Package: medfit (via r-universe)

June 12, 2026

Type Package

Title Infrastructure for Mediation Model Fitting and Extraction

Version 0.3.1

Date 2026-06-11

Description Provides S7-based infrastructure for fitting mediation models, extracting path coefficients, and performing bootstrap inference. Designed as a foundation package for the mediation analysis ecosystem, supporting 'probmed', 'RMediation', and 'medrobust' packages. Implements unified interfaces for model fitting across different engines (currently generalized linear models, with future support for mixed models and Bayesian methods), standardized extraction of mediation paths from various model types, and robust bootstrap inference methods. Mediation methods are described in MacKinnon, Lockwood and Williams (2004) <doi:10.1207/s15327906mbr3901_4>, Preacher and Hayes (2008) <doi:10.3758/brm.40.3.879>, Tofighi and MacKinnon (2011) <doi:10.3758/s13428-011-0076-x>, and VanderWeele (2014) <doi:10.1097/EDE.0000000000000121>.

License GPL (>= 3)

URL https://data-wise.github.io/medfit/, https://github.com/data-wise/medfit

BugReports https://github.com/data-wise/medfit/issues

Depends R (>= 4.1.0)

Imports S7 (>= 0.1.0), stats, methods, checkmate, generics, MASS

Suggests lavaan (>= 0.6-0), sandwich, testthat (>= 3.0.0), tibble

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE, roclets = c(``rd", ``collate", ``namespace"))

Config/testthat/edition 3

Config/Needs/website knitr, rmarkdown, quarto

RoxygenNote 7.3.3

Repository https://data-wise.r-universe.dev

Date/Publication 2026-06-12 03:08:07 UTC

RemoteUrl https://github.com/Data-Wise/medfit

RemoteRef HEAD

RemoteSha 84cf62fc8aa6d1a5d4dab904b3bf4f44d662c563

Contents

`bootstrap_mediation` *Perform Bootstrap Inference for Mediation Statistics*

Description

Conduct bootstrap inference to compute confidence intervals for mediation statistics. Supports parametric, nonparametric, and plugin methods.

Conduct bootstrap inference to compute confidence intervals for mediation statistics. Supports parametric, nonparametric, and plugin methods.

Usage

```
bootstrap_mediation(
  statistic_fn,
  method = c("parametric", "nonparametric", "plugin"),
  mediation_data = NULL,
  data = NULL,
  n_boot = 1000L,
  ci_level = 0.95,
  parallel = FALSE,
  ncores = NULL,
  seed = NULL,
  ...
)

bootstrap_mediation(
  statistic_fn,
  method = c("parametric", "nonparametric", "plugin"),
  mediation_data = NULL,
  data = NULL,
  n_boot = 1000L,
  ci_level = 0.95,
  parallel = FALSE,
  ncores = NULL,
  seed = NULL,
  ...
)
```

Arguments

`statistic_fn`	Function that computes the statistic of interest. • For parametric bootstrap: receives named parameter vector, returns scalar • For nonparametric bootstrap: receives data frame, returns scalar • For plugin: receives named parameter vector, returns scalar
`method`	Character string: bootstrap method. Options: • `"parametric"`: Sample from multivariate normal (fast, assumes normality) • `"nonparametric"`: Resample data and refit (robust, slower) • `"plugin"`: Point estimate only, no CI (fastest)
`mediation_data`	MediationData object (required for parametric/plugin)
`data`	Data frame (required for nonparametric bootstrap)
`n_boot`	Integer: number of bootstrap samples (default: 1000)
`ci_level`	Numeric: confidence level between 0 and 1 (default: 0.95)
`parallel`	Logical: use parallel processing? (default: FALSE)
`ncores`	Integer: number of cores for parallel processing. If NULL, uses `parallel::detectCores() - 1`
`seed`	Integer: random seed for reproducibility (optional but recommended)
`...`	Additional arguments (reserved for future use)

Details

Bootstrap Methods:

Parametric Bootstrap (`method = "parametric"`):

- Samples parameter vectors from $N(\hat{\theta}, \hat{\Sigma})$
- Fast and efficient
- Assumes asymptotic normality of parameters
- Recommended for most applications with n > 50

Nonparametric Bootstrap (`method = "nonparametric"`):

- Resamples observations with replacement
- Refits models for each bootstrap sample
- More robust, no normality assumption
- Computationally intensive
- Use when normality is questionable or n is small

Plugin Estimator (`method = "plugin"`):

- Computes point estimate only
- No confidence interval
- Fastest method
- Use for quick checks or when CI not needed

Parallel Processing:

Set `parallel = TRUE` to use multiple cores:

- Automatically detects available cores
- Falls back to sequential if parallel fails
- Seed handling ensures reproducibility

Reproducibility:

Always set a seed for reproducible results:

```
bootstrap_mediation(..., seed = 12345)
```

Bootstrap Methods:

Parametric Bootstrap (`method = "parametric"`):

- Samples parameter vectors from $N(\hat{\theta}, \hat{\Sigma})$
- Fast and efficient
- Assumes asymptotic normality of parameters
- Recommended for most applications with n > 50
- Requires `mediation_data` argument

Nonparametric Bootstrap (`method = "nonparametric"`):

- Resamples observations with replacement
- Refits models for each bootstrap sample
- More robust, no normality assumption
- Computationally intensive

- Use when normality is questionable or n is small
- Requires `data` argument

Plugin Estimator (`method = "plugin"`):

- Computes point estimate only
- No confidence interval
- Fastest method
- Use for quick checks or when CI not needed
- Requires `mediation_data` argument

Statistic Function:

The `statistic_fn` should be a function that:

- For parametric/plugin: Takes a named numeric vector of parameters
- For nonparametric: Takes a data frame
- Returns a single numeric value

Common statistic functions for indirect effect:

```
# Using parameter names from MediationData
indirect_fn <- function(theta) {
  theta["m_X"] * theta["y_M"]
}
```

Parallel Processing:

Set `parallel = TRUE` to use multiple cores:

- Uses `parallel::mclapply()` on Unix systems
- Falls back to sequential on Windows
- Automatically detects available cores

Reproducibility:

Always set a seed for reproducible results:

```
bootstrap_mediation(..., seed = 12345)
```

Value

A BootstrapResult object containing:

- Point estimate
- Confidence interval bounds
- Bootstrap distribution (for parametric and nonparametric)
- Method used

A BootstrapResult object containing:

- Point estimate
- Confidence interval bounds
- Bootstrap distribution (for parametric and nonparametric)
- Method used

See Also

BootstrapResult, MediationData, `extract_mediation()`

BootstrapResult, MediationData, `fit_mediation()`

Examples

```
## Not run:
# Parametric bootstrap for indirect effect
result <- bootstrap_mediation(
  statistic_fn = function(theta) theta["a"] * theta["b"],
  method = "parametric",
  mediation_data = med_data,
  n_boot = 5000,
  ci_level = 0.95,
  seed = 12345
)

# Nonparametric bootstrap with parallel processing
result <- bootstrap_mediation(
  statistic_fn = function(data) {
    # Refit models and compute statistic
    # ...
  },
  method = "nonparametric",
  data = mydata,
  n_boot = 5000,
  parallel = TRUE,
  seed = 12345
)

# Plugin estimator (no CI)
result <- bootstrap_mediation(
  statistic_fn = function(theta) theta["a"] * theta["b"],
  method = "plugin",
  mediation_data = med_data
)

## End(Not run)

# Generate example data
set.seed(123)
n <- 100
mydata <- data.frame(X = rnorm(n))
mydata$M <- 0.5 * mydata$X + rnorm(n)
mydata$Y <- 0.3 * mydata$X + 0.4 * mydata$M + rnorm(n)

# Fit mediation model
med_data <- fit_mediation(
  formula_y = Y ~ X + M,
  formula_m = M ~ X,
  data = mydata,
  treatment = "X",
```

```
  mediator = "M"
)

# Define indirect effect function
indirect_fn <- function(theta) theta["m_X"] * theta["y_M"]

# Plugin estimator (point estimate only, fastest)
result_plugin <- bootstrap_mediation(
  statistic_fn = indirect_fn,
  method = "plugin",
  mediation_data = med_data
)
print(result_plugin)


# Parametric bootstrap (recommended for most applications)
result <- bootstrap_mediation(
  statistic_fn = indirect_fn,
  method = "parametric",
  mediation_data = med_data,
  n_boot = 1000,
  ci_level = 0.95,
  seed = 12345
)
print(result)

# Nonparametric bootstrap (slower but more robust)
refit_fn <- function(boot_data) {
  fit_m <- lm(M ~ X, data = boot_data)
  fit_y <- lm(Y ~ X + M, data = boot_data)
  unname(coef(fit_m)["X"] * coef(fit_y)["M"])
}

result_np <- bootstrap_mediation(
  statistic_fn = refit_fn,
  method = "nonparametric",
  data = mydata,
  n_boot = 500,
  seed = 12345
)
print(result_np)
```

`BootstrapResult` *BootstrapResult S7 Class*

Description

S7 class containing results from bootstrap inference, including point estimates, confidence intervals, and bootstrap distribution.

Arguments

`estimate`	Numeric scalar: point estimate of the statistic
`ci_lower`	Numeric scalar: lower bound of confidence interval
`ci_upper`	Numeric scalar: upper bound of confidence interval
`ci_level`	Numeric scalar: confidence level (e.g., 0.95 for 95% CI)
`boot_estimates`	Numeric vector: bootstrap distribution of estimates
`n_boot`	Integer scalar: number of bootstrap samples
`method`	Character scalar: bootstrap method ("parametric", "nonparametric", or "plugin")
`call`	Call object or NULL: original function call

Details

This class standardizes bootstrap inference results across different bootstrap methods (parametric, nonparametric, plugin).

The class includes validation to ensure consistency between method type and required fields.

Value

A BootstrapResult S7 object

Examples

```
# Parametric bootstrap result
result <- BootstrapResult(
  estimate = 0.15,
  ci_lower = 0.10,
  ci_upper = 0.20,
  ci_level = 0.95,
  boot_estimates = rnorm(1000, 0.15, 0.02),
  n_boot = 1000L,
  method = "parametric",
  call = NULL
)
```

`decompose` *Four-Way Decomposition of a Mediation Effect*

Description

Return VanderWeele's (2014) four-way decomposition of the total effect for an InteractionMediationData object: controlled direct effect (CDE), reference interaction (INTref), mediated interaction (INTmed), and pure indirect effect (PIE), together with the derived natural direct/indirect and total effects.

Usage

```
decompose(x, ...)
```

Arguments

x	An InteractionMediationData object.
...	Additional arguments (ignored).

Value

A named numeric vector: `c(cde, int_ref, int_med, pie, nde, nie, total)`.

See Also

`nie()`, `nde()`, `te()`

---

extract_mediation *Extract Mediation Structure from Fitted Models*

Description

Generic function to extract mediation structure (a, b, c' paths and variance-covariance matrices) from fitted models. This function provides a unified interface for extracting mediation information from various model types (lm, glm, lavaan, lmer, brms, etc.).

Usage

```
extract_mediation(object, ...)
```

Arguments

object	Fitted model object (lm, glm, lavaan, etc.)
...	Additional arguments passed to methods. Common arguments include: • `treatment`: Character string specifying treatment variable name • `mediator`: Character string specifying mediator variable name • Method-specific arguments (see individual method documentation)

Details

The `extract_mediation()` generic provides methods for different model types:

- **lm/glm**: Extract from linear and generalized linear models
- **lavaan**: Extract from structural equation models
- **lmerMod**: Extract from mixed-effects models (future)
- **brmsfit**: Extract from Bayesian models (future)

Note: OpenMx extraction is planned for a future release.

All methods return a standardized MediationData object that can be used with other medfit functions and dependent packages (probmed, RMediation, medrobust).

Value

A MediationData object containing:

- Path coefficients (a, b, c')
- Full parameter vector and variance-covariance matrix
- Residual variances (for Gaussian models)
- Variable names and metadata
- Original data (if available)

See Also

MediationData, `fit_mediation()`, `bootstrap_mediation()`

Examples

```
# Simulate data with a single mediator (X -> M -> Y)
set.seed(123)
n <- 200
X <- rnorm(n)
M <- 0.5 * X + rnorm(n)
Y <- 0.3 * M + 0.2 * X + rnorm(n)
dat <- data.frame(X = X, M = M, Y = Y)

# Extract the mediation structure from fitted lm models
fit_m <- lm(M ~ X, data = dat)
fit_y <- lm(Y ~ X + M, data = dat)
med_data <- extract_mediation(fit_m, model_y = fit_y,
                              treatment = "X", mediator = "M")
```

---

fit_mediation *Fit Mediation Models*

Description

Fit mediation models using a specified modeling engine. This function provides a convenient formula-based interface for fitting both the mediator and outcome models simultaneously.

Fit mediation models using a specified modeling engine. This function provides a convenient formula-based interface for fitting both the mediator and outcome models simultaneously.

Usage

```
fit_mediation(
  formula_y,
  formula_m,
  data,
  treatment,
  mediator,
  engine = "glm",
  family_y = stats::gaussian(),
  family_m = stats::gaussian(),
  weights = NULL,
  se_type = c("model", "sandwich"),
  ...
)

fit_mediation(
  formula_y,
  formula_m,
  data,
  treatment,
  mediator,
  engine = "glm",
  family_y = stats::gaussian(),
  family_m = stats::gaussian(),
  weights = NULL,
  se_type = c("model", "sandwich"),
  ...
)
```

Arguments

formula_y	Formula for outcome model (e.g., Y ~ X + M + C)
formula_m	Formula for mediator model (e.g., M ~ X + C)
data	Data frame containing all variables
treatment	Character string: name of treatment variable
mediator	Character string: name of mediator variable
engine	Character string: modeling engine to use. Currently supports: • "glm": Generalized linear models (default)
family_y	Family object for outcome model (default: gaussian())
family_m	Family object for mediator model (default: gaussian())
weights	Optional numeric vector of case weights (length nrow(data)), passed to both the mediator and outcome stats::glm() fits. Use for inverse-probability weighting (IPW). NULL (default) fits unweighted.
se_type	Variance-covariance estimator for @vcov: "model" (default, model-based stats::vcov) or "sandwich" (heteroskedasticity-consistent sandwich::vcovHC, type HC3,

recommended for IPW-weighted fits). The `"sandwich"` option requires the suggested **sandwich** package. Applies to the single-mediator path.

... Additional arguments passed to the fitting function

Details

The `fit_mediation()` function fits both the mediator model and outcome model using the specified engine, then extracts the mediation structure using `extract_mediation()`.

Supported Engines:

GLM (`engine = "glm"`):

- Fits models using `stats::glm()`
- Supports all GLM families (gaussian, binomial, poisson, etc.)
- For Gaussian models, extracts residual variances

Future Engines:

- `"lmer"`: Mixed-effects models via lme4
- `"brms"`: Bayesian models via brms

Model Specification:

The formulas should follow standard R formula syntax:

- `formula_m`: Mediator model (e.g., `M ~ X + C1 + C2`)
- `formula_y`: Outcome model (e.g., `Y ~ X + M + C1 + C2`)

The mediator must appear in `formula_y`, and the treatment must appear in both formulas.

Model Specification:

The function fits two models:

1. **Mediator model**: `formula_m` (e.g., `M ~ X + C1 + C2`)
2. **Outcome model**: `formula_y` (e.g., `Y ~ X + M + C1 + C2`)

The treatment variable must appear in both formulas. The mediator variable must appear in the outcome formula but NOT in the mediator formula (as it is the response).

GLM Engine:

When `engine = "glm"` (default):

- Models are fit using `stats::glm()`
- Supports all GLM families (gaussian, binomial, poisson, etc.)
- For Gaussian models, residual standard deviations are extracted
- Non-Gaussian outcomes have `sigma_y = NULL`

Common Family Specifications:

- `gaussian()`: Continuous outcomes (default)
- `binomial()`: Binary outcomes
- `poisson()`: Count outcomes
- `Gamma()`: Positive continuous outcomes

Value

A MediationData object containing the fitted mediation structure

A MediationData object containing the fitted mediation structure

See Also

MediationData, `extract_mediation()`, `bootstrap_mediation()`

MediationData, `extract_mediation()`, `bootstrap_mediation()`

Examples

```
## Not run:
# Fit Gaussian mediation model
med_data <- fit_mediation(
  formula_y = Y ~ X + M + C,
  formula_m = M ~ X + C,
  data = mydata,
  treatment = "X",
  mediator = "M",
  engine = "glm"
)

# Fit with binary outcome
med_data <- fit_mediation(
  formula_y = Y ~ X + M + C,
  formula_m = M ~ X + C,
  data = mydata,
  treatment = "X",
  mediator = "M",
  engine = "glm",
  family_y = binomial()
)

## End(Not run)

# Generate example data
set.seed(123)
n <- 100
mydata <- data.frame(
  X = rnorm(n),
  C = rnorm(n)
)
mydata$M <- 0.5 * mydata$X + 0.2 * mydata$C + rnorm(n)
mydata$Y <- 0.3 * mydata$X + 0.4 * mydata$M + 0.1 * mydata$C + rnorm(n)

# Simple mediation with continuous variables
med_data <- fit_mediation(
  formula_y = Y ~ X + M,
  formula_m = M ~ X,
  data = mydata,
  treatment = "X",
```

```
  mediator = "M"
)
print(med_data)

# With covariates
med_data_cov <- fit_mediation(
  formula_y = Y ~ X + M + C,
  formula_m = M ~ X + C,
  data = mydata,
  treatment = "X",
  mediator = "M"
)


# Binary outcome (takes longer to fit)
mydata$Y_bin <- rbinom(n, 1, plogis(0.3 * mydata$X + 0.4 * mydata$M))
med_data_bin <- fit_mediation(
  formula_y = Y_bin ~ X + M,
  formula_m = M ~ X,
  data = mydata,
  treatment = "X",
  mediator = "M",
  family_y = binomial()
)
```

`InteractionMediationData` *InteractionMediationData: Mediation with Treatment-Mediator Interaction*

Description

S7 class for simple mediation **with a treatment-by-mediator interaction** ($X \to M \to Y$ where the outcome model contains an $X \times M$ term). It carries VanderWeele's (2014) four-way decomposition of the total effect into controlled direct effect (CDE), reference interaction (INTref), mediated interaction (INTmed), and pure indirect effect (PIE):

$$TE = CDE + INTref + INTmed + PIE$$

with $NDE = CDE + INTref$ and $NIE = INTmed + PIE$. medfit computes the decomposition; causal interpretation is the user's responsibility (it requires the four no-unmeasured-confounding assumptions of VanderWeele 2014).

Arguments

`a_path`	Numeric scalar: treatment -> mediator effect (β_1).
`b_path`	Numeric scalar: mediator -> outcome main effect (θ_2).

c_prime	Numeric scalar: treatment -> outcome main effect (θ_1).
interaction	Numeric scalar: treatment x mediator coefficient (θ_3).
cde, int_ref, int_med, pie	Numeric scalars: the four-way components (controlled direct, reference interaction, mediated interaction, pure indirect).
nde, nie, total_effect	Numeric scalars: derived natural direct effect (CDE + INTref), natural indirect effect (INTmed + PIE), and total effect (the sum of all four components).
m_star	Numeric scalar: reference mediator level for the decomposition (default 0).
estimates	Numeric vector of all parameter estimates.
vcov	Square variance-covariance matrix of estimates.
sigma_m	Optional numeric scalar mediator residual SD, or NULL.
sigma_y	Optional numeric scalar outcome residual SD, or NULL.
treatment, mediator, outcome	Single character strings naming the treatment / mediator / outcome.
mediator_predictors, outcome_predictors	Character vectors of predictor names for the mediator and outcome models.
data	Optional data frame, or NULL.
n_obs	Integer number of observations.
converged	Logical convergence flag.
source_package	Character name of the originating package.

Details

Path coefficients follow the outcome model $Y = \theta_0 + \theta_1 X + \theta_2 M + \theta_3 XM + \ldots$ and mediator model $M = \beta_0 + \beta_1 X + \ldots$: a_path = β_1, b_path = θ_2, c_prime = θ_1, interaction = θ_3. With reference level m_star (m^*) the components are $CDE = \theta_1 + \theta_3 m^*$, $INTmed = \theta_3 \beta_1$, and $PIE = \theta_2 \beta_1$. When $\theta_3 = 0$ the decomposition collapses to standard simple mediation (CDE = NDE = θ_1; INTref = INTmed = 0; NIE = PIE = $\theta_2 \beta_1$).

Value

An InteractionMediationData S7 object.

Examples

```
# Hand-built object (theta3 = 0.2 interaction, m* = 0)
imd <- InteractionMediationData(
  a_path = 0.5, b_path = 0.3, c_prime = 0.1, interaction = 0.2,
  cde = 0.1, int_ref = 0.04, int_med = 0.10, pie = 0.15,
  nde = 0.14, nie = 0.25, total_effect = 0.39, m_star = 0,
  estimates = c(a = 0.5, b = 0.3, c_prime = 0.1, theta3 = 0.2),
  vcov = diag(0.01, 4),
  treatment = "X", mediator = "M", outcome = "Y",
  mediator_predictors = "X", outcome_predictors = c("X", "M", "X:M"),
  n_obs = 200L, converged = TRUE, source_package = "medfit"
```

```
)

nie(imd)       # INTmed + PIE = 0.25
decompose(imd) # all four components + derived effects
```

`med` *Simple Mediation Analysis*

Description

A simplified entry point for mediation analysis. Specify the data and variable names, and get results with minimal configuration.

This is the recommended starting point for most mediation analyses. For more control over model specifications, use `fit_mediation()` directly.

Usage

```
med(
  data,
  treatment,
  mediator,
  outcome,
  covariates = NULL,
  boot = FALSE,
  n_boot = 1000L,
  seed = NULL,
  ...
)
```

Arguments

`data`	A data frame containing all variables
`treatment`	Character: name of treatment (exposure) variable
`mediator`	Character: name of mediator variable
`outcome`	Character: name of outcome variable
`covariates`	Character vector: names of covariates to include (optional, default: none)
`boot`	Logical: compute bootstrap confidence intervals? (default: FALSE for speed)
`n_boot`	Integer: number of bootstrap samples (default: 1000)
`seed`	Integer: random seed for reproducibility (optional)
`...`	Additional arguments passed to `fit_mediation()`

Details

`med()` is designed to be the simplest way to run a mediation analysis. It constructs the model formulas automatically from variable names.

Default Behavior:

- Fits Gaussian (continuous) mediator and outcome models
- No covariates unless specified
- No bootstrap unless requested (use `boot = TRUE`)

Accessing Results:

After running `med()`, use:

- `nie(result)`: Natural indirect effect
- `nde(result)`: Natural direct effect
- `te(result)`: Total effect
- `pm(result)`: Proportion mediated
- `quick(result)`: One-line summary
- `summary(result)`: Detailed summary

Value

A MediationData object with mediation results

See Also

`fit_mediation()` for full control, `quick()` for instant summary, `nie()`, `nde()`, `te()`, `pm()` for extracting effects

Examples

```
# Generate example data
set.seed(123)
n <- 200
mydata <- data.frame(
  treatment = rnorm(n),
  covariate = rnorm(n)
)
mydata$mediator <- 0.5 * mydata$treatment + 0.2 * mydata$covariate + rnorm(n)
mydata$outcome <- 0.3 * mydata$treatment + 0.4 * mydata$mediator +
                  0.1 * mydata$covariate + rnorm(n)

# Simple mediation (no covariates)
result <- med(
  data = mydata,
  treatment = "treatment",
  mediator = "mediator",
  outcome = "outcome"
)
print(result)
```

```
# With covariates
result_cov <- med(
  data = mydata,
  treatment = "treatment",
  mediator = "mediator",
  outcome = "outcome",
  covariates = "covariate"
)

# Quick summary
quick(result)


# With bootstrap CI (slower)
result_boot <- med(
  data = mydata,
  treatment = "treatment",
  mediator = "mediator",
  outcome = "outcome",
  boot = TRUE,
  n_boot = 1000,
  seed = 42
)
```

`MediationData` *MediationData S7 Class*

Description

S7 class containing standardized mediation model structure, including path coefficients, parameter estimates, variance-covariance matrix, and metadata.

Arguments

`a_path`	Numeric scalar: effect of treatment on mediator (a path)
`b_path`	Numeric scalar: effect of mediator on outcome (b path)
`c_prime`	Numeric scalar: direct effect of treatment on outcome (c' path)
`estimates`	Numeric vector: all parameter estimates
`vcov`	Numeric matrix: variance-covariance matrix of estimates
`sigma_m`	Numeric scalar or NULL: residual SD for mediator model
`sigma_y`	Numeric scalar or NULL: residual SD for outcome model
`family_m`	GLM `family` object (or NULL): family/link for the mediator model (e.g. `gaussian()`, `binomial()`). Defaults to unset (empty), which consumers treat as Gaussian.

family_y	GLM family object (or NULL): family/link for the outcome model. Required by scale-free estimands (e.g. probmed) to simulate non-Gaussian potential outcomes on the correct scale. Defaults to unset (empty), treated as Gaussian.
treatment	Character scalar: name of treatment variable
mediator	Character scalar: name of mediator variable
outcome	Character scalar: name of outcome variable
mediator_predictors	Character vector: predictor names in mediator model
outcome_predictors	Character vector: predictor names in outcome model
data	Data frame or NULL: original data
n_obs	Integer scalar: number of observations
converged	Logical scalar: whether models converged
source_package	Character scalar: package/engine used for fitting

Details

This class provides a unified container for mediation model information extracted from various model types (lm, glm, lavaan, etc.). It ensures consistency across the mediation analysis ecosystem.

The class includes comprehensive validation to ensure data integrity.

Value

A MediationData S7 object

Examples

```
# Create a MediationData object
med_data <- MediationData(
  a_path = 0.5,
  b_path = 0.3,
  c_prime = 0.2,
  estimates = c(0.5, 0.3, 0.2),
  vcov = diag(3) * 0.01,
  sigma_m = 1.0,
  sigma_y = 1.2,
  treatment = "X",
  mediator = "M",
  outcome = "Y",
  mediator_predictors = "X",
  outcome_predictors = c("X", "M"),
  data = NULL,
  n_obs = 100L,
  converged = TRUE,
  source_package = "stats"
)
```

nde *Extract Natural Direct Effect (NDE)*

Description

Extract the natural direct effect from a mediation analysis result. The NDE represents the effect of treatment on outcome that does NOT operate through the mediator.

Usage

```
nde(x, ...)
```

Arguments

x	A MediationData, SerialMediationData, or BootstrapResult object
...	Additional arguments passed to methods

Details

For both simple and serial mediation:

$$NDE = c'$$

where c' is the direct effect coefficient.

Value

A numeric value with optional attributes for confidence intervals

See Also

nie(), te(), pm(), paths()

Examples

```
# Generate example data
set.seed(123)
n <- 100
mydata <- data.frame(X = rnorm(n))
mydata$M <- 0.5 * mydata$X + rnorm(n)
mydata$Y <- 0.3 * mydata$X + 0.4 * mydata$M + rnorm(n)

med_data <- fit_mediation(
  formula_y = Y ~ X + M,
  formula_m = M ~ X,
  data = mydata,
  treatment = "X",
  mediator = "M"
)
```

```
nde(med_data)
```

`nie` *Extract Natural Indirect Effect (NIE)*

Description

Extract the natural indirect effect from a mediation analysis result. The NIE represents the effect of treatment on outcome that operates through the mediator.

Usage

```
nie(x, ...)
```

Arguments

`x`	A MediationData, SerialMediationData, or BootstrapResult object
`...`	Additional arguments passed to methods

Details

For simple mediation (MediationData):

$$NIE = a \times b$$

For serial mediation (SerialMediationData):

$$NIE = a \times d_{21} \times d_{32} \times \ldots \times b$$

Value

A numeric value (or named vector for SerialMediationData) with optional attributes for confidence intervals if available

See Also

`nde()`, `te()`, `pm()`, `paths()`

Examples

```
# Generate example data
set.seed(123)
n <- 100
mydata <- data.frame(X = rnorm(n))
mydata$M <- 0.5 * mydata$X + rnorm(n)
mydata$Y <- 0.3 * mydata$X + 0.4 * mydata$M + rnorm(n)

med_data <- fit_mediation(
```

```
  formula_y = Y ~ X + M,
  formula_m = M ~ X,
  data = mydata,
  treatment = "X",
  mediator = "M"
)

nie(med_data)
```

`ParallelMediationData` *ParallelMediationData: Parallel (Multiple-Mediator) Mediation Structure*

Description

S7 class for **parallel** mediation, where a treatment affects an outcome through two or more *independent* mediators operating in parallel ($X \to M_j \to Y$ for $j = 1, \ldots, k$). The total indirect effect is the sum of the per-mediator products, $\sum_{j=1}^{k} a_j b_j$. This complements MediationData (simple) and SerialMediationData (serial chains).

Arguments

`a_paths`	Numeric vector: treatment -> mediator effects $(a_1, \ldots, a_k)$.
`b_paths`	Numeric vector: mediator -> outcome effects $(b_1, \ldots, b_k)$; must be the same length as `a_paths`.
`c_prime`	Numeric scalar: direct effect $X \to Y$.
`estimates`	Numeric vector of all parameter estimates.
`vcov`	Square variance-covariance matrix of `estimates`.
`sigma_mediators`	Optional numeric vector of mediator residual SDs (length k), or NULL.
`sigma_y`	Optional numeric scalar outcome residual SD, or NULL.
`treatment`, `outcome`	Single character strings naming the treatment / outcome.
`mediators`	Character vector of mediator names (length k, unique).
`mediator_predictors`	List of predictor-name vectors, one per mediator.
`outcome_predictors`	Character vector of outcome-model predictor names.
`data`	Optional data frame, or NULL.
`n_obs`	Integer number of observations.
`converged`	Logical convergence flag.
`source_package`	Character name of the originating package.

Value

A ParallelMediationData S7 object.

Examples

```
pmd <- ParallelMediationData(
  a_paths = c(0.5, 0.4),
  b_paths = c(0.6, 0.3),
  c_prime = 0.2,
  estimates = c(0.5, 0.4, 0.6, 0.3, 0.2),
  vcov = diag(0.01, 5),
  treatment = "X",
  mediators = c("M1", "M2"),
  outcome = "Y",
  mediator_predictors = list("X", "X"),
  outcome_predictors = c("X", "M1", "M2"),
  n_obs = 200L,
  converged = TRUE,
  source_package = "medfit"
)

nie(pmd)   # total indirect effect: sum(a_j * b_j) = 0.42
paths(pmd) # a1, b1, a2, b2, c_prime
```

---

paths *Extract All Path Coefficients*

Description

Extract all path coefficients from a mediation analysis result.

Usage

```
paths(x, ...)
```

Arguments

x	A MediationData or SerialMediationData object
...	Additional arguments passed to methods

Details

For simple mediation (MediationData):

- a: Treatment -> Mediator (X -> M)
- b: Mediator -> Outcome (M -> Y | X)
- c_prime: Direct effect (X -> Y | M)

For serial mediation (SerialMediationData):

- a: Treatment -> First mediator
- d21, d32, ...: Mediator-to-mediator paths
- b: Last mediator -> Outcome
- c_prime: Direct effect

Value

A named numeric vector of path coefficients

See Also

nie(), nde(), te(), pm()

Examples

```
# Generate example data
set.seed(123)
n <- 100
mydata <- data.frame(X = rnorm(n))
mydata$M <- 0.5 * mydata$X + rnorm(n)
mydata$Y <- 0.3 * mydata$X + 0.4 * mydata$M + rnorm(n)

med_data <- fit_mediation(
  formula_y = Y ~ X + M,
  formula_m = M ~ X,
  data = mydata,
  treatment = "X",
  mediator = "M"
)

paths(med_data)
```

---

pm *Extract Proportion Mediated (PM)*

Description

Extract the proportion of the total effect that is mediated (operates through the mediator).

Usage

```
pm(x, ...)
```

Arguments

x	A MediationData, SerialMediationData, or BootstrapResult object
...	Additional arguments passed to methods

Details

$$PM = \frac{NIE}{TE} = \frac{NIE}{NIE + NDE}$$

The proportion mediated can be:

- Between 0 and 1: Normal mediation
- Greater than 1: Suppression (direct and indirect effects have opposite signs)
- Negative: Inconsistent mediation

Value

A numeric value between 0 and 1 (or negative/greater than 1 in cases of suppression effects)

See Also

nie(), nde(), te(), paths()

Examples

```
# Generate example data
set.seed(123)
n <- 100
mydata <- data.frame(X = rnorm(n))
mydata$M <- 0.5 * mydata$X + rnorm(n)
mydata$Y <- 0.3 * mydata$X + 0.4 * mydata$M + rnorm(n)

med_data <- fit_mediation(
  formula_y = Y ~ X + M,
  formula_m = M ~ X,
  data = mydata,
  treatment = "X",
  mediator = "M"
)

pm(med_data)
```

---

print.mediation_effect

Print Method for mediation_effect

Description

Print Method for mediation_effect

Usage

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

Arguments

x	A mediation_effect object
...	Additional arguments (ignored)

Value

Invisibly returns x (the mediation_effect object). Called for its side effect of printing a formatted effect summary to the console.

---

print.summary.BootstrapResult

Print Summary for BootstrapResult

Description

Print Summary for BootstrapResult

Usage

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

Arguments

x	A summary.BootstrapResult object
...	Additional arguments (ignored)

Value

Invisibly returns x (the summary.BootstrapResult object). Called for its side effect of printing the formatted summary to the console.

print.summary.MediationData

Print Summary for MediationData

Description

Print Summary for MediationData

Usage

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

Arguments

x	A summary.MediationData object
...	Additional arguments (ignored)

Value

Invisibly returns x (the summary.MediationData object). Called for its side effect of printing the formatted summary to the console.

print.summary.SerialMediationData

Print Summary for SerialMediationData

Description

Print Summary for SerialMediationData

Usage

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

Arguments

x	A summary.SerialMediationData object
...	Additional arguments (ignored)

Value

Invisibly returns x (the summary.SerialMediationData object). Called for its side effect of printing the formatted summary to the console.

quick *Quick Summary of Mediation Results*

Description

Print a one-line summary of mediation results, perfect for quick checks or ADHD-friendly workflows.

Usage

```
quick(x, digits = 3, ...)
```

Arguments

x	A MediationData object (or result from med())
digits	Integer: number of significant digits (default: 3)
...	Additional arguments (ignored)

Details

Prints a compact one-line summary showing:

- NIE (Natural Indirect Effect) with CI if available
- NDE (Natural Direct Effect)
- Proportion Mediated (PM)

If bootstrap results are available (from med(..., boot = TRUE)), confidence intervals are shown for NIE.

Value

Invisibly returns x

See Also

med(), nie(), nde(), pm()

Examples

```
# Generate example data
set.seed(123)
n <- 100
mydata <- data.frame(X = rnorm(n))
mydata$M <- 0.5 * mydata$X + rnorm(n)
mydata$Y <- 0.3 * mydata$X + 0.4 * mydata$M + rnorm(n)

result <- med(
  data = mydata,
```

```
  treatment = "X",
  mediator = "M",
  outcome = "Y"
)

# One-line summary
quick(result)
```

`SerialMediationData` *SerialMediationData S7 Class*

Description

S7 class for serial mediation models where the effect flows through multiple mediators in sequence: X -> M1 -> M2 -> ... -> Mk -> Y.

This class supports serial mediation chains of any length, from simple two-mediator models (product-of-three: a * d * b) to complex chains with many mediators (product-of-k).

Arguments

`a_path`	Numeric scalar: effect of treatment on first mediator (X -> M1)
`d_path`	Numeric vector: sequential mediator-to-mediator effects • For 2 mediators (X -> M1 -> M2 -> Y): scalar d21 (M1 -> M2) • For 3 mediators (X -> M1 -> M2 -> M3 -> Y): c(d21, d32) • For k mediators: vector of length (k-1)
`b_path`	Numeric scalar: effect of last mediator on outcome (Mk -> Y)
`c_prime`	Numeric scalar: direct effect of treatment on outcome (X -> Y)
`estimates`	Numeric vector: all parameter estimates
`vcov`	Numeric matrix: variance-covariance matrix of estimates
`sigma_mediators`	Numeric vector or NULL: residual SDs for mediator models. Length should match number of mediators. First element is residual SD for M1 model, second element for M2 model, etc.
`sigma_y`	Numeric scalar or NULL: residual SD for outcome model
`treatment`	Character scalar: name of treatment variable
`mediators`	Character vector: names of mediators in sequential order. First element is M1, second element is M2, etc.
`outcome`	Character scalar: name of outcome variable
`mediator_predictors`	List of character vectors: predictor names for each mediator model. First list element contains predictors for M1 (typically just "X"), second element contains predictors for M2 (typically c("X", "M1")), etc.

`outcome_predictors`	Character vector: predictor names in outcome model
`data`	Data frame or NULL: original data
`n_obs`	Integer scalar: number of observations
`converged`	Logical scalar: whether all models converged
`source_package`	Character scalar: package/engine used for fitting

Details

Serial Mediation Structure:

Serial mediation models the indirect effect flowing through a sequence of mediators. The total indirect effect is the product of all path coefficients:

- **2 mediators (product-of-three)**: Indirect = a * d * b
- **3 mediators (product-of-four)**: Indirect = a * d21 * d32 * b
- **k mediators (product-of-k+1)**: Indirect = a * d21 * d32 * ... * d(k,k-1) * b

Path Notation:

- `a`: Treatment -> First mediator (X -> M1)
- `d21`: First -> Second mediator (M1 -> M2)
- `d32`: Second -> Third mediator (M2 -> M3)
- `dji`: Previous mediator -> Current mediator
- `b`: Last mediator -> Outcome (Mk -> Y)
- `c'`: Direct effect (X -> Y, controlling for all mediators)

Extensibility:

This class is designed to handle serial chains of any length:

- Minimal case: 2 mediators (length(d_path) = 1)
- No upper limit on chain length
- Validator ensures consistency between mediators and paths

Value

A SerialMediationData S7 object

Examples

```
# Two-mediator serial mediation (X -> M1 -> M2 -> Y)
# Product-of-three: a * d * b
serial_data <- SerialMediationData(
  a_path = 0.5,       # X -> M1
  d_path = 0.4,       # M1 -> M2 (scalar for 2 mediators)
  b_path = 0.3,       # M2 -> Y
  c_prime = 0.1,      # X -> Y (direct)
  estimates = c(0.5, 0.4, 0.3, 0.1),
  vcov = diag(4) * 0.01,
  sigma_mediators = c(1.0, 1.1),  # SD for M1, M2 models
  sigma_y = 1.2,
```

```
  treatment = "X",
  mediators = c("M1", "M2"),
  outcome = "Y",
  mediator_predictors = list(
    c("X"),           # M1 ~ X
    c("X", "M1")      # M2 ~ X + M1
  ),
  outcome_predictors = c("X", "M1", "M2"),  # Y ~ X + M1 + M2
  data = NULL,
  n_obs = 100L,
  converged = TRUE,
  source_package = "lavaan"
)

# Three-mediator serial mediation (X -> M1 -> M2 -> M3 -> Y)
# Product-of-four: a * d21 * d32 * b
serial_data_3 <- SerialMediationData(
  a_path = 0.5,           # X -> M1
  d_path = c(0.4, 0.35),  # M1 -> M2, M2 -> M3 (vector for 3 mediators)
  b_path = 0.3,           # M3 -> Y
  c_prime = 0.1,
  estimates = c(0.5, 0.4, 0.35, 0.3, 0.1),
  vcov = diag(5) * 0.01,
  sigma_mediators = c(1.0, 1.1, 1.05),  # SD for M1, M2, M3 models
  sigma_y = 1.2,
  treatment = "X",
  mediators = c("M1", "M2", "M3"),
  outcome = "Y",
  mediator_predictors = list(
    c("X"),              # M1 ~ X
    c("X", "M1"),        # M2 ~ X + M1
    c("X", "M1", "M2")   # M3 ~ X + M1 + M2
  ),
  outcome_predictors = c("X", "M1", "M2", "M3"),
  data = NULL,
  n_obs = 100L,
  converged = TRUE,
  source_package = "lavaan"
)
```

---

te *Extract Total Effect (TE)*

Description

Extract the total effect from a mediation analysis result. The TE is the sum of the indirect and direct effects.

Usage

```
te(x, ...)
```

Arguments

x	A MediationData, SerialMediationData, or BootstrapResult object
...	Additional arguments passed to methods

Details

$$TE = NIE + NDE$$

Value

A numeric value with optional attributes for confidence intervals

See Also

nie(), nde(), pm(), paths()

Examples

```
# Generate example data
set.seed(123)
n <- 100
mydata <- data.frame(X = rnorm(n))
mydata$M <- 0.5 * mydata$X + rnorm(n)
mydata$Y <- 0.3 * mydata$X + 0.4 * mydata$M + rnorm(n)

med_data <- fit_mediation(
  formula_y = Y ~ X + M,
  formula_m = M ~ X,
  data = mydata,
  treatment = "X",
  mediator = "M"
)

te(med_data)

# Verify: TE = NIE + NDE
nie(med_data) + nde(med_data)
```

Index