Package: medfit (via r-universe)

October 31, 2026

Type Package

Title Infrastructure for Mediation Model Fitting and Extraction

Version 0.16.0

Date 2026-10-30

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 (generalized linear models, linear mixed models for cluster-randomized designs, and a maximum-likelihood structural equation engine; Bayesian methods are planned), 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, lifecycle, MASS, nloptr (>= 1.0.4), rlang

Suggests knitr, lavaan (>= 0.6-0), lme4, numDeriv, OpenMx, pbkrtest, regmedint, sandwich, testthat (>= 3.0.0), tibble, withr, yaml

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

Config/roxygen2/version 8.1.0

LazyData true

Config/pak/sysreqs cmake

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

Date/Publication 2026-10-31 06:31:28 UTC

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

RemoteRef HEAD

RemoteSha c37c7ee8b1cd1e47d0d61f3d55b0a5fce335c7cc

Contents

bootstrap_mediation . . . 3
BootstrapResult . . . 7
ClusterMediationData . . . 8
compare_lr . . . 10
decompose . . . 12
extract_mediation . . . 13
fit_mediation . . . 16
fit_sem . . . 21
InteractionMediationData . . . 27
joint_effects . . . 29
JointMediationData . . . 30
mbco_sem . . . 32
med . . . 35
medfit-syntax . . . 37
mediation_demo . . . 41
MediationData . . . 43
nde . . . 45
nie . . . 46
ParallelMediationData . . . 47
paths . . . 48
pm . . . 50
quick . . . 51
SEMFit . . . 52
SEMFit-methods . . . 53
SerialMediationData . . . 56
te . . . 59
tidy_s7_object . . . 60

Index **62**

`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.

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,
  cluster = 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`	A mediation data object (required for parametric/plugin): MediationData, SerialMediationData, ParallelMediationData, InteractionMediationData, or JointMediationData. `statistic_fn` receives its named `@estimates` vector, which includes path aliases (e.g. `a`, `d1`, `b`, `c_prime` for a serial chain; `a1`, `b1`, `a2`, `b2` for parallel mediators). For a JointMediationData the aliases alone cannot reproduce the NDE, which also needs the prefixed intercept and covariate rows (`m1_`, ..., `y_`) and the sample covariate means.
`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)
cluster	Character string: name of a cluster variable in data, for method = "nonparametric" only (an error with the other methods). The resampling then draws whole clusters with replacement and gives each draw a fresh cluster id before statistic_fn sees the data, so a cluster drawn twice is refit as two clusters, not one bigger one. Singular fits and convergence warnings from a refit count as failures: they are excluded, their messages are suppressed, and their number is added to the warning. NULL (default) resamples rows as before.
...	Additional arguments (reserved for future use)

Details

Bounded FIML objects supplied as mediation_data or data are refused, including method = "plugin"; this entry point does not provide FIML bootstrap inference. Object-aware guards run before package-controlled RNG, workers or refits. Opaque user callbacks and external-first dispatch are outside that guarantee. See fit_sem() for supported normal intervals.

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)
```

A fixed seed gives the same resamples for any `ncores`, and for sequential runs, because every resample is drawn before the work is split across cores. This holds for the nonparametric and cluster methods (earlier versions drew inside each worker). The statistic is still computed in the workers, so a `statistic_fn` that uses random numbers itself is not covered (in a sequential run its draws now follow all the resampling draws instead of interleaving with them, so its stream differs from earlier versions), and floating-point results can differ in the last digits when BLAS threads differ. The resamples are held in memory before the refits start: `n_boot` integer vectors of length `n` (about `4 * n * n_boot` bytes; 800 MB for `n = 100000` and `n_boot = 2000`).

Value

A BootstrapResult object containing:

- Point estimate
- Percentile confidence interval bounds (the $\alpha/2$ and $1 - \alpha/2$ quantiles of the bootstrap distribution)
- Bootstrap distribution (for parametric and nonparametric)
- Method used

See Also

BootstrapResult, MediationData, `fit_mediation()`

Examples

```
# Fit mediation model
med_data <- fit_mediation(
  formula_y = outcome ~ treatment + mediator1 + covariate1 + covariate2,
  formula_m = mediator1 ~ treatment + covariate1 + covariate2,
  data = mediation_demo,
  treatment = "treatment",
  mediator = "mediator1"
)

# Define indirect effect function
indirect_fn <- function(theta) theta["m_treatment"] * theta["y_mediator1"]

# 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(mediator1 ~ treatment + covariate1 + covariate2,
              data = boot_data)
  fit_y <- lm(outcome ~ treatment + mediator1 + covariate1 + covariate2,
              data = boot_data)
  unname(coef(fit_m)["treatment"] * coef(fit_y)["mediator1"])
}

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

```
if (requireNamespace("lme4", quietly = TRUE)) {
# Treatment assigned to whole clusters (needs lme4); formulas hold fixed
# effects only, the engine adds the random cluster intercept
set.seed(1)
J <- 30
id <- rep(seq_len(J), each = 6)
cdat <- data.frame(school = factor(id), X = sample(rep(0:1, J / 2))[id])
cdat$M <- 0.5 * cdat$X + rnorm(J, sd = 0.5)[id] + rnorm(J * 6)
cdat$Y <- 0.2 * cdat$X + 0.4 * cdat$M + rnorm(J, sd = 0.5)[id] + rnorm(J * 6)
# Resample whole clusters. A refit that is singular or does not converge is
# dropped, and the warning counts how many were
nie_of <- function(d) {
  unname(nie(fit_mediation(Y ~ X + M, M ~ X, data = d, treatment = "X",
                           mediator = "M", engine = "lmer",
                           cluster = "school")))
}
result_cluster <- bootstrap_mediation(
  nie_of, method = "nonparametric", data = cdat, cluster = "school",
  n_boot = 20, seed = 1
)
print(result_cluster)
}
```

`BootstrapResult` *BootstrapResult S7 Class*

Description

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

Usage

```
BootstrapResult(estimate, ci_lower, ci_upper, ci_level, boot_estimates,
  n_boot, method, call)
```

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
)
```

ClusterMediationData *ClusterMediationData: Cluster-Level Treatment Mediation (2-1-1)*

Description

S7 class for mediation with a treatment assigned to whole clusters (such as schools) and a mediator and outcome measured on the individuals in them. It holds the paths of a two-level linear mixed model pair: the mediator model `M ~ X + ... + (1 | cluster)` and the outcome model with the mediator split into its within-cluster deviation and the observed cluster mean. medfit computes the effects; causal interpretation is the user's responsibility.

Usage

```
ClusterMediationData(a_path, b_within, b_between, c_prime, estimates, vcov,
  kr_df, treatment, mediator, outcome, cluster, n_obs, n_clusters,
  cluster_sizes, parameterization, covariates_centered, se_type, reml,
  converged, sigma_m, sigma_y, tau_m, tau_y, data, source_package)
```

Arguments

a_path	Numeric scalar: effect of the treatment on the mediator.
b_within	Numeric scalar: within-cluster mediator effect on the outcome.
b_between	Numeric scalar: between-cluster (cluster-mean) mediator effect on the outcome.
c_prime	Numeric scalar: direct effect of the treatment.
estimates	Named numeric vector of parameter estimates: the four alias rows plus each model's fixed effects.
vcov	Square variance-covariance matrix of estimates, with matching dimnames.
kr_df	Named numeric vector of Kenward-Roger degrees of freedom, one per path, when se_type = "kr"; NULL otherwise.
treatment, mediator, outcome	Character names of the variables.
cluster	Character name of the cluster variable.
n_obs	Integer number of observations.
n_clusters	Integer number of clusters.
cluster_sizes	Integer vector with one size per cluster.
parameterization	"within" or "raw": how the outcome model was fitted.
covariates_centered	Logical: whether every level-1 covariate in the outcome model has a cluster-mean companion.
se_type	"model" or "kr".
reml	Logical: whether the models were fitted by REML.
converged	Logical convergence flag.
sigma_m, sigma_y	Numeric scalars: residual standard deviations of the mediator and outcome models.
tau_m, tau_y	Numeric scalars: random-intercept standard deviations of the mediator and outcome models.
data	Optional data frame, or NULL.
source_package	Character name of the originating package.

Details

With the within parameterization $Y_{ij} = \theta_0 + c'X_j + b_W(M_{ij} - \bar{M}_j) + b_B\bar{M}_j + \dots$ and a the effect of the treatment on the mediator, the natural indirect effect is ab_B. It splits into an own-mediator part ab_W and a spillover part $a(b_B - b_W)$.

The split is the large-cluster, cluster-average approximation. With the observed cluster mean, moving only member i's mediator by a also moves the mean by a/n_j, so the exact own-mediator effect is $a[b_W + (b_B - b_W)/n_j]$ and the exact spillover is $a(b_B - b_W)(n_j - 1)/n_j$; they still sum to ab_B. decompose() warns when the approximation error exceeds half the own effect's standard error.

The estimand assumes randomized treatment, intact clusters, no interference between clusters, linear models without mediator-by-treatment or mediator-by-covariate products, and interference through the observed cluster mean only; all members of a cluster must be in the analysis rows. The own effect also needs no unmeasured lower-level mediator-outcome confounding. The spillover, NIE and NDE need no unmeasured upper-level mediator-outcome confounding either; under it only the sum of the direct effect and $a(b_B - b_W)$ is identified. `print()` and `summary()` print this block. The validator ties the alias rows `a`, `c_prime`, `b_within` and `b_between` of `estimates` and `vcov` to the path properties, so an object with inconsistent numbers cannot be built.

`kr_df` holds the Kenward-Roger degrees of freedom of each path when `se_type = "kr"`, and is empty otherwise.

Value

A `ClusterMediationData` S7 object.

References

Talloen, W., Moerkerke, B., Loeys, T., De Naeghel, J., Van Keer, H., & Vansteelandt, S. (2016). Estimation of indirect effects in the presence of unmeasured confounding for the mediator-outcome relationship in a multilevel 2-1-1 mediation model. *Journal of Educational and Behavioral Statistics*, 41(4), 359-391. doi:10.3102/1076998616636855

VanderWeele, T. J. (2010). Direct and indirect effects for neighborhood-based clustered and longitudinal data. *Sociological Methods & Research*, 38(4), 515-544. doi:10.1177/0049124110366236

Examples

```
# Hand-built object (the paths and their alias rows must agree)
est <- c(a = 0.5, c_prime = 0.2, b_within = 0.3, b_between = 0.6)
vc <- diag(0.01, 4)
dimnames(vc) <- list(names(est), names(est))
cmd <- ClusterMediationData(
  a_path = 0.5, b_within = 0.3, b_between = 0.6, c_prime = 0.2,
  estimates = est, vcov = vc,
  treatment = "X", mediator = "M", outcome = "Y", cluster = "school",
  n_obs = 400L, n_clusters = 40L, cluster_sizes = rep(10L, 40),
  parameterization = "within", covariates_centered = TRUE, se_type = "model",
  reml = TRUE, converged = TRUE, sigma_m = 1, sigma_y = 1, tau_m = 0.5,
  tau_y = 0.5, source_package = "medfit"
)
cmd@a_path * cmd@b_between  # the NIE, 0.5 * 0.6
```

`compare_lr` *Compare medfit's Product-Null LR with OpenMx or lavaan*

Description

Bounded FIML fits are refused before an external-engine comparison or refit; this function does not extend their supported inference. See `fit_sem()`.

Runs the likelihood-ratio test of a product null ($ab = 0$) in an external SEM engine and compares it with medfit's own `mbco_sem()` statistic. A general-purpose optimizer handed the nonlinear constraint $ab = 0$ can drop it (OpenMx at a zero start) or stop at the wrong branch (lavaan), which reports a statistic above the true minimum. `compare_lr()` flags exactly those cases and nothing else.

A warning is raised only on observed evidence, never because an LR is zero: `medfit_warning_engine_constraint_droppe` when the engine's returned fit violates the product constraint, and `medfit_warning_engine_statistic_above_max` when the engine's statistic exceeds medfit's. Both inherit from `medfit_warning_engine_comparison`. If medfit has no accepted reference, the engines' unrestricted fits differ from medfit's, or a run fails or cannot be checked, the verdict is `"unresolved"` and neither side is blamed. medfit has no accepted reference when one of its own refits cannot be fitted (see `mbco_sem()`): the reference LR is then `NA` and the result is a comparison with that verdict, not an error.

Only observed-variable models are compared. With either engine, a saturated mean structure (every intercept free) and `fixed_x` (all, a named subset or a lenient subset) are supported: the fixed exogenous moments are written to the engine as fixed parameters at their sample values (in OpenMx, a fit with a mean structure is run on the raw data the fit used). Latent variables, product terms, fixed or constrained intercepts, weights, factors, other fixed or shared parameters, start/bound modifiers and constraints in the model are refused by name.

Known limit (#320): the flip of the `fit_sem()` defaults in 0.15.0 does not change what the comparison finds. On the pinned design (15 seeds by 4 cells, 60 runs per engine), the lavaan run ends above medfit's reference in 18 of 60 runs and the OpenMx run in 13 of 60, the same counts as under the 0.14.x defaults. Each is reported as `medfit_warning_engine_statistic_above_max`, and no row is mislabeled.

Usage

```
compare_lr(object, paths, engine = c("lavaan", "openmx"), tolerance = 1e-04)
```

Arguments

`object`	A SEMFit from `fit_sem()`.
`paths`	Character vector: the labels of the regression paths whose product is the indirect effect, as in `mbco_sem()`.
`engine`	`"lavaan"` or `"openmx"`. The package must be installed.
`tolerance`	Relative tolerance on the LR difference, scaled by `max(1, LR)`. Default `1e-4`; values above `1e-2` are refused, because a loose tolerance would call different statistics equal.

Value

An object of class `medfit_lr_comparison`: a list with `verdict` (`"agree"`, `"disagree"` or `"unresolved"`), `reason`, `reference` (medfit's LR, `NA` when it has none, and whether it was accepted), `runs` (one row per engine start with the LR, convergence, constraint residual and status) and `engine`.

See Also

mbco_sem(), fit_sem()

Examples

```
if (requireNamespace("lavaan", quietly = TRUE)) {
  fit <- fit_sem(
    "mediator1 ~ a*treatment\noutcome ~ b*mediator1 + cp*treatment",
    data = mediation_demo, fixed_x = FALSE, meanstructure = NULL
  )
  compare_lr(fit, paths = c("a", "b"), engine = "lavaan")
}
```

---

decompose *Decomposition of a Mediation Effect*

Description

Return the components of the total effect for an object whose outcome model has treatment-by-mediator products.

For an InteractionMediationData object this is VanderWeele's (2014) four-way decomposition: controlled direct effect (CDE), reference interaction (INTref), mediated interaction (INTmed), and pure indirect effect (PIE), together with the derived natural direct and indirect effects and the total effect. For a JointMediationData object it is the CDE and the joint natural direct and indirect effects (VanderWeele and Vansteelandt 2014).

Usage

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

Arguments

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

Details

For a 0/1 treatment, outcome model $Y = \theta_0 + \theta_1 X + \theta_2 M + \theta_3 XM + \ldots$, mediator model $M = \beta_0 + \beta_1 X + \ldots$, and reference mediator level m^*:

$$CDE = \theta_1 + \theta_3 m^*$$

$$INTref = \theta_3 (E[M \mid X = 0, \bar{c}] - m^*)$$

$$INTmed = \theta_3 \beta_1$$

$$PIE = \theta_2 \beta_1$$

with $NDE = CDE + INTref$, $NIE = INTmed + PIE$, and $TE = NDE + NIE$. $E[M \mid X = 0, \bar{c}]$ is the mediator model's prediction at no treatment and the covariate means.

Value

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

References

VanderWeele, T. J. (2014). A unification of mediation and interaction: A 4-way decomposition. *Epidemiology*, 25(5), 749–761.

VanderWeele, T. J., & Vansteelandt, S. (2014). Mediation analysis with multiple mediators. *Epidemiologic Methods*, 2(1), 95–115. doi:10.1515/em20120010

See Also

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

Examples

```
fit_m <- lm(mediator1 ~ treatment + covariate1 + covariate2,
            data = mediation_demo)
fit_y <- lm(outcome ~ treatment * mediator1 + covariate1 + covariate2,
            data = mediation_demo)
med_int <- extract_mediation(fit_m, model_y = fit_y,
                             treatment = "treatment", mediator = "mediator1")
decompose(med_int)
```

`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, and a native SEMFit).

Usage

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

Arguments

`object`	Fitted model object (lm, glm, lavaan, lmerMod, or a SEMFit)
`...`	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 two linear mixed models for a treatment assigned to whole clusters, as a ClusterMediationData: the mediator model is `object`, the outcome model is `model_y`, and `cluster` names the cluster variable. Both models must be fitted to the same rows; the cluster means must be computed on those rows, and a pair of fits that kept different individuals is an error (best effort: it compares the data row names the fits kept, so it cannot see a mismatch between frames whose row names were both reset)
- **SEMFit**: Extract from a model fitted by `fit_sem()` (the native maximum-likelihood engine), with the same arguments and structures as the lavaan method. A treatment-by-mediator product written `x:m` (or `m:x`) in the outcome equation gives the four-way decomposition, stored under the canonical name `m:x`; without it `decomposition = "four_way"` is refused by name. `interaction =` takes that product written with `:`.
- **brmsfit**: Extract from Bayesian models (future)

Note: OpenMx extraction is planned for a future release.

The returned objects share one interface (`nie()`, `nde()`, `te()`, `confint()`, `bootstrap_mediation()`, ...) for use by other medfit functions and dependent packages (probmed, RMediation, medrobust).

The class returned depends on the structure. A single mediator gives a MediationData object, or an InteractionMediationData object when the outcome model has a treatment-by-mediator term. A `mediator` vector of length two or more gives a SerialMediationData or ParallelMediationData object; for lm/glm fits whose outcome model has a treatment-by-mediator term written with `:` or `*`, it gives a JointMediationData object with the joint natural effects of the mediators. Any other product term in a multi-mediator fit errors.

Arguments for lm and glm models:

With `object` the fitted mediator model (the first mediator's model for several mediators), the lm/glm method takes:

- `model_y`: the fitted outcome model. With several mediators include every mediator, not only the last: omitting one that also affects the outcome biases the others' coefficients.
- `treatment`: name of the treatment variable.
- `mediator`: name of the mediator, or an ordered character vector of two or more mediator names.
- `mediator_models`: list of the fitted models for mediators 2 to k, in order; required whenever `mediator` has length two or more.
- `structure`: `"auto"` (default), `"serial"`, or `"parallel"`; `"auto"` classifies the mediators from the mediator models.
- `decomposition`: `"auto"` (default) uses the four-way decomposition when a single mediator's outcome model has a treatment-by-mediator term; `"four_way"` requires that term; `"two_way"` ignores it and returns a MediationData object.
- `m_star`: the reference mediator level for the controlled direct effect (default 0; for JointMediationData a scalar or a vector named by the interacting mediators).

- vcov_fun: function returning each model's covariance matrix (default stats::vcov(); for example sandwich::vcovHC). Not supported for JointMediationData.
- outcome, data: optional; detected from the models when omitted.

The lavaan method takes a fitted lavaan model and the variable names; see ?extract_mediation_lavaan for its arguments.

Covariance of the estimates:

For lm/glm fits, each equation's block of @vcov is that model's own covariance. The blocks for different equations are stored as zero. This is exact for simple mediation, serial chains, and the four-way model, whose later equations contain every regressor of the earlier ones, but for parallel mediators it leaves out the covariance between their a paths. JointMediationData stores the full stacked least-squares covariance, and a lavaan fit estimates all equations jointly, so for the same data the intervals can differ between engines.

The four-way effects that contain a covariate mean (int_ref, the NDE and the total effect) treat that mean as a fixed constant, so their standard errors and intervals are conditional on the observed covariates. They omit the sampling variance of the estimated mean, theta3 * gamma times the variance of the covariate mean, so they can understate the uncertainty of a population-average effect. CDE, int_med, PIE and the NIE do not depend on the mean. For population-average standard errors set options(medfit.covariate_mean_variance = TRUE): tidy() and confint() then add the variance of the estimated covariate mean, (theta3 * gamma)^2 times the variance of the covariate mean (case-weighted when the fit has weights), to int_ref, the NDE and the total effect; the default stays conditional, as lavaan's := and the glm route are. Or resample rows with bootstrap_mediation() (method = "nonparametric").

Coefficient names:

The simple single-mediator MediationData from lm/glm fits names three coefficients m_<treatment>, y_<mediator> and y_<treatment> (m_X, y_M and y_X for variables X and M), and code that looks coefficients up by name, such as the parametric bootstrap in probmed, relies on them. The lavaan method adds the same three names to @estimates and @vcov for an observed, continuous treatment, mediator and outcome. Each is a second name for the a, b or c_prime path, with the same estimate and covariances, appended after the existing rows. A lavaan fit with a latent or ordered treatment, mediator or outcome, and any extraction with standardized = TRUE, gets none of the three and works with the plugin estimators only. Serial, parallel and four-way objects are not affected.

Value

An S7 object whose class depends on the mediation structure (see Details): MediationData, InteractionMediationData, SerialMediationData, ParallelMediationData, JointMediationData, or ClusterMediationData. Each carries the path coefficients, the full parameter vector and its variance-covariance matrix, residual standard deviations (for Gaussian models), variable names, and the original data when available.

See Also

MediationData, fit_mediation(), bootstrap_mediation()

Examples

```
# Extract the mediation structure from fitted lm models, using the
# simulated mediation_demo data bundled with medfit
fit_m <- lm(mediator1 ~ treatment + covariate1 + covariate2,
            data = mediation_demo)
fit_y <- lm(outcome ~ treatment + mediator1 + covariate1 + covariate2,
            data = mediation_demo)
med_data <- extract_mediation(fit_m, model_y = fit_y,
                              treatment = "treatment", mediator = "mediator1")



if (requireNamespace("lme4", quietly = TRUE)) {
# Treatment assigned to whole clusters (needs lme4); formulas hold fixed
# effects only, the engine adds the random cluster intercept
set.seed(1)
J <- 30
id <- rep(seq_len(J), each = 6)
cdat <- data.frame(school = factor(id), X = sample(rep(0:1, J / 2))[id])
cdat$M <- 0.5 * cdat$X + rnorm(J, sd = 0.5)[id] + rnorm(J * 6)
cdat$Y <- 0.2 * cdat$X + 0.4 * cdat$M + rnorm(J, sd = 0.5)[id] + rnorm(J * 6)
# The outcome model carries the mediator as its within-cluster deviation
# plus the observed cluster mean
cdat$M_bar <- ave(cdat$M, cdat$school)
cdat$M_w <- cdat$M - cdat$M_bar
fit_m <- lme4::lmer(M ~ X + (1 | school), data = cdat)
fit_y <- lme4::lmer(Y ~ X + M_w + M_bar + (1 | school), data = cdat)
cluster_med <- extract_mediation(fit_m, model_y = fit_y, treatment = "X",
                                 mediator = "M", cluster = "school")
nie(cluster_med)
}
```

`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.

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", "kr"),
  engine_args = list(),
  m_star = 0,
  cluster = NULL,
  model = NULL,
  ...
)
```

Arguments

formula_y	Formula for outcome model (e.g., Y ~ X + M + C). Not used with engine = "native", which takes model instead.
formula_m	Formula for mediator model (e.g., M ~ X + C). Not used with engine = "native".
data	Data frame containing all variables. With the "glm" engine, rows with a missing value on any variable in either formula are dropped before fitting, so both models share one sample; nobs() and @n_obs count the rows kept. Row-level arguments (weights and, in ..., subset, offset, etastart and mustart) refer to the rows of data as passed: they are applied to those rows first, then incomplete rows are dropped.
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) • "regmedint": Closed-form regression-based (in)direct effects via the suggested **regmedint** package (VanderWeele's regression approach, with optional treatment-mediator interaction) • "native": medfit's own maximum-likelihood SEM engine, fitted from model syntax given as model (see fit_sem()); the fit is then extracted with extract_mediation(). Supports observed and latent mediators, equality and inequality constraints, and := parameters. Case weights and se_type = "sandwich" map to fit_sem()'s weights (weighted maximum likelihood) and its robust sandwich covariance. • "lmer": Linear mixed models for a treatment assigned to whole clusters (the 2-1-1 design), via the suggested **lme4** package; needs cluster = and returns a ClusterMediationData. The engine drops incomplete rows once, splits the mediator and each level-1 covariate into its within-cluster deviation and observed cluster mean (<M>_cwc, <M>_cm), adds a random cluster intercept to both models, and fits by REML. The formulas hold fixed effects only (a random-effect term such as (1 | id) is an error; random slopes go in engine_args). It does not take weights, non-Gaussian families, se_type = "sandwich", or extra arguments.

`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. With `engine = "lmer"` only, `"kr"` uses the Kenward-Roger adjusted covariance and stores a degrees of freedom for each path, so the path intervals are t intervals; it needs REML fits and the suggested **pbkrtest** package. With `engine = "native"`, `"sandwich"` is the HC0-type robust covariance (see `fit_sem()`), which is smaller than the glm engine's HC3 by a factor that shrinks like 1/n.

`engine_args` Named list of engine-specific overrides (default: `list()`, no overrides). Ignored by `engine = "glm"`. For `engine = "native"`, recognized names are `information`, `n_starts`, `meanstructure`, `fixed_x`, `missing` and `control` (passed to `fit_sem()`, whose defaults for `fixed_x` and `meanstructure` changed in 0.15.0) and `outcome`, `structure`, `decomposition`, `interaction`, `a_label`, `b_label` and `cp_label` (passed to `extract_mediation()`); any other name is an error. Bounded FIML requires `engine_args = list(missing = "fiml", fixed_x = FALSE)` and the settings/limits in `fit_sem()`; `se_type = "model"` stays at the top level. Listwise defaults and the other engines' missing-data behavior are unchanged. For `engine = "lmer"`, recognized names are `random_y` and `random_m` (one-sided formulas of level-1 terms that get correlated random slopes; `~ M` in `random_y` is applied to the within-cluster mediator term) and `REML` (default `TRUE`); any other name is an error. For `engine = "regmedint"`, recognized names are `interaction`, `cvar`, `mreg`, `yreg`, `a0`, `a1`, and `c_cond`; each replaces the value the adapter would otherwise derive from the formulas, families, and data. The reference mediator level is set with `m_star`, not here.

`m_star` Numeric scalar: reference mediator level m^* at which the controlled direct effect is evaluated (default: `0`). Used only when `formula_y` carries a treatment-by-mediator term, i.e. when the returned object is an InteractionMediationData. Supplying it for a fit that has no such term is an error rather than a silent no-op. That check keys on whether the argument was given at the call site, not on whether it differs from the default, so a wrapper that forwards `m_star` unconditionally will trigger it on two-way fits; forward it only when its own caller supplied one.

`cluster` Character string: name of the cluster variable in `data`. Required with `engine = "lmer"`, an error with the other engines.

`model` Character string: model syntax for `engine = "native"` (required there, an error with any other engine). `treatment` and `mediator` name variables in the syntax; a latent mediator need not be a column of `data`. Because `model` is a named argument, an argument passed to `stats::glm()` through `...` that partially matches it (`mod = FALSE`) now binds here and is an error with the glm engine; drop it, since `glm()`'s own `model` argument cannot be passed through.

... Additional arguments passed to the fitting function. With the "glm" engine, subset (evaluated in the calling frame), offset, etastart and mustart describe the rows of data as passed, so a vector must have length nrow(data); incomplete rows are dropped after they are applied. Each is evaluated in the calling frame, not inside data, and is passed to both the mediator and outcome models.

Details

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.)
- With a non-identity link the path coefficients are on the link scale, where the product ab is not a natural indirect effect on the outcome scale; interpret it with care or use engine = "regmedint" for closed-form effects of a logistic outcome
- A treatment-by-mediator term in formula_y requires Gaussian identity-link models (the four-way formulas are linear); other families error
- 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

regmedint Engine:

When engine = "regmedint":

- Delegates to regmedint::regmedint() (suggested package) for closed-form natural (in)direct effects, with or without a treatment-mediator interaction
- Returns MediationData or InteractionMediationData depending on whether formula_y contains a treatment-by-mediator interaction term; use engine_args to override the derived regmedint arguments

Reference Mediator Level:

When the fit yields an InteractionMediationData, m_star fixes the level m^* at which the controlled direct effect is read off: $CDE = \theta_1 + \theta_3 m^*$ and $INTref = \theta_3(E[M \mid X = 0] - m^*)$. The

two shift in compensating directions, so `nde()`, `nie()`, `te()`, and `pm()` are invariant to `m_star`; only the CDE/INTref split moves.

The engines reach the same result by different routes. `engine = "glm"` applies `m_star` at *extraction* time, after the coefficients are fit; `engine = "regmedint"` passes it to `regmedint::regmedint()` as `m_cde`, where it is consumed by that package's closed-form estimator at *fitting* time. Supplying `m_star` for a fit with no treatment-by-mediator term is an error, not a silent no-op.

Value

A MediationData object containing the fitted mediation structure, or an InteractionMediationData object when `formula_y` contains a treatment-by-mediator interaction term.

References

VanderWeele, T. J. (2014). A unification of mediation and interaction: A 4-way decomposition. *Epidemiology*, 25(5), 749–761.

See Also

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

Examples

```
# mediation_demo is simulated data bundled with medfit; its covariates
# confound the mediator-outcome relation, so both models adjust for them
med_data <- fit_mediation(
  formula_y = outcome ~ treatment + mediator1 + covariate1 + covariate2,
  formula_m = mediator1 ~ treatment + covariate1 + covariate2,
  data = mediation_demo,
  treatment = "treatment",
  mediator = "mediator1"
)
print(med_data)


# Binary outcome (takes longer to fit): dichotomize the outcome at its median
demo <- mediation_demo
demo$outcome_bin <- as.integer(demo$outcome > stats::median(demo$outcome))
med_data_bin <- fit_mediation(
  formula_y = outcome_bin ~ treatment + mediator1 + covariate1 + covariate2,
  formula_m = mediator1 ~ treatment + covariate1 + covariate2,
  data = demo,
  treatment = "treatment",
  mediator = "mediator1",
  family_y = binomial()
)



if (requireNamespace("lme4", quietly = TRUE)) {
# Treatment assigned to whole clusters (needs lme4); formulas hold fixed
# effects only, the engine adds the random cluster intercept
```

```
set.seed(1)
J <- 30
id <- rep(seq_len(J), each = 6)
cdat <- data.frame(school = factor(id), X = sample(rep(0:1, J / 2))[id])
cdat$M <- 0.5 * cdat$X + rnorm(J, sd = 0.5)[id] + rnorm(J * 6)
cdat$Y <- 0.2 * cdat$X + 0.4 * cdat$M + rnorm(J, sd = 0.5)[id] + rnorm(J * 6)
cluster_fit <- fit_mediation(
  Y ~ X + M, M ~ X, data = cdat,
  treatment = "X", mediator = "M",
  engine = "lmer", cluster = "school"
)
nie(cluster_fit)       # a * b_between
decompose(cluster_fit) # own-mediator and spillover parts
}
```

`fit_sem` *Fit a Structural Equation Model by Maximum Likelihood*

Description

Fits a model written in lavaan-style syntax to raw data with medfit's native maximum-likelihood engine, and returns a SEMFit object. The engine handles observed and latent variables, user labels, fixed values, starts and bounds, equality and inequality constraints (==, <, >, linear or nonlinear), and defined parameters (:=) with delta-method standard errors.

Usage

```
fit_sem(
  model,
  data,
  information = c("observed", "expected"),
  n_starts = NULL,
  control = list(),
  weights = NULL,
  se_type = c("model", "sandwich"),
  meanstructure = TRUE,
  fixed_x = TRUE,
  sampling_weights = lifecycle::deprecated(),
  missing = c("listwise", "fiml")
)
```

Arguments

`model`	Character: model syntax. Lines may use ~ (regression), =~ (latent variable), ~~ (variance or covariance), `label*variable` modifiers, `start()`, `lower()` and `upper()`, and the constraint operators ==, <, > and :=. An intercept is written y

~ 1 (alone or with other terms, as in y ~ 1 + x) and turns on the mean structure. A product of two observed variables is written x:m on the right side of a regression (canonical name m:x); ?"medfit-syntax" lists the supported class and each limit.

data — A data frame with the model's observed variables. Numeric columns are used as is; a factor, character or logical column named on a regression's right side is expanded to 0/1 dummies (?"medfit-syntax" lists the rules and limits). Rows with missing values in those columns are dropped once (listwise deletion) and counted in n_dropped under the default missing = "listwise". Bounded missing = "fiml" accepts numeric model columns only, retains rows with any observed model variable, and drops only all-empty model rows; extra columns do not select rows.

information — Character: "observed" (default) or "expected" information for the standard errors. "expected" is an error for a model with a product term that has an endogenous factor (the product is a function of another variable, so the expected information gives wrong standard errors); use the default with se_type = "sandwich". Bounded FIML supports only "observed".

n_starts — Positive integer or NULL: number of optimizer starts tried before the fit is declared failed (NULL uses 5). Retries use perturbed starts drawn under a fixed seed and never change the caller's random number stream; with 3 or more starts the last attempt is the model's own moment-based start, so a start in a bad basin does not end the fit. With a nonlinear constraint, which can leave several local optima, every start is tried and the best accepted one is kept (a warning says when the accepted starts disagree); otherwise the first accepted start is used. A supported nonzero-product equality searches both sign branches independently, with n_starts starts per nonempty branch; aggregate diagnostics report the actual attempts, not the number of branches. For FIML, only NULL or 3 is supported: three starts per solver, with refinements counted separately.

control — List: options passed to the optimizer (nloptr). FIML requires the empty list under its initial validated numerical protocol.

weights — Optional numeric vector of non-negative case weights, one per row of data (NULL, the default, fits unweighted). The fit is weighted maximum likelihood on the weighted mean and covariance; the scale of the weights does not matter. Rows dropped for missing values drop their weights with them. Use with se_type = "sandwich". FIML rejects every non-NULL weight vector.

se_type — Character: "model" (default) takes the covariance of the estimates from the inverse information; "sandwich" uses the robust sandwich covariance $A^{-1}BA^{-1}$ with casewise scores (HC0-type: it equals sandwich::vcovHC(type = "HC0") per equation and lavaan's robust standard errors with sampling.weights, and is smaller than the HC3 default of the glm route by a factor that shrinks like $1/n$). Model-based standard errors are not valid under weights.

meanstructure — Logical or NULL. TRUE (default) fits a mean structure; NULL fits one only when the model writes an intercept (y ~ 1, free or fixed); FALSE fits covariances only and errors if an intercept is written. With a mean structure every observed variable whose intercept is not written has a free one, so a model with only free intercepts has the same estimates and standard errors as the covariance-only fit. A latent variable's mean is fixed at 0. A model with a product term (y ~ x + m +

x:m) always fits the mean structure (it needs the mediator intercept for the four-way decomposition), so with a product meanstructure = FALSE is an error. An explicit NULL or FALSE is kept as given for listwise fits. FIML requires TRUE.

fixed_x — TRUE (default), FALSE, or a character vector of exogenous observed variable names. The selected variables (TRUE: every exogenous observed variable, product columns included) have their mean (with a mean structure), variance and the exogenous covariances the model contains fixed at the sample moments of the estimation sample (weighted when weights are given) and removed from estimation, as lavaan does with fixed.x = TRUE. The structural estimates, their standard errors, the degrees of freedom and the chi-square do not change; the parameter count and the log likelihood do: the log likelihood (and so AIC and BIC) is conditional on the fixed variables, and glance()$fixed_x lists them. Models whose fixed_x sets or data differ cannot be compared with stats::anova(), mbco_sem(), AIC() or BIC(). A (co)variance or intercept statement on a selected variable is an error (lavaan would silently keep it free). The fixed rows stay in the parameter table (fixed_x column, no standard error) but not in coef() or vcov(). Left unset it resolves to an explicit subset: every exogenous observed variable (product and factor-dummy columns included) except those a ~~ or ~1 statement names, which are left free with a message of class medfit_message_fixed_x_lenient (an explicit TRUE errors there instead; an explicit FALSE is kept). glance()$fixed_x names the set the fit used.

sampling_weights — **[Deprecated]** Renamed weights, the name fit_mediation() uses; the old name still works with a warning.

missing — Exact missing-data method: listwise (default) or bounded Gaussian fiml (no abbreviations). FIML requires explicit fixed_x = FALSE; see the Bounded FIML section. The argument is appended to preserve existing positional calls.

Value

A SEMFit object. With a product term whose factor is a regression target (x:m with m regressed on x), glance() reports the chi-square, df and p-value as NA with a note, and anova() and mbco_sem() refuse; see SEMFit-methods. diagnostics$ledger holds the versioned outcome ledger described below.

Defaults changed in 0.15.0

fixed_x is the exogenous observed variables (lavaan's fixed.x = TRUE) and meanstructure = TRUE, so a default fit matches a default lavaan::sem() fit. Against 0.14.x the parameter count, logLik(), AIC(), BIC() and the names in coef(), vcov() and tidy() change; the structural estimates and their standard errors did not change on the cases measured. Which random start a multi-start fit tries can change too (the retry seed follows the parameter names). To keep the 0.14.x results, pass fixed_x = FALSE, meanstructure = NULL. The 0.14.x options(medfit.native_defaults) opt-in no longer has any effect.

Bounded FIML

Explicit missing = "fiml" fits a joint Gaussian random-X model under ignorable missingness (for example, MAR with the usual ignorability conditions). It evaluates observed components, with-

out imputing values. It supports exactly three numeric continuous observed variables in X -> M -> Y, with the X -> Y path free or fixed exactly at zero, independent disturbances and free intercepts. Names and labels are arbitrary. Extra predictors, latent variables, products, factors, correlated disturbances, parameter ties, bounds, general constraints, defined parameters and syntax-supplied starts are not supported. Input checks and local numerical gates do not establish global identification, a global optimum or inference coverage.

Supply `fixed_x = FALSE` explicitly; omission, `TRUE` and named subsets are errors. Require `meanstructure = TRUE`, `information = "observed"`, `se_type = "model"`, no weights, empty `control`, and `n_starts = NULL` or `3`. These numerical-control restrictions reproduce the initial validated protocol. Missing values never select FIML automatically, and a failed fit never falls back to listwise deletion. Native `fit_mediation()` forwards these settings through `engine_args`; its `se_type = "model"` remains a top-level argument.

`nobs()` counts rows with any observed model variable; `n_dropped` counts only all-empty rows. Singleton patterns and data with no complete rows are supported when the local identification and information checks pass. Original positional row IDs and pattern counts are retained. `logLik()` includes the full Gaussian constants, with its `df` attribute equal to the free-parameter count (8 or 9). AIC and BIC use that likelihood and retained row count. The discrepancy `f` and model-test `df` are `NA`: no saturated missing-data reference model is fitted.

Supported operations include SEMFit coefficient/covariance and likelihood summaries, parameter tidy/normal confidence intervals, and extraction followed by `paths()`, `nie()`, `nde()`, `te()`, `coef()`, `vcov()`, print/summary, tidy and normal path/effect confidence intervals. Extracted aliases follow topology; source labels/full covariance remain in the validated durable fit context. At a = b = 0, the NIE point value remains available but its inferential fields are unavailable; a known fixed direct path is a different case.

SEMFit `glance()` reports likelihood/IC fields with unavailable model-test fields; extracted-object `glance()`, `quick()`, `pm()`, bootstrap, profiles, MBCO and fit tests are deferred and refuse FIML objects. Object-aware package guards precede package-controlled refits/RNG/workers; opaque user callbacks and external-first dispatch remain outside that guarantee. Adapter/topology/numerical failures retain stage, reason, original parent and completed attempts; basic argument-type errors can occur earlier. A dedicated `fiml-1` ledger binds likelihood, rows, settings and attempts; the version-1 ledger below continues to describe listwise fits. See the Methods and Formulas article, Bounded FIML, for the likelihood and the native engine guide for the supported workflow. Bounded FIML was introduced in 0.16.0; released 0.15.0 does not provide this option.

Nonzero-product search coverage

One equality `c*a*b == q` is supported when the constants and `q/c` are finite and nonzero, `a` and `b` are distinct free scalar labels after expanding definitions, and no other constraint row involves either label. The engine fits both signs of `a` independently with the existing optimizer and acceptance gates, respecting user bounds and effective variance floors. It returns the lowest accepted discrepancy only when each required sign branch is accepted or excluded by an explicit bounds contradiction. An unresolved branch stops the fit with `medfit_error_no_convergence`; an all-empty search returns no fit. Zero-product fitting is unchanged. Repeated factors, multiple product equalities, coupled rows and other unsupported shapes retain local search.

`internals$search_evidence` is a separate versioned internal record, not a change to the outcome-ledger schema. Its `coverage` is `charted` for covered sign branches, `local` for unsupported shapes, or `unresolved` for an incomplete search. It retains restrictions, context, branch states, candidates,

objectives, gates and attempts; validation checks their attribution to the model and sample. Charted coverage does not prove a global optimum within a branch. Existing optimum status meanings are retained, and objects without this optional record remain usable. Counts distinguish optimizer starts, accepted starts and synthetic reference attempts.

The stored raw constraint residual is diagnostic only. Its optional standardized value uses the gradient and covariance at an already available matching reference fit without the tested restriction, never the projected constrained covariance. No additional fit or covariance calculation is requested. The ordinary optimizer reference lacks covariance, so the standard error and standardized residual are `NA` with reason `reference_covariance_unavailable`. These diagnostics do not affect acceptance or inference.

Constrained standard errors

Observed-information covariance includes the curvature of active nonlinear constraints; expected information retains its Fisher definition. Sandwich standard errors use the corresponding information as bread. Active inequalities, user bounds and solver variance floors give a local tangent covariance, with exactly zero variance for a pinned coordinate. This is not the full sampling distribution at a boundary. See the Methods and Formulas article, under Constraints, for the covariance formula.

Outcome ledger

`diagnostics$ledger` is a list with `schema_version = 1L`. Each node has `id`, `parent_id`, the exact parsed `constraints`, and `context` (the estimation `sample`, the model syntax and parameters, the likelihood `domain`, the settings and `n_obs`). `enumeration` records `state` and the number of required children; `complete` means every required restriction was attempted, not that any was globally optimized. `outcome` is `resolved` or `unresolved`. Separate fields record `optimizer` (acceptance state, original status, decrement, retries and optimum diagnostics), `likelihood` (availability and `f`), `feasibility` (`witness` or `unknown`), `covariance` (`available`, `unavailable`, `not_retained` or `failed`), `improper`, the retained `notices` (warnings and messages), the original `condition` (class, message, call) and the recursive `children`. IDs are deterministic positions within the call, not certificates of equivalent parameter spaces. No global optimum is asserted, and a failed branch is unknown, never certified infeasible.

A node whose constraints hold zero-product rows also has `branch_map`: `state`, `raw_count`, the parser `factor_counts`, `unique_count` and `raw_to_canonical`, a structural restriction-set ID for each raw row. The IDs collapse duplicate or permuted constraint rows, not arbitrary affine equivalence, and every raw row is still fitted in its original order (ties go to the first). More than 64 raw combinations, duplicates included, is refused before the grid is allocated, even when the unique count would be smaller; that node is `not_enumerated`, with an unknown unique count and no raw mapping, and nothing is fitted or certified. Its message prints the count without scientific notation, labels a count above 2^{53} as approximate (a double need not be an exact cardinality there) and names an overflow; `raw_count` is `Inf` on overflow. If the map cannot be built while an error is recorded, its `state` is `unavailable` with that condition, and the original error is not replaced. The map is rebuilt from the node's constraints during validation, so a forged map is refused.

The one proof of emptiness in this version-1 ledger is the literal certificate `literal_zero_nonzero`: an explicit `label == 0` equality and an explicit finite nonzero literal equality on the same parsed label (`a == 0` with `a == 1`). Its label, source rows and values are kept in `certificate` and rebuilt

from the node's own parsed parameters and constraints before any inference uses it. A certified-empty branch is retained with `outcome = "excluded"`, `feasibility = "certified_empty"` and an unavailable likelihood: it is not fitted and is not a successful fit. No tolerance, solver status, numerical rank, bound, general affine contradiction or fixed value in the parameter table is a proof, and a literal too small to represent (`1e-999`) is unknown.

A zero-product fit returns only if at least one required branch resolves and every other branch resolves or is certified empty. A direct fit whose own equalities contradict literally stops with `medfit_sem_infeasible`, and a union whose every branch is certified empty with `medfit_sem_empty_union`; both are refinements of `medfit_error_no_convergence` and carry the ledger. Otherwise it stops with a `medfit_error_no_convergence` error, never a fit; the error carries the ledger of every required branch as `$ledger` and, when a branch survived, `$best_survivor` (its `id` and discrepancy `f`, diagnostic only, never a p-value). The union's own node does not aggregate its children's notices; inspect each child. The notices of a failed call are in the ledger only; the error message keeps its text. A data or model error keeps its own class and also carries `$ledger`. The estimation sample is stored once, on the root node; a descendant fitted on the same rows carries only `context$sample_hash`. A fit made before the ledger existed must be refitted before `mbco_sem()` can use it, which refuses it with a `medfit_error_ledger` error.

When the optimizer accepts a fit and the covariance stage explicitly diagnoses non-positive-definite information or a negative variance, the call stops with `medfit_sem_covariance_error`, never a fit. The class is a sibling of `medfit_error_no_convergence`: no optimizer failed, so the handlers that report an unresolved result for an optimizer failure do not take it. Its fields are `schema_version = 1L`, `stage = "covariance"`, `partial` (the accepted optimizer result: `f`, `theta`, sample, domain, dimensions and settings; no `vcov` or `defined`) and `ledger` (the same node as `partial$ledger`, with `covariance = "failed"` and `post_fit`: versioned evidence with the original condition's class, message, call and diagnostic kind, `information_not_pd` or `negative_variance`). Other covariance errors propagate unchanged, without a retained likelihood. Singular information still warns and returns an NA covariance, recorded as `unavailable` rather than `failed`. The message is the original one. Resolved here means an accounted-for accepted likelihood, not a successful covariance. A union root reports the first failing branch's `post_fit`, in branch order, even when another branch supplies its minimum likelihood, and does not claim the winning branch failed: inspect the branch IDs. A branch that did not converge still stops the call with `medfit_error_no_convergence` first. Definition values and gradients are checked before the covariance, and model, data, optimizer, objective and other errors get no covariance certificate. The condition is not accepted as an input to `mbco_sem()`, and a fit that carries `post_fit` is refused with `medfit_error_ledger`.

See Also

SEMFit, `fit_mediation()`

Examples

```
fit <- fit_sem(
  "mediator1 ~ a*treatment\noutcome ~ b*mediator1 + treatment\nab := a*b",
  data = mediation_demo, fixed_x = FALSE, meanstructure = NULL
)
fit@defined

# A treatment-by-mediator product: the four-way decomposition
fit4 <- fit_sem(
```

```
  "mediator1 ~ treatment\noutcome_int ~ treatment + mediator1 + treatment:mediator1",
  data = mediation_demo, se_type = "sandwich"
)
decompose(extract_mediation(fit4, treatment = "treatment", mediator = "mediator1",
                            outcome = "outcome_int"))
```

`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).

Usage

```
InteractionMediationData(a_path, b_path, c_prime, interaction, cde,
  int_ref, int_med, pie, nde, nie, total_effect, m_star, estimates, vcov,
  sigma_m, sigma_y, treatment, mediator, outcome, mediator_predictors,
  outcome_predictors, data, n_obs, converged, source_package)
```

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 $Y = \theta_0 + \theta_1 X + \theta_2 M + \theta_3 XM + \dots$ and mediator model $M = \beta_0 + \beta_1 X + \dots$: `a_path` = β_1, `b_path` = θ_2, `c_prime` = θ_1, `interaction` = θ_3. With reference level `m_star` (m^*) the components are $CDE = \theta_1 + \theta_3 m^*$, $INTref = \theta_3(E[M \mid X = 0, \bar{c}] - m^*)$, $INTmed = \theta_3\beta_1$, 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 = $\theta_2\beta_1$).

$E[M \mid X = 0, \bar{c}] = \beta_0 + \gamma^\top \bar{c}$ is the mediator model's prediction at no treatment and the covariate means $\bar{c}$. The effects are for the treatment contrast 0 to 1, and the lm/glm extractor requires Gaussian identity-link models. From `fit_mediation(engine = "regmedint")`, `@estimates` and `@vcov` also carry regmedint's own delta-method covariance of the four components and the derived effects.

Value

An `InteractionMediationData` S7 object.

References

VanderWeele, T. J. (2014). A unification of mediation and interaction: A 4-way decomposition. *Epidemiology*, 25(5), 749–761.

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
```

`joint_effects` *Joint Effects at a Given Parameter Vector*

Description

Recomputes the joint CDE, NDE, NIE and total effect of a JointMediationData object from a named parameter vector, holding the object's structure, reference levels (`m_star`) and sample covariate means fixed. With the default `estimates = object@estimates` it returns the stored effects. Its main use is as the statistic of a parametric bootstrap: the draws `bootstrap_mediation()` passes to `statistic_fn` are named like `@estimates`, and the NDE needs the prefixed intercept and covariate rows (`m1_`, ..., `y_`) and the covariate means, not only the path aliases.

Usage

```
joint_effects(object, estimates = object@estimates)
```

Arguments

`object`	A JointMediationData object.
`estimates`	Named numeric vector containing every prefixed source row of `object@estimates` (`m1_...`, `y_...`); alias rows are ignored. A missing source row is an error, not a zero.

Value

Named numeric vector: `cde`, `nde`, `nie`, `te`.

Examples

```
d <- mediation_demo
fit <- extract_mediation(
  lm(mediator1 ~ treatment + covariate1 + covariate2, d),
  model_y = lm(outcome_int ~ treatment * mediator1 + mediator2 +
                covariate1 + covariate2, d),
  treatment = "treatment", mediator = c("mediator1", "mediator2"),
  mediator_models = list(lm(mediator2 ~ treatment + mediator1 +
                            covariate1 + covariate2, d))
)
joint_effects(fit)

# Parametric bootstrap of the joint NIE
boot <- bootstrap_mediation(
  function(theta) joint_effects(fit, theta)[["nie"]],
  method = "parametric", mediation_data = fit, n_boot = 500, seed = 1
```

```
)
boot@ci_lower
boot@ci_upper
```

`JointMediationData` *JointMediationData: Joint Natural Effects of Multiple Mediators*

Description

S7 class for the **joint** natural effects of two or more mediators, serial or parallel, when the outcome model may contain treatment-by-mediator product terms (VanderWeele and Vansteelandt 2014). The mediators are treated as one block: the natural indirect effect (NIE) runs through every mediator and every path among them, and the natural direct effect (NDE) is the remainder. medfit computes the effects; causal interpretation is the user's responsibility.

Usage

```
JointMediationData(structure, mediators, treatment, outcome, interactions,
  a_total, b_paths, theta3, c_prime, cde, nde, nie, total_effect, m_star,
  estimates, vcov, data, n_obs, converged, source_package)
```

Arguments

`structure`	Single string, `"serial"` or `"parallel"`.
`mediators`	Character vector of mediator names, in causal order.
`treatment`, `outcome`	Single character strings.
`interactions`	Character vector naming the mediators that carry a treatment-by-mediator product in the outcome model (may be empty).
`a_total`	Numeric vector named by `mediators`: total effect of the treatment on each mediator (β^*_{1i}).
`b_paths`	Numeric vector named by `mediators`: outcome main effect of each mediator (θ_{2i}).
`theta3`	Numeric vector named by `interactions`: product coefficients (θ_{3i}).
`c_prime`	Numeric scalar: treatment main effect on the outcome (θ_1).
`cde`, `nde`, `nie`, `total_effect`	Numeric scalars: controlled direct, natural direct, natural indirect and total effects.
`m_star`	Numeric vector named by `interactions`: reference mediator levels for the CDE.
`estimates`	Named numeric vector of parameter estimates.
`vcov`	Square variance-covariance matrix of `estimates`.
`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

With outcome model $Y = \theta_1 X + \sum_i (\theta_{2i} + \theta_{3i} X) M_i + \dots$ and β^*_{1i} the total (propagated) effect of the treatment on mediator i (for a serial chain this includes the paths through earlier mediators), the unit-contrast effects are

$$NIE = \sum_i (\theta_{2i} + \theta_{3i}) \beta^*_{1i}$$

$$NDE = \theta_1 + \sum_i \theta_{3i} \mu^*_{0i}$$

$$CDE = \theta_1 + \sum_i \theta_{3i} m^*_i$$

and $TE = NDE + NIE$. θ_{3i} is zero for a mediator without a product term, and μ^*_{0i} is the mean of mediator i under no treatment at the covariate means (propagated down a serial chain like β^*_{1i}). The validator enforces the NIE and CDE identities, so an object with inconsistent numbers cannot be built.

What the joint NIE is, and is not:

The NIE is the effect through the mediators **as a block**: every path from the treatment through any mediator, including paths among the mediators. It is not split into per-mediator or per-path pieces, and none of its parts should be reported as the effect "through M1". For a serial chain it therefore differs from the default `nie()` of a SerialMediationData (`a * d * b`, the effect through the full chain only): with no product term, the joint NIE of `M1 -> M2` is $a_1 b_1 + (a_2 + d a_1) b_2$, not $a_1 d b_2$. The effects use the unit contrast of a 0/1 treatment (0 to 1).

Assumptions:

The joint effects are identified under no-unmeasured-confounding assumptions stated for the whole mediator vector: none for the treatment and outcome, none for the mediators and the outcome, none for the treatment and the mediators, and no mediator-outcome confounder affected by the treatment **outside the mediator vector** (VanderWeele and Vansteelandt 2014). In a serial chain the earlier mediators are affected by the treatment and confound the later ones; that is allowed precisely because they are part of the vector. Mediator-outcome and treatment-mediator confounders must be controlled for **every** mediator.

medfit's requirements:

- Every model (each mediator model and the outcome model) must carry the **same covariates**. This is a medfit limitation that keeps the serial-chain algebra and the covariance exact, not a requirement of the method; differing sets error, naming the terms.
- The models must be Gaussian with the identity link, unweighted, with an intercept, fit to the same rows; every mediator must appear in the outcome model; `mediator` lists the mediators in causal order.
- A product must be written in the outcome formula with `:` or `*`. A product **precomputed as a data column** (for example `XM <- X * M1` added to the data and then used as `Y ~ X + M1 + M2 + XM`) cannot be recognized from the formula: medfit treats it as an ordinary covariate and the effects ignore the interaction, with no error. Write `X * M1` in the formula instead.

Covariates and standard errors:

The NDE depends on the covariates; it is evaluated at their sample means. The effects are linear in the covariates, so this equals the sample average of the per-observation effects. Standard errors

use the delta method with analytic gradients over a stacked-OLS covariance that includes the correlation between parallel mediator equations. They treat the covariate means as fixed, so they are conditional on the observed covariates and slightly understate the uncertainty of a population-average effect. For a parametric bootstrap use `joint_effects()` as the statistic.

Value

A `JointMediationData` S7 object.

References

VanderWeele, T. J., & Vansteelandt, S. (2014). Mediation analysis with multiple mediators. *Epidemiologic Methods*, 2(1), 95–115. doi:10.1515/em20120010

Examples

```
# Hand-built parallel object with an X x M2 product (m2* = 0)
jmd <- JointMediationData(
  structure = "parallel", mediators = c("M1", "M2"),
  treatment = "X", outcome = "Y", interactions = "M2",
  a_total = c(M1 = 0.5, M2 = 0.3), b_paths = c(M1 = 0.3, M2 = 0.4),
  theta3 = c(M2 = 0.2), c_prime = 0.1,
  cde = 0.1, nde = 0.12, nie = 0.33, total_effect = 0.45,
  m_star = c(M2 = 0),
  estimates = c(a1 = 0.5, a2 = 0.3, b1 = 0.3, b2 = 0.4,
                theta3_M2 = 0.2, c_prime = 0.1),
  vcov = diag(0.01, 6),
  n_obs = 200L, converged = TRUE, source_package = "medfit"
)
jmd@nie  # (0.3 + 0) * 0.5 + (0.4 + 0.2) * 0.3 = 0.33
```

`mbco_sem` *Minimum-of-k Likelihood-Ratio Test for an Indirect Effect (MBCO)*

Description

Tests the null hypothesis that an indirect effect, the product of two or more labeled paths (ab, or a serial adb), is zero in a model fitted by `fit_sem()`. The product is zero exactly when one factor is, so the model is refitted once per path with that path set to zero; each is a linear equality constraint, so each refit is a single solve. The statistic is the smallest of the likelihood-ratio statistics, referred to a chi-square distribution with one degree of freedom (the minimum-of-k LRTs; see the Methods and Formulas article). The nonlinear constraint $ab = 0$ is never solved, because a nonlinear solver can stop at the branch with the larger statistic.

Usage

```
mbco_sem(
  object,
  a,
  b,
  n_starts = NULL,
  control = NULL,
  fit = lifecycle::deprecated(),
  paths = NULL
)
```

Arguments

object	A SEMFit from fit_sem(), fitted without weights.
a, b	Character strings: the labels of the two free regression paths whose product is the indirect effect (for example "a" and "b" in "M ~ a*X" and "Y ~ b*M"). They must be different, and each must be a labeled ~ row of the fit; a variance, a loading, an unlabeled path and a defined parameter are refused. For a product of more than two paths pass them all as paths, or as a character vector in a with b missing.
n_starts, control	Passed to the optimizer for the two refits, as in fit_sem(). NULL, the default, uses the settings object was made with, so a better optimum in the full fit than in a refit cannot inflate the statistic; an explicit value overrides them.
fit	**[Deprecated]** Renamed object; the old name still works with a warning.
paths	Character vector of two or more labels: the paths whose product is the indirect effect (c("a", "d", "b")). NULL, the default, uses a and b.

Details

Bounded FIML fits are refused before any constrained refit; their model tests and MBCO inference are not supported. See fit_sem() for the bounded limits.

A fit made with weights is refused: the weighted discrepancy is not a likelihood, so its difference is not chi-square. So is a fit that did not converge, since its discrepancy is the reference both refits are measured against.

A fit whose ledger is missing, malformed or does not describe the fit, and a refit that is not the null asked for, are refused with a medfit_error_ledger error: the evidence is never inferred from converged.

A refit that fails the acceptance gate of the optimizer does not stop the function: its row has converged = FALSE, f and diff_ll NA and the failure text in note, a warning repeats the text, and statistic and p.value are NA (never the minimum over the refits that worked, which would hide the failure). A zero-product branch that cannot be fitted stops fit_sem() itself (there is no partial fit to test against).

Inequality constraints in the model matter only when they put the null on the boundary of the parameter space. If a bound or an inequality that involves the tested parameter is active at its refit (a > 0 with the null a = 0), the function warns that the statistic is a mixture of chi-squares, not

chi-square(1), and that the p-value is not valid. A bound the null sits inside (`a < 0.2`), an inactive inequality and one on another parameter are not mentioned. A null the constraints rule out (`a > 0.3` with the null `a = 0`) is an unresolved row whose note says so.

Only a product of labeled paths is tested; a sum of products (several parallel indirect effects together) is not. If two of the paths are tied by an equality (`a == b`), their refits impose the same restriction and their `diff_ll` values coincide. The test is conservative: under a null where only one factor is zero its size is at most the nominal level, and with more than one zero factor it is smaller still.

Value

An object of class `c("mbco_sem", "htest")`: `statistic` (the smallest diffLL), `parameter` (`df = 1`), `p.value`, `method`, `data.name`, `n_obs`, `unresolved` (`TRUE` when any refit failed, see Details), and `tests`, a data frame with one row per refit (`constraint`, the minimized discrepancy `f`, `diff_ll` equal to `n_obs * (f - fit@f)`, `converged`, and `note`, the failure text of a refit that could not be fitted, otherwise empty). It prints with the standard `htest` layout. `ledger` is the version-1 outcome ledger documented in `fit_sem()`, with children `mbco/full` and `mbco/null:1`, `mbco/null:2`, ..., in the requested path order. A null refit that is itself a zero-product union keeps its branch children and notices. A null node's notices also record those raised at the MBCO boundary, including the union winner's replay, so they can repeat a child's. Every null is attempted even if another fails, and an unresolved null leaves the root `unresolved`. A nested union branch that is certified empty by a literal zero/nonzero contradiction (see the outcome ledger in `fit_sem()`) is retained as `excluded`; a null that is itself infeasible or an all-empty union stays an unresolved row, never a finite LR or a survivor-only p-value. A null whose optimizer accepted but whose covariance stage explicitly diagnosed non-positive-definite information or a negative variance (`medfit_sem_covariance_error` in `fit_sem()`) keeps its likelihood after a check that it is the null asked for (same rows, domain and one more restriction): the row stays resolved, the statistic and p-value are unchanged, a warning names the failure, and the node has `covariance = "failed"` with `post_fit` and the notices. No covariance is invented; a fit that carries `post_fit` is refused as `object`. Other covariance errors propagate unchanged; singular information keeps its warning and NA covariance.

See Also

`fit_sem()`, SEMFit

Examples

```
fit <- fit_sem(
  "mediator1 ~ a*treatment\noutcome ~ b*mediator1 + treatment",
  data = mediation_demo, fixed_x = FALSE, meanstructure = NULL
)
mbco_sem(fit, "a", "b")

# a serial indirect effect, the product of three paths
serial <- fit_sem(
  paste0("mediator1 ~ a*treatment\nmediator2 ~ d*mediator1 + treatment\n",
         "outcome ~ b*mediator2 + treatment"),
  data = mediation_demo
)
mbco_sem(serial, paths = c("a", "d", "b"))
```

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 a bootstrap confidence interval for the indirect effect? (default: FALSE for speed). Uses a parametric bootstrap of ab with a 95% percentile interval, attached to the result as the `"bootstrap"` attribute; call `bootstrap_mediation()` directly for another method or level.
`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

```
# mediation_demo is simulated data bundled with medfit; its covariates
# confound the mediator-outcome relation, so adjust for them
result <- med(
  data = mediation_demo,
  treatment = "treatment",
  mediator = "mediator1",
  outcome = "outcome",
  covariates = c("covariate1", "covariate2")
)
print(result)

# Quick summary
quick(result)


# With bootstrap CI (slower)
result_boot <- med(
  data = mediation_demo,
  treatment = "treatment",
  mediator = "mediator1",
```

```
  outcome = "outcome",
  covariates = c("covariate1", "covariate2"),
  boot = TRUE,
  n_boot = 1000,
  seed = 42
)
```

`medfit-syntax` *medfit Model Syntax*

Description

The model syntax read by `fit_sem()` and by `fit_mediation(engine = "native")`. It is a lavaan-style core plus two extensions, each marked "medfit extension" below. One statement per line (or separated by `;`).

Missing-data scope

The general syntax below describes the default listwise path. The separate bounded `fit_sem(missing = "fiml", fixed_x = FALSE)` option accepts only three numeric observed variables in simple mediation with a free or exactly zero direct path and free intercepts. General constraints, definitions, starts, bounds, latent variables, products and factors are not supported for FIML. See `fit_sem()` for all settings and downstream limits.

Core statements

Statement	Meaning
`y ~ x1 + x2`	regression of `y` on `x1` and `x2`
`f =~ y1 + y2 + y3`	latent variable `f` measured by the indicators
`y1 ~~ y2`	covariance; `y1 ~~ y1` a variance
`y ~ x + m + x:m`	product term: the observed variable `x * m` as a regressor (see Product terms)
`y ~ a*x`	label the path `a`; equal labels are one parameter
`y ~ 0.5*x`	fix the path at 0.5
`y ~ start(0.3)*x, lower(0)*, upper(1)*`	starting value, lower and upper bound
`a == b, a + b == 1, cp == a*b`	equality constraint on labels; may be nonlinear
`a < b, a > 0`	inequality constraint; `label < number` is a bound
`ab := a*b`	defined parameter, with a delta-method standard error

Constraints may be nonlinear in the labels (`cp == a*b`, `a*b == 0.1`, `a*b > 0.1`, `ab > 0` with `ab := a*b`), solved with the Jacobian recomputed at every point; a model with one tries every start and keeps the best accepted solution, warning when the starts disagree. For such a model the residual variances that appear in a nonlinear row get a lower bound of `1e-4` times their natural unit (the variable's variance), because a step to a negative variance is outside the row's domain; the fit warns when a bound is active, and a residual variance below that bound (an R-squared above 0.9999) is

not fitted exactly. The model without the nonlinear rows is fitted first: its estimate, and copies of it with the signs of the regression paths in the rows flipped (a product constraint cannot change sign along its surface, so each sign pattern is its own branch), are extra starts, and a feasible reference estimate is never replaced by a worse accepted fit. Models without a nonlinear row are not affected. One nonzero-product equality (`a*b == 0.1`, including a finite nonzero constant multiplier and definition aliases) has a separate search for each sign of its two distinct free labels. Each required sign branch must be accepted or excluded by its bounds; an unresolved branch stops the fit. Multiple product equalities, repeated factors and another constraint involving either factor retain local search. The internal coverage record distinguishes charted branches from local search, not global from local optima; see `fit_sem()`. A product of labels set to zero (`a*b == 0`, `b1*b2*b3 == 0`) is instead solved exactly: one linear fit per factor (`a == 0`, `b == 0`), keeping the smallest discrepancy; if a branch cannot be fitted, `fit_sem()` stops with a `medfit_error_no_convergence` error instead of returning the best of the others; the error carries the outcome ledger of every branch as `$ledger` (see `fit_sem()`). A sum of products (`a1*b1 + a2*b2 == 0`) has no exact reduction, so it is best-effort: the fit is the best of the starts tried, a disagreement warning means a better optimum may exist, and `n_starts` raises the number of starts. Constrained standard errors use the active tangent space: observed information includes nonlinear constraint curvature, while expected information retains its Fisher definition. Active inequalities and bounds, including solver variance floors, give a local tangent approximation rather than the full boundary sampling distribution. See `fit_sem()` and the Methods and Formulas article for details. Labels used in a constraint or a definition must be defined in the model. A constraint with no free parameter is an error. Exogenous variables are given free variances and covariances, as lavaan does with `fixed.x = FALSE`. An exogenous variable named in any `~~` statement is left out of that pairing: its variance and the covariances the model writes are kept, and its covariances with the other exogenous variables are 0 unless written (lavaan does the same). `fixed_x = TRUE` (or a vector of names) in `fit_sem()` fixes the moments of the selected ones at the sample values instead, as `fixed.x = TRUE` does. Unlike lavaan, which keeps an explicit `x ~~ x` or `x ~ 1` free under `fixed.x = TRUE` and lets the degrees of freedom move, medfit stops with an error when the model has a statement on a selected variable's moments. Any intercept statement (`y ~ 1`, or `1` as a term: `y ~ 1 + x`) turns the mean structure on and frees every observed intercept the model does not write, as lavaan does; `y ~ 0*1` then restricts the mean of `y` to 0 (one degree of freedom), and `meanstructure = TRUE` in `fit_sem()` turns it on without any statement, and a product term (`x:m`, see Product terms) always turns it on. A latent variable's mean is fixed at 0; a free latent `~1` is an error. A repeated `1` for one variable is an error. A variable cannot appear on both sides of `~` or `=~` (`y ~ y`), and a path, a (co)variance or an intercept takes one row: a repeated `y ~ x` is an error, not a second row that competes for the label.

Factor covariates

A factor, character or logical column named on a regression's right side is replaced by one 0/1 dummy per non-reference level, named `<column>_<level>` (`g` with levels `a`, `b`, `c` gives `g_b` and `g_c`). The reference level is the first factor level; a character column sorts in the C locale, so the result does not depend on the session locale. A logical column gives one dummy, `<column>_TRUE`. Dummies are built from the full column before rows with missing values are dropped, and a missing value in the factor drops the row once. `tidy()` and the parameter table show the dummies by name, and the fit's data holds the dummies, not the factor.

Limit	Message (starts with, or contains)
An ordered factor	"is an ordered factor"
A factor with one level	"has only one level"

A level that makes an invalid name	"makes the invalid name"
A dummy that is already a column	"is already a column of data"
A label, fixed value, start or bound on a factor (b*g)	"expands to"
A constraint that names the factor (g == 0)	"is a categorical column"
A product of two factors	"both sides are categorical columns"
A dummy with no variation in the rows used	"is constant in the rows the fit uses"
A factor as the outcome or a mediator	"must be numeric"
A factor as treatment or mediator in extract_mediation()	"is a factor in the model"

A label goes on one dummy: b*g_b works, b*g stops, and a constraint names the label (b == 0), never the factor. A bootstrap that refits from raw data re-expands the factor in each resample, so a level missing from a character column's resample changes the reference level. g:m expands to g_b:m and g_c:m, each a product term under the rules below. fixed_x = "g" fixes every dummy of g, and a dummy name fixes one. A factor covariate in the mediator equation enters the four-way covariate means through its dummies; a factor treatment or mediator is not supported (recode a two-level factor as a 0/1 numeric column), and the four-way route refuses a g:m product because its effects differ by level.

Product terms

x:m is the product of two observed variables, written with no spaces, on the right side of a regression (y ~ x + m + x:m). As in lavaan, the product is an observed exogenous variable: the engine builds the column x * m from data after listwise deletion, with a free variance, free covariances with the other exogenous variables and (the mean structure is on) a free mean. The likelihood does not change. x:m and m:x are one term, stored and printed under the canonical name, the two names sorted in the C locale (m:x); the coefficient is named y~m:x, or by its label (b*x:m labels the whole product). extract_mediation() finds the treatment-by-mediator product through the same name and returns the four-way decomposition; centering the mediator moves the split between the controlled direct effect and the reference interaction, not the natural effects.

A column in data named m:x or x:m is used only if it equals x * m on the rows kept; otherwise the fit stops. A product term turns the mean structure on, and meanstructure = FALSE is then an error. A coefficient equals the equation-wise least-squares estimate inside the class below, and the sandwich covariance (se_type = "sandwich") is the standard-error choice, because a product is not normal; the engine warns once when it is not used. fixed_x includes the product columns.

Limit	Message (starts with, or contains)
A space around :	"a product ':' needs a variable name on each side,
x:x	"of a variable with itself is not supported"
x:m:w	"a three-way or higher product"
Product as a left side (x:m ~ y, x:m ~~ y)	"cannot be a left side"
Product in =~	"is only valid on the right side of a regression"
Product inside a constraint or :=	"not inside the expression"
A latent variable in a product	"products with latent variables are not supported i
y ~ x:m + m:x (one product twice in an equation)	"are the same product (canonical name"
A factor that is the target or downstream of it (y ~ x + x:y)	"(rule 8: no outcome-derived regressors)"
A loop of regressions	"The model is not recursive"
A residual covariance between outcomes (m1 ~~ m2)	"cannot have a residual covariance (rule 9)"

A label shared by two rows	"is used on more than one parameter"
A fixed coefficient (`0.3*x`, `0*x`)	"has a fixed coefficient (rule 9)"
A bound (`lower(0)*x`)	"has a bound (rule 9"
==, < or > naming a coefficient label	"names the coefficient label"
A fixed, shared or constrained intercept	"(rule 9, intercept clause"
`meanstructure = FALSE` with a product	"`meanstructure = FALSE`, but the model has a pro
A column `m:x` in `data` that is not `x * m`	"is not the product of"
A product input that is not numeric, factor, character or logical (a date)	"needs numeric variables"
`information = "expected"`, endogenous factor	"with a product of an endogenous variable"
`anova()` or `mbco_sem()` on such a fit	"is not available for ... a model with a product tern
Another product on the treatment or mediator, in `extract_mediation()`	"does not support the product"

Rules 8 and 9 are checked on the parsed table, so they need no data. Rule 8 rejects a regressor built from the outcome (it is correlated with the equation's error); rule 9 names what is not yet supported with products, and the same model works without the product. The covariance that completion would add between two pure outcomes is fixed at 0 in a product model, with one note. Products of observed variables only: a product with a latent variable, `x:x` and three-way products are not supported.

The global chi-square is not valid when a factor of a product is itself a regression target (the model treats the product as exogenous although it is a function of that variable) or when the exogenous (co)variance block is restricted. `glance()` then reports chi-square, df and p-value as `NA` with a note, `anova()` and `mbco_sem()` refuse, and expected information is refused for an endogenous product; `logLik()`, AIC and BIC stay defined, but those of a fit with a product are not comparable with those of a fit without one, so `AIC(fit_a, fit_b)`, `BIC()`, `anova()` and `mbco_sem()` refuse such a pair. A product of exogenous variables with the block free keeps the ordinary test.

Defaults changed in 0.15.0

Since medfit 0.15.0 an unset `fixed_x` means the exogenous observed variables (as lavaan's `fixed.x = TRUE`) and an unset `meanstructure` means `TRUE`, so every observed intercept is free without a `~1` statement. Against 0.14.x the parameter count, `logLik()`, `AIC()`, `BIC()` and the parameter names move; the structural estimates and their standard errors did not move on the cases measured. A variable a `~~` or `~1` statement names is left free (message class `medfit_message_fixed_x_lenient`) instead of the error an explicit `fixed_x = TRUE` gives. To keep the 0.14.x results, pass `fixed_x = FALSE, meanstructure = NU`
The 0.14.x `options(medfit.native_defaults)` opt-in no longer has any effect. `fit_sem()` has the details.

medfit extensions

- **Comma shorthand** (medfit extension). A comma-separated left side expands to one statement per name: `y1, y2 ~ x` means `y1 ~ x` and `y2 ~ x`. Commas on the right side are not supported; use `+`. **lavaan reads** `y1, y2 ~ x` **differently: it silently keeps only** `y2`**.** A model copied between the two packages can therefore change meaning without an error.
- `CONSTRAINT(...)` (medfit extension). A constraint line may be written `CONSTRAINT(a == b)`, equivalent to the bare `a == b`. The keyword is case-sensitive and starts the statement, one pair of parentheses encloses the whole constraint, it holds exactly one comparison, and no text follows the closing parenthesis.

Expressions

Both sides of a constraint and the right side of a definition are arithmetic expressions in the labels, with `+ - * / ^`, parentheses, finite numbers, and the functions `exp`, `log`, `log10`, `sqrt`, `abs`, `min`, `max`, `sin`, `cos`, `tan`, `pnorm` and `qnorm`. Anything else is rejected with the construct named.

See Also

`fit_sem()`, SEMFit

Examples

```
# Labeled paths and a defined parameter
fit_sem(
  "mediator1 ~ a*treatment\noutcome ~ b*mediator1 + cp*treatment\nab := a*b",
  data = mediation_demo, fixed_x = FALSE, meanstructure = NULL
)@defined

# Comma shorthand: one left-hand side per name
coef(fit_sem(
  "mediator1, mediator2 ~ treatment\noutcome ~ mediator1 + mediator2 + treatment",
  data = mediation_demo
))[1:3]

# An equality constraint, written with the CONSTRAINT() keyword
coef(fit_sem(
  "mediator1 ~ a*treatment\noutcome ~ b*mediator1 + treatment\nCONSTRAINT(a == b)",
  data = mediation_demo
))[c("a", "b")]
```

`mediation_demo` *Simulated Mediation Data for Examples*

Description

A simulated dataset for demonstrating medfit's simple, serial, parallel, and treatment-by-mediator interaction workflows with one running example. The data are **simulated**: they describe no real study, population, or finding.

Usage

```
mediation_demo
```

Format

A data frame with 400 rows and 8 variables:

treatment Integer, 0/1. Randomized treatment assignment.

mediator1 Numeric. First mediator; affected by `treatment`.

mediator2 Numeric. Serial mediator; affected by `treatment` and `mediator1`.

mediator3 Numeric. Parallel mediator; affected by `treatment` only.

covariate1 Numeric. Continuous confounder of the mediator-outcome relations.

covariate2 Integer, 0/1. Binary confounder of the mediator-outcome relations.

outcome Numeric. Outcome with no product terms; use it for the simple, serial, and parallel models.

outcome_int Numeric. `outcome` plus a `treatment` by `mediator1` interaction; use it only for models with that interaction: the four-way decomposition with a single mediator, or joint effects with several mediators (JointMediationData).

Details

The variables follow a fictional workplace-training story, used only so the examples read concretely: randomized assignment to a training program (`treatment`) raises skill confidence (`mediator1`), which builds task mastery (`mediator2`); the assignment also triggers supervisor check-ins (`mediator3`); `outcome` is job performance.

All errors are independent standard normal. The generating equations are

$$M_1 = 0.5X + 0.3C_1 + 0.3C_2 + e_1$$

$$M_2 = 0.2X + 0.5M_1 + 0.2C_1 + e_2$$

$$M_3 = 0.5X + e_3$$

$$Y = 0.2X + 0.4M_1 + 0.3M_2 + 0.3M_3 + 0.3C_1 + 0.2C_2 + e_4$$

and `outcome_int` adds $0.5XM_1$.

Because the covariates confound the mediator-outcome relations, every model should adjust for both `covariate1` and `covariate2`. A model that omits a downstream mediator estimates reduced-form coefficients rather than the structural ones above: for example, the simple model `outcome ~ treatment + mediator1 + covariates` has limiting coefficients 0.55 for `mediator1` and 0.41 for `treatment`. For a serial model, include `mediator1` in the outcome model as well; the serial indirect effect $a \times d \times b$ is then the effect through the chain `mediator1` to `mediator2` only.

Source

Simulated for package demonstration; not drawn from or representing any real study. The generating script is `data-raw/mediation_demo.R` in the package source repository.

Examples

```
data(mediation_demo)
str(mediation_demo)

# Simple mediation, adjusting for both covariates
fit <- fit_mediation(
  formula_y = outcome ~ treatment + mediator1 + covariate1 + covariate2,
  formula_m = mediator1 ~ treatment + covariate1 + covariate2,
  data = mediation_demo,
  treatment = "treatment",
  mediator = "mediator1"
)
nie(fit)
```

`MediationData` *MediationData S7 Class*

Description

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

Usage

```
MediationData(a_path, b_path, c_prime, estimates, vcov, sigma_m, sigma_y,
  family_m, family_y, treatment, mediator, outcome, mediator_predictors,
  outcome_predictors, data, n_obs, converged, source_package)
```

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(s).

Usage

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

Arguments

x	A MediationData, SerialMediationData, ParallelMediationData, InteractionMediationData, or JointMediationData object.
...	Additional arguments passed to methods

Details

Without a treatment-by-mediator product (`MediationData`, `SerialMediationData`, `ParallelMediationData`) the NDE is the direct-path coefficient, $NDE = c'$. With a product it also depends on the mediator's mean under no treatment:

- `InteractionMediationData`: $NDE = CDE + INTref = \theta_1 + \theta_3 E[M \mid X = 0, \bar{c}]$.
- `JointMediationData`: $NDE = \theta_1 + \sum_i \theta_{3i}\mu^*_{0i}$, with μ^*_{0i} the mean of mediator i under no treatment at the covariate means.

Value

A numeric scalar of class `mediation_effect` carrying a `type` attribute.

See Also

`nie()`, `te()`, `pm()`, `paths()`, `decompose()`

Examples

```
med_data <- fit_mediation(
  formula_y = outcome ~ treatment + mediator1 + covariate1 + covariate2,
  formula_m = mediator1 ~ treatment + covariate1 + covariate2,
  data = mediation_demo,
  treatment = "treatment",
  mediator = "mediator1"
)

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(s).

Usage

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

Arguments

`x`	A MediationData, SerialMediationData, ParallelMediationData, InteractionMediationData, JointMediationData, or BootstrapResult object. For a `BootstrapResult`, `nie()` returns the bootstrapped point estimate (with a warning if the statistic was not an NIE).
`...`	Additional arguments passed to methods. For a `SerialMediationData`, `type = c("chain", "total")` selects the chain-specific (default) or the total indirect effect (see Details).

Details

The effects are for a unit contrast of the treatment. By class:

- `MediationData`: $NIE = ab$.
- `SerialMediationData`: with `type = "chain"` (the default), the effect through the full chain only, $NIE = a\, d_1 \cdots d_{k-1}\, b$; with `type = "total"`, the total indirect effect, the sum over every treatment-to-outcome path through at least one mediator (including paths that skip a mediator, such as X -> M2 -> Y), which equals `te(x) - nde(x)`.
- `ParallelMediationData`: $NIE = \sum_j a_j b_j$.
- `InteractionMediationData`: $NIE = INTmed + PIE = (\theta_2 + \theta_3)\beta_1$.
- `JointMediationData`: the joint NIE through all the mediators, $\sum_i (\theta_{2i} + \theta_{3i})\beta^*_{1i}$.

The "Methods and Formulas" article on the package website gives the full formulas.

Value

A numeric scalar of class `mediation_effect` carrying a `type` attribute. For intervals use `confint()` or `bootstrap_mediation()`.

See Also

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

Examples

```
med_data <- fit_mediation(
  formula_y = outcome ~ treatment + mediator1 + covariate1 + covariate2,
  formula_m = mediator1 ~ treatment + covariate1 + covariate2,
  data = mediation_demo,
  treatment = "treatment",
  mediator = "mediator1"
)

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).

Usage

```
ParallelMediationData(a_paths, b_paths, c_prime, estimates, vcov,
  sigma_mediators, sigma_y, treatment, mediators, outcome,
  mediator_predictors, outcome_predictors, data, n_obs, converged,
  source_package)
```

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, SerialMediationData, ParallelMediationData, InteractionMediationData, or JointMediationData 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
- d (two mediators) or d21, d32, ...: Mediator-to-mediator paths
- b: Last mediator -> Outcome
- c_prime: Direct effect

For parallel mediation (ParallelMediationData): a1, b1, a2, b2, ..., c_prime.

For InteractionMediationData: a, b, c_prime, and theta3 (the treatment-by-mediator coefficient).

For JointMediationData: the raw coefficients a1..aK, dij, b1..bK, theta3_<mediator>, and c_prime. The a paths here are the raw coefficients, not the propagated a* values used by the effects.

Value

A named numeric vector of path coefficients

See Also

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

Examples

```
med_data <- fit_mediation(
  formula_y = outcome ~ treatment + mediator1 + covariate1 + covariate2,
  formula_m = mediator1 ~ treatment + covariate1 + covariate2,
  data = mediation_demo,
  treatment = "treatment",
  mediator = "mediator1"
)

paths(med_data)
```

pm *Extract Proportion Mediated (PM)*

Description

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

Usage

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

Arguments

x	A MediationData, SerialMediationData, ParallelMediationData, InteractionMediationData, or JointMediationData object.
...	Additional arguments passed to methods

Details

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

For serial mediation (SerialMediationData) the numerator is the total indirect effect, `nie(x, type = "total")`, and the denominator the full total effect from `te()`.

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

The ratio has no delta-method standard error in medfit; bootstrap it with `bootstrap_mediation()` for supported objects. `pm()` and bootstrap inference refuse data extracted from bounded FIML fits; see `fit_sem()`.

Value

A numeric scalar of class `mediation_effect`, usually between 0 and 1 (negative or greater than 1 in cases of suppression effects), or `NA` with a warning when the total effect is numerically zero.

See Also

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

Examples

```
med_data <- fit_mediation(
  formula_y = outcome ~ treatment + mediator1 + covariate1 + covariate2,
  formula_m = mediator1 ~ treatment + covariate1 + covariate2,
  data = mediation_demo,
  treatment = "treatment",
  mediator = "mediator1"
)

pm(med_data)
```

---

quick *Quick Summary of Mediation Results*

Description

Print a one-line summary of mediation results, perfect for quick checks or a compact report line.

Usage

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

Arguments

x	A MediationData or SerialMediationData object (or result from med()). Other classes error; use print() or summary() for them.
digits	Integer: number of significant digits (default: 3)
...	Additional arguments (ignored)

Details

quick() refuses data extracted from bounded FIML fits. Use print(), summary() or tidy() for those objects; see fit_sem() for their limits.

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

```
result <- med(
  data = mediation_demo,
  treatment = "treatment",
  mediator = "mediator1",
  outcome = "outcome",
  covariates = c("covariate1", "covariate2")
)

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

SEMFit *SEMFit S7 Class*

Description

S7 class holding a structural equation model fitted by the native maximum-likelihood engine (see fit_sem()): estimates, their covariance, the parameter table, defined parameters (:=) with delta-method standard errors, and the diagnostics of the optimizer.

Usage

```
SEMFit(theta, vcov, table, defined, f, n_obs, n_dropped, df, information,
  converged, data, model, diagnostics, internals, call)
```

Arguments

theta	Named numeric vector: free-parameter estimates
vcov	Numeric matrix: covariance of theta (projected onto the active constraints and bounds; NA when the model is not identified)
table	Data frame: one row per parameter-table row with lhs, op, rhs, label, free, est and se
defined	Data frame of := parameters: name, expr, est, se
f	Numeric scalar: minimized maximum-likelihood discrepancy; NA for bounded FIML
n_obs, n_dropped	Integer scalars: rows used and rows dropped for missing values (listwise deletion by default; only all-empty model rows for FIML)
df	Numeric scalar: model-test degrees of freedom; NA for bounded FIML, which stores the free-parameter count separately in internals$likelihood$k

information	Character scalar: "observed" or "expected"
converged	Logical scalar: the acceptance gate passed. fit_sem() returns a fit only for an accepted solution (otherwise it errors), so this is TRUE for every fit it returns and glance() does not report it
data	Data frame: the rows used, on the model's observed variables
model	Character scalar: the model syntax
diagnostics	List: the versioned outcome ledger (see fit_sem()) and optimizer diagnostics (status, decrement, retries, active_bounds, active_constraints, improper, sign_check, and optimum: a list with n_starts, n_accepted, best_of, certified, spread (discrepancy range among accepted starts, NA when fewer than two or not best-of), and status (unique, certified, agreed, disagreed, or single))
internals	List: the internal index objects the engine needs (ram, par_map, constraints); not part of the interface
call	Call object or NULL: the call that produced the fit

Value

A SEMFit S7 object

See Also

fit_sem()

Examples

```
fit <- fit_sem(
  "mediator1 ~ a*treatment\noutcome ~ b*mediator1 + treatment\nab := a*b",
  data = mediation_demo, fixed_x = FALSE, meanstructure = NULL
)
fit@table[fit@table$op == "~", c("lhs", "rhs", "est", "se")]
fit@defined
fit@converged
```

SEMFit-methods *Methods for SEMFit objects*

Description

Accessors and display for a SEMFit returned by fit_sem().

Arguments

object, x	A SEMFit object
...	Additional arguments (ignored)

Details

- `coef()` returns the free-parameter estimates, named as lavaan names its coefficients: the label when the row has one, else `lhs op rhs` without spaces (`"M~X"`, `"X~~X"`, `"eta=~m2"`). `vcov()`, `tidy()` and the accessors `extract_mediation()` reads use the same names.
- `vcov()` returns their covariance, projected onto the active constraints and bounds; rows for a pinned parameter are zero.
- `nobs()` returns rows used after listwise deletion by default; for bounded FIML it counts rows with any observed model variable.
- For listwise fits, `logLik()` returns the log likelihood with the means free; its `df` attribute counts the free parameters plus one mean per observed variable that is not fixed. With `fixed_x` it is conditional on the fixed variables (lavaan's value), so `AIC()` and `BIC()` of fits with different fixed sets or data stop instead of ranking them; the same holds for `anova()` and `mbco_sem()`. A `fixed_x` fit also lists its fixed moments after the free parameters in `tidy()` (`fixed_x` column `TRUE`, no standard error) and in `summary()`; `coef()` and `vcov()` leave them out. `print()` and `summary()` say what the log likelihood is conditional on.
- **Bounded FIML.** Raw total Gaussian `logLik()` includes constants; its `df` attribute counts the eight or nine free parameters. AIC/BIC use retained N. `f` and model-test `df` are unavailable: no saturated missing-data reference fit is performed. SEMFit `glance()` retains likelihood/IC fields with unavailable model tests and a reason. Extracted-object `glance()` is instead deferred. Normal parameter intervals, extraction and unstandardized path/effect summaries are supported; bootstrap, profiles, MBCO and fit tests refuse. See `fit_sem()` for settings, provenance and dispatch boundaries.
- **Model test with a product term.** The global chi-square is not valid when a factor of a product term (`x:m`) is itself a regression target, because the model then treats the product as exogenous although it is a function of `m`; nor when the variances and covariances of the exogenous variables (every product included) are restricted (fixed, labeled, bounded, or left out because a `~~` statement names one of them; a `fixed_x` moment is not a restriction). `fit_sem()` then warns once, `glance()` reports `chisq`, `df` and `p_value` as `NA` (with the reason in `attr(, "note")`), `summary()` prints the reason, and `anova()` and `mbco_sem()` stop naming the product. The coefficients equal equation-wise least squares and the sandwich standard errors are unaffected, but only with observed information: `fit_sem(information = "expected")` stops for a product with an endogenous factor, because its standard errors (the sandwich built on them included) are wrong there. `fit_sem()` also warns once for any product term fitted without `se_type = "sandwich"`. `logLik()`, `AIC()` and `BIC()` stay defined, but the values of a fit with a product are not comparable with those of a fit without one (the product column enters the likelihood), so `AIC(fit_a, fit_b)`, `BIC()`, `anova()` and `mbco_sem()` refuse such a pair.
- `anova()` tests nested fits by the likelihood-ratio statistic $n(F_r - F_f)$ on the difference in model degrees of freedom: `anova(fit_full, fit_restricted)`, or several fits, each tested against the one with fewer degrees of freedom. It compares fits that were already made and never refits. The fits must use the same observed variables and rows, and neither may have sampling weights or have failed to converge. Nesting is checked by parameter-table inclusion: the restricted fit may free no parameter the other leaves out and must have more degrees of freedom. That is necessary for nesting, not sufficient; whether two models are nested in substance is the analyst's responsibility. A restricted fit with a smaller discrepancy than the model it restricts is an error (one fit is not at its global optimum). When a bound or an inequality on a tested parameter is active at the restricted fit, the null lies on the boundary of

the parameter space, the statistic is a mixture of chi-squares, and the function warns that the p-value is not valid, as mbco_sem() does. The p-value is still returned; the logical attribute attr(x, "p_valid") (named by row, NA for the first) is FALSE for each such row, so code can read the verdict without parsing the warning. It belongs to the object anova() returns: subsetting it with [or binding rows drops the attribute. Returns an object of class "anova" with columns Df, Chisq (n times the discrepancy), Chisq diff, Df diff and Pr(>Chisq). Name rows by naming the arguments after the first.

- confint() returns Wald (normal) intervals for the free parameters and the defined parameters, confint(fit, parm, level = 0.95); parm takes names or positions. A parameter pinned by a constraint or sitting on a bound has a zero standard error and a point interval.
- tidy() returns one row per free parameter, then the moments fixed_x fixed (a fixed_x column, only when something is fixed), then per defined parameter: term (the name coef() uses), op (the operator of its row in the parameter table, := for a defined parameter), estimate, std_error, statistic (the z value, NA when the standard error is zero), p_value (two-sided), and with conf_int = TRUE the interval confint() returns (conf_low, conf_high, at conf_level). type is "all" (default), "parameters" or "defined".
- glance() returns one row: nobs, npar (free parameters after the equality constraints), df, chisq (n times the minimized discrepancy, the test of the model against the saturated one), p_value (NA when df is 0), logLik, AIC and BIC (with the means free, as logLik() defines them) and fixed_x, the names the fit holds at their sample moments, comma-joined ("" when none). A fit with sampling weights has no likelihood, so its chisq, p_value, logLik, AIC and BIC are NA. There is no converged column: fit_sem() returns a fit only for a solution that passed the acceptance gate and errors otherwise, so @converged is TRUE for every SEMFit and would carry no information.
- summary() collects the estimates, standard errors and z statistics, the defined parameters, and the diagnostics: rows dropped, information type, degrees of freedom, retries, stationarity, optimum status, active bounds and constraints, improper-solution flags and the multiplier sign check.

Value

coef() a named numeric vector; vcov() a matrix; nobs() an integer; logLik() an object of class "logLik"; summary() an object of class "summary_SEMFit"; print() invisibly returns its argument.

See Also

fit_sem(), SEMFit

Examples

```
fit <- fit_sem(
  "mediator1 ~ a*treatment\noutcome ~ b*mediator1 + treatment\nab := a*b",
  data = mediation_demo, fixed_x = FALSE, meanstructure = NULL
)
coef(fit)
vcov(fit)[c("a", "b"), c("a", "b")]
nobs(fit)
summary(fit)
```

```
generics::tidy(fit, conf_int = TRUE)
generics::glance(fit)

# a likelihood-ratio test of b = 0 against the fitted model
fit0 <- fit_sem(paste0(fit@model, "\nb == 0"), data = mediation_demo,
  fixed_x = FALSE, meanstructure = NULL)
anova(fit, fit0)
```

`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).

Usage

```
SerialMediationData(a_path, d_path, b_path, c_prime, estimates, vcov,
  sigma_mediators, sigma_y, treatment, mediators, outcome,
  mediator_predictors, outcome_predictors, data, n_obs, converged,
  source_package)
```

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 indirect effect through the full chain is the product of the chain's 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

This is the effect through the chain only, and what `nie()` returns by default. Paths that skip a mediator (`X -> M2`, `M1 -> Y`, ...) are not part of it; `nie(x, type = "total")` adds them, and `te()` is the full total effect over every path, so `pm()` is the total indirect share of it. With treatment-by-mediator products in the outcome model, `extract_mediation()` returns a JointMediationData object with the effect through all the mediators together.

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 sum of the indirect and direct effects.

Usage

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

Arguments

`x`	A MediationData, SerialMediationData, ParallelMediationData, InteractionMediationData, or JointMediationData object.
`...`	Additional arguments passed to methods

Details

$$TE = NIE + NDE$$

For a SerialMediationData object the total effect is the sum over every directed X-to-Y path in the fitted models: the direct path, the full chain, and every path that skips a mediator (for two mediators, $c' + a_1 b_1 + a_2 b_2 + a_1 d_{21} b_2$). With the same covariates in every equation and linear models this equals the treatment coefficient of the outcome regressed on the treatment and covariates alone. A path missing from its model counts as zero; when a path is in a model but its coefficient was not recorded (a hand-built object), `te()` returns `NA` with a warning. For glm fits with a non-identity link the sum of path products is on the linear-predictor scale, as for MediationData.

Value

A numeric scalar of class `mediation_effect` carrying a `type` attribute.

See Also

`nie()`, `nde()`, `pm()`, `paths()`, `decompose()`

Examples

```
med_data <- fit_mediation(
  formula_y = outcome ~ treatment + mediator1 + covariate1 + covariate2,
  formula_m = mediator1 ~ treatment + covariate1 + covariate2,
  data = mediation_demo,
  treatment = "treatment",
  mediator = "mediator1"
)

te(med_data)

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

`tidy_s7_object` *Tidy, Glance, and Inference Methods for medfit Objects*

Description

`tidy()` converts a mediation data object or a BootstrapResult into a tidy tibble, one row per path coefficient or effect. `glance()` returns a one-row summary. The base generics `stats::coef()`, `stats::vcov()`, `stats::confint()`, and `stats::nobs()` also have methods for every mediation class; see Details.

Usage

```
tidy_s7_object(x, ...)

glance_s7_object(x, ...)
```

Arguments

`x`	A MediationData, SerialMediationData, ParallelMediationData, InteractionMediationData, JointMediationData, ClusterMediationData, BootstrapResult, or SEMFit object (a `SEMFit` has its own columns; see SEMFit-methods).
`...`	Passed to the class method: `type` (`"all"`, `"paths"`, `"effects"`, and for interaction objects `"components"`), `conf_int` (logical, add `conf_low`/`conf_high`), and `conf_level` (default 0.95).

Details

Path standard errors are the square roots of the diagonal of `@vcov`. Effect standard errors (NIE, NDE, TE, and for interaction objects the four-way components) use the delta method over the full `@vcov`, the same computation as `confint(parm = "effects")`, so `tidy(conf_int = TRUE)` reproduces `confint()` exactly. Intervals are normal approximations ($\hat{\theta} \pm z\,SE$); the sampling distribution of a product of coefficients is skewed, so for inference on indirect effects prefer `bootstrap_mediation()`. `tidy()` raises no warning about this; `confint()` does.

For a serial chain fitted as separate lm/glm regressions, `@vcov` has zero covariances between equations, and the effect standard errors inherit that. The proportion mediated (reported by `glance()`) has no standard error: it is a ratio whose delta-method standard error is unstable when the total effect is near zero, so bootstrap it instead.

Bounded FIML:

For data extracted from a bounded FIML fit, `tidy()`, coefficients, covariance and normal path/effect intervals are supported, but `glance()`, `pm()` and bootstrap intervals refuse. The bootstrap recommendations above apply only to other supported objects. `glance()` on the source SEMFit instead retains likelihood and information-criterion fields, with model tests unavailable. See `fit_sem()` for the bounded model, assumptions and inference limits.

Base methods:

- `coef(object, type = "paths")`: the path coefficients; `type = "effects"` gives the effects, `"all"` both, and for interaction objects `"components"` gives the four-way components.
- `vcov(object)`: the stored `@vcov`, covering every entry of `@estimates`.
- `confint(object, parm = "paths", level = 0.95)`: normal intervals for the paths, or with `parm = "effects"` for the effects using the delta-method standard errors above (interaction objects also accept `parm = "components"`). With `parm = "effects"` it warns that the normal approximation may be inaccurate for an indirect effect.
- `nobs(object)`: the number of observations.

Value

`tidy()`: a tibble (a data frame if tibble is not installed) with columns `term`, `estimate`, `std_error`, and, when `conf_int = TRUE`, `conf_low` and `conf_high`.

`glance()`: a one-row tibble with `nie`, `nde`, `te`, `pm`, `nobs`, and `converged`; interaction objects add `interaction` and `m_star`, and joint objects add `cde`, `structure`, `n_mediators`, `interactions`, and `m_star`.

See Also

`bootstrap_mediation()`, MediationData

Examples

```
med_data <- fit_mediation(
  formula_y = outcome ~ treatment + mediator1 + covariate1 + covariate2,
  formula_m = mediator1 ~ treatment + covariate1 + covariate2,
  data = mediation_demo,
  treatment = "treatment",
  mediator = "mediator1"
)
# tidy() is the generic from the generics package (also re-exported by broom)
generics::tidy(med_data)
generics::tidy(med_data, type = "effects", conf_int = TRUE)
```

Index

∗ **datasets**
mediation_demo, 41

bootstrap_mediation, 3
bootstrap_mediation(), *14, 15*, *20*, *29*, *35*, *46*, *50*, *60, 61*
BootstrapResult, *5*, 7, *46*, *60*

ClusterMediationData, 8, *14, 15*, *17*, *60*
compare_lr, 10
confint(), *14*, *46*

decompose, 12
decompose(), *45, 46*, *59*

extract_mediation, 13
extract_mediation(), *17, 18*, *20*, *57*

fit_mediation, 16
fit_mediation(), *5*, *15*, *23, 24*, *26*, *35, 36*
fit_sem, 21
fit_sem(), *4*, *11, 12*, *14*, *17, 18*, *32–34*, *37, 38*, *40, 41*, *50–55*, *61*

glance_s7_object *(*tidy_s7_object*)*, 60

InteractionMediationData, *3*, *12*, *14, 15*, *18–20*, 27, *45, 46*, *49, 50*, *59, 60*

joint_effects, 29
joint_effects(), *32*
JointMediationData, *3*, *12*, *14, 15*, *29*, 30, *42*, *45, 46*, *49, 50*, *57*, *59, 60*

mbco_sem, 32
mbco_sem(), *11, 12*, *23*, *26*, *40*, *54, 55*
med, 35
med(), *51, 52*
medfit-syntax, 37
medfit_syntax *(*medfit-syntax*)*, 37
mediation_demo, 41
MediationData, *3*, *5*, *14, 15*, *19, 20*, 43, *45–47*, *49–51*, *59–61*

nde, 45
nde(), *13, 14*, *36*, *46*, *49, 50*, *52*, *59*
nie, 46
nie(), *13, 14*, *36*, *45*, *49, 50*, *52*, *57*, *59*

ParallelMediationData, *3*, *14, 15*, *45, 46*, 47, *49, 50*, *59, 60*
paths, 48
paths(), *45, 46*, *50*, *59*
pm, 50
pm(), *36*, *45, 46*, *49*, *52*, *59*

quick, 51
quick(), *36*

SEMFit, *11*, *13*, *21*, *23*, *26*, *33, 34*, *41*, 52, *53*, *55*, *60, 61*
SEMFit-methods, *23*, 53, *60*
SerialMediationData, *3*, *14, 15*, *31*, *45–47*, *49–51*, 56, *59, 60*
stats::anova(), *23*
stats::coef(), *60*
stats::confint(), *60*
stats::glm(), *18, 19*
stats::nobs(), *60*
stats::vcov(), *15*, *60*

te, 59
te(), *13, 14*, *36*, *45, 46*, *49, 50*, *57*
tidy_s7_object, 60