Package: missingmed (via r-universe)

October 18, 2026

Type Package

Title Mediation Analysis with Multiple Imputation for Missing Data

Version 0.10.0

Date 2026-10-14

Depends R (>= 4.1.0), methods (>= 4.1.0)

Imports S7 (>= 0.2.0), medfit (>= 0.3.1), RMediation (>= 1.5.0), mice (>= 3.18.0), lavaan (>= 0.7-3), tibble (>= 3.0.0), broom (>= 0.7.0), rlang (>= 0.4.0), lifecycle (>= 1.0.3), sandwich

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

VignetteBuilder knitr, rmarkdown

Language en-US

Config/testthat/edition 3

Description Mediation analysis with incomplete data. Fits the mediator and outcome regressions on each dataset imputed with 'mice' (through 'medfit'), pools them with Rubin's rules, and infers the indirect effect with a Monte Carlo confidence interval ('RMediation') or a D4-stacked model-based constrained optimization (MBCO) likelihood-ratio test. Also provides an inverse probability weighting estimator, treatment-by-mediator interaction models, and a delta-adjustment sensitivity analysis for departures from missing at random. Structural equation models, including a latent mediator, are fit with 'lavaan'.

License GPL (>= 3)

Encoding UTF-8

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

URL https://github.com/Data-Wise/missingmed, https://data-wise.github.io/missingmed/

BugReports https://github.com/Data-Wise/missingmed/issues

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake make libicu-dev libx11-dev zlib1g-dev

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

Date/Publication 2026-10-14 21:54:04 UTC

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

RemoteRef main

RemoteSha ed28304c2276394dc1ee489d4896be6d6271ab2d

Contents

infer . . . 2
mbco_d4 . . . 4
MbcoMIResult . . . 7
MDMediationData . . . 8
MDMediationFit . . . 10
MDMediationResult . . . 11
MDSensitivityResult . . . 12
n_imp . . . 14
n_imputations . . . 14
per_imputation_list . . . 15
pool . . . 16
run . . . 17
sensitivity_mnar . . . 18
set_md_mediation . . . 21

Index **25**

`infer` *Inference on the indirect effect under multiple imputation*

Description

Computes inference for the indirect (mediated) effect from a fitted missingmed pipeline, dispatching to one of two engines:

Usage

```
infer(object, ...)
```

Arguments

`object`	An MDMediationFit (supports both `"mc"` and `"mbco"`) or an MDMediationResult (supports `"mc"`).
`...`	Method arguments: `type` (inference type, `"mc"` (default) or `"mbco"`), `level` (confidence level for `"mc"`; defaults to the object's `@conf_level`, itself `0.95` unless set in `set_md_mediation()`), `n.mc` (Monte-Carlo draws for `"mc"`, default `1e5`), and `ariv` (for `"mbco"`: `"fixed"` (default) tests every imputation on the

branch the stacked constrained fit chose; "own" uses each imputation's own winning branch, matches the D4 and r4 of missingmed 0.4.0 on full-rank designs, and can be liberal at high missingness; see mbco_d4()), and treatment_level (for "mc" on a model with an X:M term, and only there: the treatment level x at which the indirect effect $a(b + \theta_3 x)$ is evaluated, that is, the effect of a one-unit increase in X through M with X held at x in the outcome model. For a 0/1 treatment, 1 gives the total natural indirect effect and 0 the pure natural indirect effect. Required for such models; an error otherwise). Any other argument is an error, so a misspelled one (say conf.level) is not silently dropped.

Details

- type = "mc" — Monte-Carlo / distribution-of-the-product confidence interval via RMediation::ci_mediation_data() applied to the **pooled** named medfit::MediationData. When the outcome model has a treatment-by-mediator interaction (Y ~ X * M + ...), the indirect effect is $a(b + \theta_3 x)$, which depends on the treatment level x; set treatment_level to choose it. The interval then comes from RMediation::ci() on the pooled estimates and pooled covariance of (a, b, θ_3). With a non-identity link (family_m or family_y binomial, poisson, ...) the interval is on that link scale, and a warning of class missingmed_link_scale_warning says so once per session.
- type = "mbco" — **D4-stacked MBCO** likelihood-ratio test of $H_0 : ab = 0$, computed from the per-imputation datasets (MBCO does not commute with Rubin's rules; see per_imputation_list()). The engine is mbco_d4(); see there for ariv, the branch diagnostics and the cost. Every imputation is refit with an unweighted stats::glm(), so weights, offset, subset and na.action in fit_args are an error here; use type = "mc" for such fits. At least two imputations are required. It also works for engine = "lavaan" fits (including a latent mediator), with lavaan doing the refits: the tested paths are the structural regressions mediator ~ treatment and outcome ~ mediator. Only estimator = "ML" (the default) is supported; MLR, MLM, WLSMV, group, ordered, sampling.weights and method = "ipw" are refused, naming the option. A refit that did not converge stops the test, naming the dataset, the branch and the model.

Value

For "mc", the list returned by RMediation::ci_mediation_data() (CI, Estimate, SE, MC.Error); for a model with an X:M term, the same elements plus Estimand, the formula the interval is for. For "mbco", an MbcoMIResult: the named numeric c(D4, p, r4, nu, d_S) (index it with r["p"] or r[["p"]]) with the branch diagnostics as properties.

See Also

run(), pool(), per_imputation_list(), mbco_d4()

Examples

```
set.seed(1)
n <- 200
d <- data.frame(X = rnorm(n), C = rnorm(n))
d$M <- 0.4 * d$X + 0.3 * d$C + rnorm(n)
d$Y <- 0.3 * d$M + 0.2 * d$X + 0.3 * d$C + rnorm(n)
d$M[sample(n, 30)] <- NA
```

```
d$Y[sample(n, 30)] <- NA
imp <- mice::mice(d, m = 3, method = "norm", printFlag = FALSE, seed = 1)
fit <- run(set_md_mediation(imp, Y ~ X + M + C, M ~ X + C,
  treatment = "X", mediator = "M"
))
r <- infer(fit, type = "mbco", ariv = "fixed")
r
r[["p"]]
```

`mbco_d4` *D4-stacked MBCO test of an indirect effect across imputed datasets*

Description

Tests $H_0 : ab = 0$ with the model-based constrained optimization (MBCO) likelihood-ratio statistic, pooled across multiply imputed datasets with the D4 rule (Chan & Meng, 2022; Grund, Lüdtke & Robitzsch, 2023). This is the engine behind `infer(<MDMediationFit>, type = "mbco")`, exported so that other packages can call it on a plain list of completed datasets.

Usage

```
mbco_d4(
  implist,
  formula_y,
  formula_m,
  family_y = stats::gaussian(),
  family_m = stats::gaussian(),
  treatment,
  mediator,
  ariv = c("fixed", "own"),
  model = NULL,
  outcome = NULL,
  fit_args = list()
)
```

Arguments

`implist` A list of at least two completed data frames, e.g. `mice::complete(imp, "all")`. They must share the same columns (in any order) and number of rows, contain the treatment and mediator, and have no missing values in the model variables.

`formula_y`, `formula_m` Outcome and mediator model formulas. The response of `formula_m` must involve the mediator and nothing else (`M` or a transform such as `log(M)`), and the mediator may not appear in the response of `formula_y`.

`family_y`, `family_m` `stats::family()` objects for the two models (default `stats::gaussian()`); models are fit with `stats::glm()`.

`treatment`, `mediator`
: Names of the treatment and mediator variables.

`ariv`
: `"fixed"` (default) or `"own"`; see Details.

`model`, `outcome`, `fit_args`
: For a lavaan SEM, in place of `formula_y`, `formula_m` and the families: a lavaan model syntax string, the outcome variable name, and a named list of lavaan options (as in `set_md_mediation()`). `model` cannot be combined with the formula or family arguments. The tested paths are the regressions `mediator ~ treatment` (a) and `outcome ~ mediator` (b); a latent mediator uses its structural rows, and its measurement model and any direct effects of its indicators on the outcome stay free. Only `estimator = "ML"` (the default) is supported, with no `group`, `ordered` or `sampling.weights`.

Details

The MBCO constraint is a branch union, $\max(\ell_{a=0}, \ell_{b=0})$, and nulling a path drops every term that carries it (for example both `M` and `X:M` from `Y ~ X * M`). The D4 statistic is $D_4 = d_S/(k(1 + r_4))$, referred to $F(k, \nu)$, where d_S is the statistic on the stacked data divided by K and r_4 is the relative increase in variance estimated from the per-imputation statistics, and $\nu = k(K-1)(1 + 1/r_4)^2$ (Chan and Meng 2022, eq. 2.15 of arXiv:1711.08822; the same reference distribution as `mitml::testModels(method = "D4", ariv = "positive")`, since r_4 is floored at 0). Chan and Meng derive this df for the own-branch statistic at a regular null. For the fixed-branch default, and at the nonregular null $a = b = 0$, it is a working reference distribution whose calibration rests on simulation (see `vignette("mbco-mi")`, *Operating characteristics*). `ariv` chooses how those statistics are formed:

- `"fixed"` (default, calibrated): each imputation's statistic is computed on the branch (`a = 0` or `b = 0`) that the **stacked** constrained fit selected. Imputations that disagree on the winning branch then cannot pull r_4 down, and every imputation uses the stacked fit's `k`. An error is raised if that branch's constraint removes a different number of parameters in some imputation than in the stacked data (for example, a level of a factor that interacts with the treatment or mediator is absent from one imputation). `k` is a difference of design-matrix ranks, so a sparse level of a main-effect factor does not trigger it.
- `"own"`: each imputation's statistic is computed on its own winning branch (the standard Chan & Meng r_4). On full-rank designs its `D4` and `r4` equal missingmed 0.4.0's; `nu` and `p` differ slightly, because the denominator degrees of freedom now follow Chan & Meng (see NEWS). `k` is a rank difference rather than a column count, so a design with aliased columns gets a smaller `k`. It errors when the winning branches remove different numbers of parameters, since there is then no single `k`. **It can be liberal.** In simulation (1000 replications per cell, 20 imputations, MAR missingness in the mediator, 5% level) it rejected a true null up to 8.3% of the time for a plain Gaussian model with 40% missing and n = 200, and up to 13.8% for binary and count outcomes at 40% missing, where `"fixed"` stayed at or below 5.7% on the same data. Use `"fixed"` unless you need to reproduce an earlier analysis.

Cost. Every imputation is fit three times (the full model and both single-path nulls), plus the same three fits on the stacked data. The `ariv = "fixed"` statistic alone needs only the null on the stacked branch; the second null per imputation is what the branch diagnostics `branch_mix` and `p_branch_a` require.

Convergence. A `stats::glm()` refit that did not converge, or whose log-likelihood is not finite, stops the test with an error naming the dataset (an imputation or the stacked data), the branch (`full`, `a = 0` or `b = 0`) and the model: its likelihood is not a maximum, so the statistic would not be a likelihood ratio. A fit that converged but carries glm's "fitted probabilities numerically 0 or 1" warning is not refused (its likelihood is finite and the warning reaches you unchanged), but the chi-square-type reference may be poor for such near-separated data.

At least two imputations are required. For a single complete dataset, use a complete-data MBCO test such as `RMediation::mbco()`.

Value

An MbcoMIResult: the named numeric `c(D4, p, r4, nu, d_S)` with the branch diagnostics as properties.

References

Chan, K. W., & Meng, X.-L. (2022). Multiple improvements of multiple imputation likelihood ratio tests. *Statistica Sinica*.

Grund, S., Lüdtke, O., & Robitzsch, A. (2023). Pooling methods for likelihood ratio tests in multiply imputed data sets. *Psychological Methods*, *28*(5), 1207–1221. doi:10.1037/met0000556

See Also

`infer()`, MbcoMIResult

Examples

```
set.seed(1)
implist <- lapply(1:3, function(i) {
  n <- 200
  X <- rnorm(n)
  M <- 0.4 * X + rnorm(n)
  data.frame(X = X, M = M, Y = 0.3 * M + rnorm(n))
})
mbco_d4(implist, Y ~ X + M, M ~ X,
  treatment = "X", mediator = "M", ariv = "fixed"
)

# The same test for a lavaan SEM, from a model string
mbco_d4(implist,
  model = "M ~ X\nY ~ M + X", treatment = "X", mediator = "M", outcome = "Y"
)
```

MbcoMIResult *MbcoMIResult: D4-stacked MBCO test result (S7)*

Description

The value of `infer(<MDMediationFit>, type = "mbco")` and `mbco_d4()`, for the glm and the lavaan (maximum likelihood) engines alike. It is an S7 object whose parent is `class_double`: its data is the named numeric vector `c(D4, p, r4, nu, d_S)`, and the properties carry how it was computed and the branch diagnostics.

Usage

```
MbcoMIResult(
  .data = numeric(0),
  ariv = character(0),
  k = integer(0),
  m = integer(0),
  stacked_branch = character(0),
  branch_mix = logical(0),
  p_branch_a = integer(0),
  scaling = NA_real_
)
```

Arguments

`.data`	Named numeric `c(D4, p, r4, nu, d_S)`.
`ariv`	`"fixed"` or `"own"`: which per-imputation statistics fed `r4` (see `mbco_d4()`).
`k`	Number of parameters the constraint removes (numerator df).
`m`	Number of imputations pooled.
`stacked_branch`	`"a"` or `"b"`: the branch (`a = 0` or `b = 0`) chosen by the constrained fit on the stacked data.
`branch_mix`	Logical; `TRUE` when the imputations disagree on their own winning branch.
`p_branch_a`	Share of imputations whose own winning branch is `a = 0`.
`scaling`	`NA` for every public result. Internal: the mean scaling correction by which `D4` was divided, set only when the MLR gate harness enables the (publicly refused) MLR path; `d_S` stays the unscaled stacked statistic.

Details

- `D4`: the pooled statistic, referred to $F(k, \nu)$.
- `p`: its p-value.
- `r4`: the relative increase in variance, clamped at 0.
- `nu`: the denominator degrees of freedom (`Inf` when `r4` is 0).

- d_S: the MBCO statistic on the stacked data divided by the number of imputations.

x["p"], x[["p"]] and is.numeric(x) work as they did on the plain vector returned by missingmed 0.4.0 and earlier, and S7::S7_data(x) returns that vector. Two things differ: identical(x, old_vector) is FALSE, and x$p is an error.

Value

An MbcoMIResult S7 object.

See Also

infer(), mbco_d4()

Examples

```
r <- MbcoMIResult(c(D4 = 4.7, p = 0.048, r4 = 1.16, nu = 13.8, d_S = 10.2),
  ariv = "fixed", k = 1, m = 5, stacked_branch = "b",
  branch_mix = TRUE, p_branch_a = 0.4
)
r[["p"]]
r@stacked_branch
```

MDMediationData *MDMediationData: imputed data + mediation specification (S7)*

Description

An S7 class holding multiply imputed data together with a **medfit-style mediation specification** (outcome/mediator formulas + roles). It is the entry point of the missingmed S7 pipeline (set_md_mediation() -> run() -> pool() -> infer()) and the S7 successor of the removed S4 SemImputedData class.

Usage

```
MDMediationData(
  data = NULL,
  formula_y = NULL,
  formula_m = NULL,
  treatment = character(0),
  mediator = character(0),
  engine = "glm",
  family_y = NULL,
  family_m = NULL,
  method = "mi",
  mechanism = "mar",
  weight_formula = NULL,
  weight_stabilize = TRUE,
  weight_trim = 1,
```

```
  se_type = "sandwich",
  conf_int = FALSE,
  conf_level = 0.95,
  n_imputations = integer(0),
  model = character(0),
  outcome = character(0),
  fit_args = list(),
  original_data = data.frame()
)
```

Arguments

`data`	For `method = "mi"`, an object of class `mids` from the `mice` package; for `method = "ipw"`, a raw `data.frame` (the complete cases are reweighted).
`formula_y`	Outcome model formula (e.g. `Y ~ X + M + C`).
`formula_m`	Mediator model formula (e.g. `M ~ X + C`).
`treatment`	Name of the treatment/exposure variable.
`mediator`	Name of the mediator variable.
`engine`	medfit fitting engine, e.g. `"glm"` (default).
`family_y, family_m`	`stats::family` objects for the outcome and mediator models. Default `stats::gaussian()`.
`method`	Estimator axis: `"mi"` (default) or `"ipw"`.
`mechanism`	Assumed missing-data mechanism: `"mar"` (default) or `"mnar"`.
`weight_formula`	(IPW) Missingness model specification: `NULL` (default; a sequential model on the fully observed predictors), a single `formula` (one joint complete-case model), or a named `list` of formulas (a sequential model, in list order). Ignored for MI. See `set_md_mediation()`.
`weight_stabilize`	(IPW) Logical; if `TRUE` (default) use stabilized weights `P(R=1|X) / P(R=1|Z)`. Ignored for MI.
`weight_trim`	(IPW) Upper quantile at which to cap weights (e.g. `0.99`); `1` (default) disables trimming. Ignored for MI.
`se_type`	(IPW) Variance estimator passed to `medfit::fit_mediation()`: `"sandwich"` (default for IPW, HC robust) or `"model"`. Ignored for MI.
`conf_int`	Logical; whether downstream output carries confidence intervals. Defaults to `FALSE`.
`conf_level`	Numeric in (0, 1); confidence level. Defaults to `0.95`.
`n_imputations`	Number of imputations (MI) or `1` (IPW).
`model`	For `engine = "lavaan"`, the lavaan model syntax (a single string); `NULL`-equivalent (`character(0)`) otherwise.
`outcome`	For `engine = "lavaan"`, the name of the outcome variable.
`fit_args`	For `engine = "lavaan"`, a named list of extra arguments for `lavaan::sem()`. Empty otherwise.
`original_data`	The original data (pre-imputation for MI; the supplied frame for IPW).

Details

Fitting is delegated to `medfit::fit_mediation()` (one call per imputation), so the per-imputation fits carry **named** path coefficients (`a`, `b`, `c_prime`) ready for RMediation inference. The estimator (`method`) and the model are orthogonal axes: `method` selects the missing-data estimator (`"mi"`/`"ipw"`); the formulas/engine select the model.

Value

An `MDMediationData` S7 object.

See Also

`set_md_mediation()`, `medfit::fit_mediation()`

`MDMediationFit` *MDMediationFit: per-imputation mediation fits (S7)*

Description

An S7 class holding the result of fitting a mediation model across all imputations. Its defining feature is `per_imputation`: a list of **named** medfit::MediationData objects, one per imputation. This list is what the MBCO-MI path consumes, because MBCO does not commute with Rubin's rules (D4-stacked MBCO needs the per-imputation fits, not the pooled estimate).

Usage

```
MDMediationFit(
  per_imputation = list(),
  m = integer(0),
  engine = "glm",
  conf_int = FALSE,
  conf_level = 0.95,
  weights = NULL,
  source = NULL
)
```

Arguments

`per_imputation`	A list of named medfit::MediationData objects (length `m`).
`m`	Integer number of imputations.
`engine`	medfit fitting engine used (e.g. `"glm"`).
`conf_int`	Logical; whether output carries confidence intervals.
`conf_level`	Numeric in (0, 1); confidence level.
`weights`	(IPW) Full-length numeric IPW weight vector (`NA` for dropped rows); `NULL` for MI fits.
`source`	The originating MDMediationData (retained so MBCO can refit constrained/unconstrained models against the imputed data).

Details

It is the S7 successor of the removed S4 `SemResults` class.

Value

An `MDMediationFit` S7 object.

See Also

`run()`, `per_imputation_list()`

`MDMediationResult` *MDMediationResult: pooled mediation result (S7)*

Description

An S7 class holding the Rubin's-rules pooled mediation result. Its defining feature is `pooled`: a single **named** medfit::MediationData built from the pooled estimates and total variance-covariance, valid as input to `RMediation::ci_mediation_data()` / `RMediation::medci()` (path coefficients resolve by name). It is the S7 successor of the removed S4 `PooledSEMResults` class.

Usage

```
MDMediationResult(
  pooled = NULL,
  tidy_table = data.frame(),
  cov_total = NULL,
  cov_between = NULL,
  cov_within = NULL,
  m = integer(0),
  engine = "glm",
  conf_int = FALSE,
  conf_level = 0.95
)
```

Arguments

`pooled`	A named medfit::MediationData carrying the pooled estimates and total vcov (path labels `a`, `b`, `c_prime`, ...).
`tidy_table`	A data frame of pooled estimates, one row per coefficient: `term`, `estimate`, `std_error`, `var_w`, `var_b`, `var_tot`, and the Rubin-pooled Wald test `statistic`, `df` (Barnard & Rubin, 1999, using each model's complete-data df; infinite for binomial and poisson models), `riv` (relative increase in variance), `fmi` (fraction of missing information) and `p_value`. At `m = 1` (IPW, or a single imputation) this is the ordinary single-fit Wald test, with `riv = fmi = 0`. The rows `a`, `b` and `c_prime` are aliases of `m_<treatment>`, `y_<mediator>` and `y_<treatment>`, and repeat their values. **These are tests of the individual paths, not of the**

indirect effect: a * b is nonlinear and its null is non-regular, so test mediation with infer() (Monte Carlo CI or MBCO) instead.

cov_total, cov_between, cov_within
: The Rubin's-rules total, between-, and within-imputation covariance matrices.

m
: Integer number of imputations pooled.

engine
: medfit fitting engine used (e.g. "glm").

conf_int
: Logical; whether the tidy table carries per-coefficient conf_low and conf_high columns (set in set_md_mediation()).

conf_level
: Numeric in (0, 1); confidence level.

Value

An MDMediationResult S7 object.

See Also

pool(), infer()

MDSensitivityResult *MDSensitivityResult: MNAR sensitivity curve (S7)*

Description

An S7 class holding the result of sensitivity_mnar(): one inference result per rung of a delta grid, plus the grid itself and the realized marginal sensitivity parameters.

Usage

```
MDSensitivityResult(
  rungs = list(),
  failed = character(0),
  grid = data.frame(),
  msp = numeric(),
  target = character(0),
  type = "mc",
  level = 0.95,
  seed = integer(0),
  seed_source = "argument",
  method_target = NA_character_,
  mechanism_used = NA_character_,
  scale = NA_character_,
  source = NULL
)
```

Arguments

rungs	List of inference results, one per row of grid. A rung that failed under sensitivity_mnar(on_error = "continue") is NULL.
failed	Character vector, one per rung: the error message of a failed rung and NA for a rung that succeeded. Empty (the default) when no rung failed.
grid	Data frame of delta values; one column per target, one row per rung.
msp	Numeric vector of realized marginal sensitivity parameters, one per rung, computed on the first target (target[1]).
target	Character; the shifted variable(s).
type	Inference type used for each rung ("mc" or "mbco").
level	Numeric in (0, 1); the confidence level the rungs were built at. Needed by summary() to decide whether an "mbco" rung retains the null, since that test reports a p-value rather than an interval.
seed	Integer seed pinned across rungs.
seed_source	"mids" if taken from the supplied mids, "argument" if passed explicitly, "default" if neither was available.
method_target	The mice imputation method(s) used for the target variable(s).
mechanism_used	How the delta was applied, one entry per target: "post" (drawn values shifted), "mnar.norm" or "mnar.logreg" (mice's NARFCS methods).
scale	The delta scale, one entry per target: "raw" (units of the target) or "logodds" (the mnar.logreg route).
source	The originating MDMediationData.

Value

An MDSensitivityResult S7 object.

The delta scale

@grid holds **conditional** sensitivity parameters (CSPs) – the values the user supplied. @msp holds the **marginal** sensitivity parameter actually realized at each rung: the mean difference between imputed and observed values of the target. They are not the same quantity and can differ substantially; see vignette("technical") and sensitivity_mnar().

See Also

sensitivity_mnar()

n_imp — *Get the number of imputations from a mids object*

Description

Returns the number of imputations `m` stored in a mice::mids object.

Usage

```
n_imp(x)
```

Arguments

`x`	A mice::mids object.

Value

An integer: the number of imputations.

Examples

```
imp <- mice::mice(mice::nhanes, m = 3, printFlag = FALSE, seed = 1)
n_imp(imp)
```

n_imputations — *Number of imputations*

Description

Number of imputations

Usage

```
n_imputations(object, ...)
```

Arguments

`object`	An MDMediationFit or MDMediationResult object.
`...`	Unused.

Value

Integer count of imputations.

Examples

```
set.seed(1)
n <- 150
d <- data.frame(X = rbinom(n, 1, 0.5), C = rnorm(n))
d$M <- 0.5 * d$X + 0.3 * d$C + rnorm(n)
d$Y <- 0.3 * d$M + 0.2 * d$X + 0.3 * d$C + rnorm(n)
d$M[sample(n, 25)] <- NA
imp <- mice::mice(d, m = 3, method = "norm", printFlag = FALSE, seed = 1)
md <- set_md_mediation(imp, Y ~ X + M + C, M ~ X + C,
  treatment = "X", mediator = "M"
)
fit <- run(md)
n_imputations(fit)
n_imputations(pool(fit))
```

---

per_imputation_list *Access the per-imputation mediation fits (for MBCO)*

Description

Returns the list of per-imputation **named** medfit::MediationData objects held in an MDMediationFit, together with the number of imputations `m`.

Usage

```
per_imputation_list(object, ...)
```

Arguments

`object`	An MDMediationFit object.
`...`	Unused.

Details

This accessor exists because **MBCO does not commute with Rubin's rules**: D4-stacked MBCO needs the per-imputation fits, not the pooled estimate. The list it returns is the shape consumed by `infer()(type = "mbco")` and by an external `RMediation::mbco()` MI entry point (missingmed issue #2).

Value

A list with components `per_imputation` (a length-`m` list of named medfit::MediationData) and `m` (the number of imputations).

See Also

`run()`, `infer()`

Examples

```
set.seed(1)
n <- 150
d <- data.frame(X = rbinom(n, 1, 0.5), C = rnorm(n))
d$M <- 0.5 * d$X + 0.3 * d$C + rnorm(n)
d$Y <- 0.3 * d$M + 0.2 * d$X + 0.3 * d$C + rnorm(n)
d$M[sample(n, 25)] <- NA
imp <- mice::mice(d, m = 3, method = "norm", printFlag = FALSE, seed = 1)
md <- set_md_mediation(imp, Y ~ X + M + C, M ~ X + C,
  treatment = "X", mediator = "M"
)
fit <- run(md)
pl <- per_imputation_list(fit)
pl$m
# The a path in each imputation, before pooling
vapply(pl$per_imputation, function(x) x@a_path, numeric(1))
```

`pool` *Pool per-imputation mediation fits with Rubin's rules*

Description

Applies Rubin's (1987) rules to the list of per-imputation **named** medfit::MediationData objects in an MDMediationFit, producing a single pooled named medfit::MediationData (the `pooled` slot of the returned MDMediationResult). Because the estimates and variance-covariance carry the mediation path names (`a`, `b`, `c_prime`, ...), the pooled object is valid input to `RMediation::ci_mediation_data()` / `RMediation::medci()`. For a model with a treatment-by-mediator interaction, use `infer()(type = "mc", treatment_le` instead: those RMediation functions use only ab, the indirect effect at treatment level 0.

Usage

```
pool(object, ...)
```

Arguments

`object`	An MDMediationFit object. Anything else (a `mice::mira`, say) is forwarded to `mice::pool()`.
`...`	Unused.

Details

Pooling math (migrated from the S4 `pool_sem` / `pool_tidy` / `pool_cov`):

$$\bar{Q} = \frac{1}{m}\sum_i Q_i, \quad \bar{U} = \frac{1}{m}\sum_i U_i, \quad B = \mathrm{cov}(Q_1, \ldots, Q_m), \quad T = \bar{U} + (1 + 1/m)B.$$

It is the S7 successor of the S4 `pool_sem()` method.

The returned tidy table also carries a per-coefficient Wald test (`statistic`, `df`, `riv`, `fmi`, `p_value`) and, when `conf_int = TRUE` was set in `set_md_mediation()`, per-coefficient `conf_low` and `conf_high` at `conf_level` on the same t reference; see MDMediationResult for the columns and why they do not test or bound the indirect effect.

Value

An MDMediationResult object.

References

Rubin, D. B. (1987). *Multiple Imputation for Nonresponse in Surveys*. Wiley.

Barnard, J., & Rubin, D. B. (1999). Small-sample degrees of freedom with multiple imputation. *Biometrika*, 86(4), 948–955.

See Also

`run()`, `infer()`

Examples

```
set.seed(1)
n <- 150
d <- data.frame(X = rbinom(n, 1, 0.5), C = rnorm(n))
d$M <- 0.5 * d$X + 0.3 * d$C + rnorm(n)
d$Y <- 0.3 * d$M + 0.2 * d$X + 0.3 * d$C + rnorm(n)
d$M[sample(n, 25)] <- NA
imp <- mice::mice(d, m = 3, method = "norm", printFlag = FALSE, seed = 1)
md <- set_md_mediation(imp, Y ~ X + M + C, M ~ X + C, conf_int = TRUE,
  treatment = "X", mediator = "M"
)
res <- pool(run(md))
res
# Per-coefficient table, with Rubin's df and conf_low/conf_high
res@tidy_table[, c("term", "estimate", "std_error", "df", "conf_low", "conf_high")]
```

`run` *Fit the mediation model across imputations*

Description

Runs the mediation specification held in an MDMediationData object on every imputed dataset, delegating each fit to `medfit::fit_mediation()`. The result is an MDMediationFit whose `per_imputation` slot is a list of **named** medfit::MediationData objects (one per imputation) — the shape consumed by both Rubin's-rules pooling (`pool()`) and D4-stacked MBCO (`infer()`).

Usage

```
run(object, ...)
```

Arguments

`object`	An MDMediationData object.
`...`	**[Deprecated]** Additional arguments forwarded to `medfit::fit_mediation()`; set them with `fit_args` in `set_md_mediation()` instead. They are still honored, with a warning, and may not repeat a name already in `fit_args`.

Details

An engine error is rethrown with the engine and the imputation it failed on. Engine warnings (a `glm` that did not converge, fitted probabilities of 0 or 1) are collected and raised once, naming the imputations that produced them.

It is the S7 successor of the S4 `run_sem()` method.

Value

An MDMediationFit object.

See Also

`set_md_mediation()`, `pool()`, `infer()`

Examples

```
set.seed(1)
n <- 150
d <- data.frame(X = rbinom(n, 1, 0.5), C = rnorm(n))
d$M <- 0.5 * d$X + 0.3 * d$C + rnorm(n)
d$Y <- 0.3 * d$M + 0.2 * d$X + 0.3 * d$C + rnorm(n)
d$M[sample(n, 25)] <- NA
imp <- mice::mice(d, m = 3, method = "norm", printFlag = FALSE, seed = 1)
md <- set_md_mediation(imp, Y ~ X + M + C, M ~ X + C,
  treatment = "X", mediator = "M"
)
fit <- run(md)
fit
```

`sensitivity_mnar` *MNAR sensitivity analysis by delta-adjusted imputation*

Description

Re-imputes the data across a grid of delta values and re-runs the mediation pipeline at each rung, producing a **sensitivity curve** for the indirect effect. It is not an estimator: MAR versus MNAR is not testable from observed data, so nothing here is identified. A rung answers "if the unobserved values of `target` sit delta units away from what MAR imputation implies, the indirect effect is X".

Usage

```
sensitivity_mnar(
  object,
  delta,
  target = NULL,
  type = c("mc", "mbco"),
  seed = NULL,
  level = NULL,
  n.mc = 1e+05,
  ums = NULL,
  treatment_level = NULL,
  ariv = c("fixed", "own"),
  on_error = c("stop", "continue"),
  ...
)
```

Arguments

`object` An MDMediationData with `method = "mi"`.

`delta` Numeric vector (one rung per value, applied to `target`), or a data frame (one rung per row, one column per target variable). Supply either `delta` or `ums`, not both.

`target` Name of the variable to shift. Defaults to the mediator. Must be `NULL` when `delta` is a data frame.

`type` Inference per rung: `"mc"` (default) or `"mbco"`. `"mbco"` works for `engine = "glm"` and for `engine = "lavaan"` (maximum likelihood only, see `infer()`). If a refit does not converge at a rung, the sweep stops by default and the error names the rung and its delta (see `on_error`).

`seed` Integer seed pinned across rungs. Defaults to the seed stored in the `mids` object, or `20260822L` when that is `NA`. A fractional value is truncated, as `set.seed()` does, and the result records the integer used.

`level`, `n.mc`, `treatment_level`
Passed to `infer()` for `type = "mc"`. `treatment_level` is required when the outcome model has a treatment-by-mediator interaction and an error otherwise. For `type = "mbco"`, `level` sets the test size used to find the tipping point (p > 1 - `level` retains the null), and `n.mc` and `treatment_level` do not apply (a warning names them if supplied).

`ums` Optional character vector for a **covariate-varying** delta, one rung per string, passed verbatim to `mice`'s NARFCS `ums` (e.g. `"1 + 0.5*C"`: the offset is 1 + 0.5 C per row). Each string needs exactly one intercept term. Only for a single target routed to `mnar.norm` or `mnar.logreg`. A `ums` grid has no numeric ordering, so `summary()` does not compute a tipping point for it.

`ariv` For `type = "mbco"`: passed to `infer()` for every rung (`"fixed"`, the default, or `"own"`; see `mbco_d4()`). Ignored, with a warning, for `type = "mc"`.

`on_error` **[Experimental]** What to do when fitting or inference fails at a rung (a refit that did not converge, a non-finite likelihood). `"stop"` (default) aborts the sweep

and the error names the rung and its delta. `"continue"` keeps the rungs that succeeded, records the message of each failed rung in `@failed` (and in a column `error` of `tidy()`), and warns once naming the failed rungs; a failed rung reads as `NA` in `tidy()` and in the tipping point, which is then reported as undetermined if a failed rung could change it. It is an error if every rung fails. Re-imputation errors and a non-finite shifted value always stop. Only errors are caught; warnings pass through.

`...` **[Deprecated]** Extra arguments for the engine; set them with `fit_args` in `set_md_mediation()`, which every refit here reads. They are still honored, with one warning, and may not repeat a name already in `fit_args`.

Value

An MDSensitivityResult.

Method

Delta-adjusted imputation in the pattern-mixture sense (van Buuren, *FIMD* §9.2; Leacy et al. 2017). For a continuous target the canonical procedure imputes under MAR and then adds the constant to the imputed values (Hayati Rezvan et al. 2018). How the delta enters depends on the target's imputation method:

- `"norm"` with a `ums` string: delegated to `mice`'s NARFCS method `mnar.norm` (Tompsett et al. 2018; Moreno-Betancur, van Buuren & White 2020), delta in raw units. A numeric `delta` on a `norm` target uses the `post` shift below – for a constant delta the two give identical draws.
- `"logreg"` (a binary target): delegated to `mnar.logreg`, which offsets the imputation model's linear predictor – delta on the **log-odds** scale.
- anything else continuous (`pmm`, `norm.nob`, `cart`, ...), and `norm` with a numeric `delta`: the drawn values are shifted through `mice`'s `post` argument, delta in raw units.

Each rung re-imputes from the `mids` object's stored settings – never from its recorded `call`, which does not resolve outside the function that built it.

The delta scale (read this before choosing a value)

`delta` is a **conditional** sensitivity parameter (CSP): a difference conditional on all remaining variables and their missingness indicators. The quantity an analyst can actually reason about – "non-respondents average delta units higher" – is a **marginal** sensitivity parameter (MSP), and the two are different numbers. Supplying an elicited MSP as if it were a CSP is the standard failure mode of this method and can badly damage coverage (Tompsett et al. 2018). This function therefore reports the **realized MSP** at every rung; compare it against what you meant.

Limitations

- Only `method = "mi"`. IPW has no imputations to shift.
- A numeric 0/1 target whose imputations are themselves 0/1 (`pmm`, `cart`, `sample`, ...) is refused unless it is imputed by `logreg`: an added delta would turn its values into 1s and 2s. A normal-model (`norm*`) imputation of a 0/1 variable is continuous and is allowed; its delta is on the raw (probability) scale, so keep it small and check the realized `msp`.

- Categorical targets: binary via `logreg` only. Multinomial and ordinal targets (`polyreg`, `polr`, `lda`) are refused – `mice` has no NARFCS method for them – and so is `logreg.boot`, which has no counterpart.
- With `pmm` (mice's default), shifted values may fall outside the observed range that `pmm` otherwise guarantees. A message is emitted once.
- The curve assumes the supplied imputation model is compatible with the mediation model; missingmed cannot verify this.

See Also

`infer()`, MDSensitivityResult

Examples

```
set.seed(1)
n <- 150
d <- data.frame(X = rbinom(n, 1, 0.5), C = rnorm(n))
d$M <- 0.5 * d$X + 0.3 * d$C + rnorm(n)
d$Y <- 0.3 * d$M + 0.2 * d$X + 0.3 * d$C + rnorm(n)
d$M[sample(n, 25)] <- NA
imp <- mice::mice(d, m = 3, method = "norm", printFlag = FALSE, seed = 1)
md <- set_md_mediation(imp, Y ~ X + M + C, M ~ X + C,
  treatment = "X", mediator = "M"
)
# Shift the imputed mediator values by 0, 0.5 and 1 (on the mediator's
# scale) and recompute the Monte Carlo interval at each rung.
sens <- sensitivity_mnar(md, delta = c(0, 0.5, 1), n.mc = 1e3)
sens
```

`set_md_mediation` *Set up a mediation analysis with missing data (MI or IPW)*

Description

Constructs an MDMediationData object: the entry point of the missingmed S7 pipeline. It records a **medfit-style mediation specification** (outcome and mediator formulas plus the treatment/mediator roles) together with the data. Fitting is delegated to `medfit::fit_mediation()` downstream by `run()`. It is the S7 successor of the S4 `set_sem()` constructor (removed in 0.7.0).

Usage

```
set_md_mediation(
  data,
  formula_y,
  formula_m,
  treatment,
  mediator,
```

```
  engine = "glm",
  family_y = stats::gaussian(),
  family_m = stats::gaussian(),
  method = c("mi", "ipw"),
  mechanism = c("mar", "mnar"),
  weight_formula = NULL,
  weight_stabilize = TRUE,
  weight_trim = 1,
  se_type = c("sandwich", "model"),
  conf_int = FALSE,
  conf_level = 0.95,
  model = NULL,
  outcome = NULL,
  fit_args = list()
)
```

Arguments

`data` For `method = "mi"`, a mice::mids object; for `method = "ipw"`, a `data.frame` (may contain `NA`s; complete cases are reweighted).

`formula_y` Outcome model formula (e.g. `Y ~ X + M + C`). Not used with `engine = "lavaan"`.

`formula_m` Mediator model formula (e.g. `M ~ X + C`). Not used with `engine = "lavaan"`.

`treatment` Name of the treatment/exposure variable.

`mediator` Name of the mediator variable.

`engine` Fitting engine: `"glm"` (default), `"regmedint"` (needs medfit 0.4.0 or later and the regmedint package; `method = "mi"` only), or `"lavaan"` (a structural equation model given as `model` syntax instead of formulas; with `method = "ipw"` too: complete cases are weighted and the SEs are robust).

`family_y, family_m` `stats::family` objects for the outcome and mediator models. Default `stats::gaussian()`.

`method` Estimator axis: `"mi"` (default) or `"ipw"`.

`mechanism` **[Deprecated]** The pipeline estimates under MAR regardless, so this argument never changed behavior. Passing `"mnar"` warns and is ignored. Use `sensitivity_mnar()` to assess departures from MAR; it sets `mechanism = "mnar"` on the objects it creates.

`weight_formula` (IPW) Missingness model: `NULL` (default), a single `formula`, or a named `list` of per-variable formulas. `NULL` and a list fit a sequential (chain-rule) model, P(V1 observed) x P(V2 observed | V1 observed) x ..., each factor on the rows where the earlier variables are observed. `NULL` orders the incomplete model variables by ascending share of missing values and predicts with the fully observed model variables; a list gives the order (its names) and the predictors (its formulas). A single formula fits **one** model for "every model variable is observed", which is right when the variables go missing together and biased when they go missing separately. With one incomplete variable the three agree.

`weight_stabilize` (IPW) Use stabilized weights? Default `TRUE`.

weight_trim	(IPW) Upper quantile to cap weights; 1 (default) = none.
se_type	(IPW) "sandwich" (default, HC robust) or "model".
conf_int	Logical; if TRUE, pool() adds per-coefficient conf_low and conf_high columns to the pooled tidy table, at conf_level on Rubin's t reference. Defaults to FALSE. These bound single coefficients; for the indirect effect use infer().
conf_level	Numeric in (0, 1); confidence level for conf_int and the default level of infer(). Defaults to 0.95.
model	(engine = "lavaan") lavaan model syntax, a single string, for example "M ~ a*X + C\nY ~ b*M + cp*X + C". A latent mediator ("Mlat =~ m1 + m2 + m3") is allowed. Must be NULL for other engines. The paths are found by role (treatment, mediator, outcome), not by label. The labels a, b and c_prime are the names the pooled estimates use for the three structural paths, so each may only label its own path (mediator ~ treatment, outcome ~ mediator, outcome ~ treatment); on any other path it is an error.
outcome	Name of the outcome variable. Required for engine = "lavaan", where it must be regressed on the mediator in model. For the other engines it is optional: it defaults to the response of formula_y and, when given, must match it.
fit_args	Named list of extra arguments for the engine, stored on the object and used by run() and sensitivity_mnar(). For engine = "lavaan" they go to lavaan::sem(), for example list(estimator = "MLR"), and cannot set model or data. For the other engines they go to medfit::fit_mediation() (for example list(engine_args = list(...)) with a medfit that has it) and cannot restate what this function sets (formula_y, formula_m, data, treatment, mediator, engine, family_y, family_m).

Details

Two estimators share the interface (method):

- "mi" — data is a mice::mids object; run() fits every imputation.
- "ipw" — data is a raw data.frame; run() reweights the complete cases by inverse missingness probability and fits once.

The model is validated before anything is fit. Formulas are first expanded against the data, so Y ~ . is checked as the model that run() fits. set_md_mediation() refuses:

- a one-sided formula, or a treatment/mediator that is not a single variable name;
- a variable in either formula that is neither a column of the data nor defined in the formula's environment (the treatment and mediator must be columns);
- a formula_m whose response is not the bare mediator column (a transform such as log(M) needs its own column), or a formula_y whose response involves the mediator;
- a treatment that is not a main effect of formula_m and of formula_y, or a mediator that is not a main effect of formula_y;
- any other term involving the treatment or mediator. Both enter only as main effects, plus, in formula_y only, one treatment-by-mediator interaction (X:M, M:X or from X * M). Products such as X:C, M:W or X:M:W, transforms such as I(X^2), poly(X, 2) or log(M), and offsets involving either variable are refused. For moderated models, mbco_d4() tests the indirect effect on the completed datasets;

- an `X:M` term when `family_y` or `family_m` is not Gaussian with an identity link;
- a treatment column that is not numeric. Factor, character and logical treatments must be recoded to numeric (0/1 for a binary treatment);
- a `treatment` or `mediator` that is not a syntactic R name (such as `"my M"`), which medfit cannot fit. Covariates may have any name;
- an `engine` other than `"glm"`, or `"regmedint"` with medfit >= 0.4.0 and `method = "mi"` (regmedint takes no case weights);
- a `weight_formula` that is not `NULL`, a formula or a named list of formulas, or that uses a variable found neither in the data nor in its environment.

Value

An MDMediationData object.

See Also

MDMediationData, `run()`, `pool()`, `infer()`, `medfit::fit_mediation()`

Examples

```
set.seed(1)
d <- data.frame(X = rbinom(200, 1, .5), C = rnorm(200))
d$M <- .5 * d$X + .3 * d$C + rnorm(200)
d$Y <- .2 * d$X + .4 * d$M + .3 * d$C + rnorm(200)
d$M[sample(200, 30)] <- NA
# MI
imp <- mice::mice(d, m = 5, printFlag = FALSE, seed = 1)
md_mi <- set_md_mediation(imp, Y ~ X + M + C, M ~ X + C,
  treatment = "X", mediator = "M")
# IPW (raw data.frame)
md_ipw <- set_md_mediation(d, Y ~ X + M + C, M ~ X + C,
  treatment = "X", mediator = "M", method = "ipw")
# A structural equation model: lavaan syntax in `model`, and the `outcome`
md_sem <- set_md_mediation(imp,
  model = "M ~ a * X + C\nY ~ b * M + cp * X + C",
  treatment = "X", mediator = "M", outcome = "Y", engine = "lavaan")
```

Index

infer, 2
infer(), *6*, *8*, *12*, *15–19*, *21*, *23*, *24*

lavaan::sem(), *9*, *23*

mbco_d4, 4
mbco_d4(), *3*, *7*, *8*, *19*, *23*
MbcoMIResult, *3*, *6*, 7
MDMediationData, 8, *10*, *13*, *17–19*, *21*, *24*
MDMediationFit, *2*, 10, *14–18*
MDMediationResult, *2*, 11, *14*, *16*, *17*
MDSensitivityResult, 12, *20*, *21*
medfit::fit_mediation(), *9*, *10*, *17*, *18*, *21*, *23*, *24*
medfit::MediationData, *3*, *10*, *11*, *15–17*
mice::mids, *14*, *22*, *23*
mice::pool(), *16*

n_imp, 14
n_imputations, 14

per_imputation_list, 15
per_imputation_list(), *3*, *11*
pool, 16
pool(), *3*, *8*, *12*, *17*, *18*, *23*, *24*

RMediation, *10*
RMediation::ci(), *3*
RMediation::ci_mediation_data(), *3*, *11*, *16*
RMediation::mbco(), *6*, *15*
RMediation::medci(), *11*, *16*
run, 17
run(), *3*, *8*, *11*, *15*, *17*, *21*, *23*, *24*

sensitivity_mnar, 18
sensitivity_mnar(), *12*, *13*, *22*, *23*
set.seed(), *19*
set_md_mediation, 21
set_md_mediation(), *2*, *5*, *9*, *10*, *12*, *17*, *18*, *20*
stats::family(), *4*
stats::gaussian(), *4*
stats::glm(), *3*, *4*, *6*

tidy(), *20*