `tol`, `lower`, `upper`, `method.SE`, `nrep.bootstrap`, `vis`

`LTA` `responses`, `L`, `type.analysis`, `type.model`, `covariates`, `covariates.time.cross`, `ref.class`, `dependent.variables`, `family`, `dependent.variable.structure`, `dependent.variable.time`, `dependent.variable.time.cross`, `method.model`, `step1.pool`, `control.model`, `control.EM`, `control.Mplus`, `control.NNE`, `control.Rmixmod`, `method.3step`, `CEP.error`, `CEP.time.cross`, `method.regression`, `maxiter`, `tol`, `lower`, `upper`, `method.SE`, `nrep.bootstrap`, `vis`

`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`, `distribution`, `constraint`, `poly.value`, `IQ`, `mean.range`, `covs.range`, `params`, `is.sort`, `rate`, `covariates`, `ref.class`, `beta`, `gamma`

Details

Internally, each method extracts the stored `arguments` list from the input object `object`, 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.

Invalid, unnamed, or duplicated arguments are rejected before the constructor is called. This prevents a misspelled option from appearing to update an object while leaving the fitted model unchanged.

Value

An object of the same class as `object`, 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

adjust.model, 4
adjust.response, 5, *25*, *48*

check.response, 7
compare.model, 8, *11*, *101*, *118*

extract, *9*, 10, *102*

fit.index, *14, 15*, *102, 103*, *121, 122*

get.AvePP, 16, *74*, *77*
get.CEP, 18, *28*, *55*, *57*, *60*, *84*, *87*, *91*
get.entropy, 21, *74*, *77*
get.fit.index, *8, 9*, *11*, *14*, 23, *74*, *76*, *101*, *118*
get.Log.Lik.LCA, 24, *47*, *57*
get.Log.Lik.LPA, 25, *57*
get.Log.Lik.LTA, 26
get.npar.LCA, 29, *47*
get.npar.LPA, 30
get.npar.LTA, 32
get.P.Z.Xn.LCA, 34
get.P.Z.Xn.LPA, 35
get.SE, *11*, 37, *101*, *118*, *121*
getwd, *45*, *66*

install_python_dependencies, 38, *43*, *48*, *64*, *69*

Kmeans.LCA, 40
Kmeans.LPA, 41, *65*

LCA, *8, 9*, *11*, *17*, *21*, *23*, *37*, *41*, 42, *47*, *52–55*, *57*, *60*, *81–83*, *87*, *91*, *97*, *101*, *118*, *122–124*
LCPA, *11*, 52, *87, 88*, *90, 91*, *101*, *118*, *122–124*
logit, 62
LPA, *8, 9*, *11*, *17*, *21*, *23*, *37*, *41*, *53–55*, *57*, *60*, 63, *68*, *81–83*, *87*, *91*, *97*, *101*, *118*, *122–124*
LRT.test, *8, 9*, 73, *74*
LRT.test.Bootstrap, *8*, *74*, 75
LRT.test.VLMR, *8, 9*, 78
LTA, *11*, *26*, *28*, *56*, *60*, 80, *101*, *118*, *122–124*

modifyList, *124*

normalize, *41*, *53*, *63*, *81*, 95

plot, 97, *97*
plot.LCA, *97*
plot.LPA, *97*
plotResponse, 98
print, 99, *102*, *122*
print.SE, *101*
print.summary.fit.index, *101*
print.summary.SE, *101*
print.summary.sim.LCA, *101*
print.summary.sim.LPA, *101*
print.summary.sim.LTA, *101*

rdirichlet, 103

scale, *41*, *53*, *63*, *82*
SE, *15*, *102, 103*, *121*
sim.correlation, 104
sim.LCA, *11*, *13*, *76*, *101*, 107, *118*, *122–124*
sim.LPA, *11*, *14*, *76*, *101*, 110, *114*, *118*, *122, 123*
sim.LTA, *11*, *101*, 113, *118*, *122, 123*
solve_LSAP, *4*
summary, *101*, 117, *122*
summary.compare.model, *101*, *103*
summary.fit.index, *101*, *103*, *121*
summary.LCA, *101, 102*, *122*
summary.LCPA, *101, 102*
summary.LPA, *101, 102*, *122*
summary.LTA, *101, 102*
summary.SE, *101*, *121, 122*
summary.sim.LCA, *101, 102*
summary.sim.LPA, *101, 102*
summary.sim.LTA, *101*, *103*

update, 122