Package 'clarify'

February 22, 2023

Type Package

Title Simulation-Based Inference for Regression Models

Version 0.1.2

Description Performs simulation-based inference as an alternative to the delta method for obtaining valid confidence intervals and p-values for regression post-estimation quantities, such as average marginal effects and predictions at representative values. This framework for simulation-based inference is especially useful when the resulting quantity is not normally distributed and the delta method approximation fails. The methodology is described in King, Tomz, and Wittenberg (2000) <doi:10.2307/2669316>. 'clarify' is meant to replace some of the functionality of the archived package 'Zelig'; see the vignette ``Translating Zelig to clarify'' for replicating this functionality.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 3.5.0)

Imports ggplot2, pbapply, chk (>= 0.8.1), rlang, insight (>= 0.19.0), marginaleffects (>= 0.9.0), mvnfast

Suggests testthat (>= 3.0.0), MatchIt (>= 4.0.0), parallel, knitr, rmarkdown, Amelia, MASS, betareg, survey, estimatr, fixest, logistf, geepack, rms, robustbase, robust, AER, ivreg, mgcv, sandwich

Config/testthat/edition 3

RoxygenNote 7.2.3

URL https://github.com/iqss/clarify, https://iqss.github.io/clarify/

BugReports https://github.com/iqss/clarify/issues

VignetteBuilder knitr

NeedsCompilation no

Author Noah Greifer [aut, cre] (<https://orcid.org/0000-0003-3067-7154>), Steven Worthington [aut] (<https://orcid.org/0000-0001-9550-5797>), Stefano Iacus [aut] (<https://orcid.org/0000-0002-4884-0047>), Gary King [aut] (<https://orcid.org/0000-0002-5327-7631>)

Maintainer Noah Greifer <ngreifer@iq.harvard.edu>

Repository CRAN

Date/Publication 2023-02-22 15:40:05 UTC

R topics documented:

misim 2
plot.clarify_adrf 4
plot.clarify_est 5
plot.clarify_setx 8
sim 9
sim_adrf 11
sim_ame 14
sim_apply 17
sim_setx 20

Index **23**

`misim` *Simulate model coefficients after multiple imputation*

Description

`misim()` simulates model parameters from multivariate normal or t distributions after multiple imputation that are then used by `sim_apply()` to calculate quantities of interest.

Usage

```
misim(fitlist, n = 1000, vcov = NULL, coefs = NULL, dist = NULL)
```

Arguments

`fitlist`	a list of model fits, one for each imputed dataset, or a `mira` object (the output of a call to `with()` applied to a `mids` object in `mice`).
`n`	the number of simulations to run for each imputed dataset; default is 1000. More is always better but resulting calculations will take longer.
`vcov`	a square covariance matrix of the coefficient covariance estimates, a function to use to extract it from `fit`, or a list thereof with an element for each imputed dataset. By default, uses `stats::vcov()` or `insight::get_varcov()` if that doesn't work.
`coefs`	a vector of coefficient estimates, a function to use to extract it from `fit`, or a list thereof with an element for each imputed dataset. By default, uses `stats::coef()` or `insight::get_parameters()` if that doesn't work.

`dist` a character vector containing the name of the multivariate distribution(s) to use to draw simulated coefficients. Should be one of `"normal"` (multivariate normal distribution) or `"t_{#}"` (multivariate t distribution), where `{#}` corresponds to the desired degrees of freedom (e.g., `"t_100"`). If `NULL`, the right distributions to use will be determined based on heuristics; see `sim()` for details.

Details

`misim()` essentially combines multiple `sim()` calls applied to a list of model fits, each fit in an imputed dataset, into a single combined pool of simulated coefficients. When simulation-based inference is to be used with multiply imputed data, many imputations are required; see Zhou and Reiter (2010).

Value

A `clarify_misim` object, which inherits from `clarify_sim` and has the following components:

`sim.coefs` a matrix containing the simulated coefficients with a column for each coefficient and a row for each simulation for each imputation

`coefs` a matrix containing the original coefficients extracted from `fitlist` or supplied to `coefs`, with a row per imputation.

`fit` the list of model fits supplied to `fitlist`

`imp` a identifier of which imputed dataset each set of simulated coefficients corresponds to.

The `"dist"` attribute contains `"normal"` if the coefficients were sampled from a multivariate normal distribution and `"t({df})"` if sampled from a multivariate t distribution. The `"clarify_hash"` attribute contains a unique hash generated by `rlang::hash()`.

References

Zhou, X., & Reiter, J. P. (2010). A Note on Bayesian Inference After Multiple Imputation. *The American Statistician*, 64(2), 159–163. doi:10.1198/tast.2010.09109

See Also

- `sim()` for simulating model coefficients for a single dataset
- `sim_apply()` for applying a function to each set of simulated coefficients
- `sim_ame()` for computing average marginal effects in each simulation draw
- `sim_setx()` for computing marginal predictions and first differences at typical values in each simulation draw

Examples

```
data("africa", package = "Amelia")

# Multiple imputation using Amelia
a.out <- Amelia::amelia(x = africa, m = 10,
```

```
                        cs = "country",
                        ts = "year", logs = "gdp_pc",
                        p2s = 0)

fits <- with(a.out, lm(gdp_pc ~ infl * trade))

# Simulate coefficients
s <- misim(fits)
s
```

`plot.clarify_adrf` *Plot marginal predictions from* `sim_adrf()`

Description

`plot.clarify_adrf()` plots the output of `sim_adrf()`. For the average dose-response function (ADRF, requested with `contrast = "adrf"` in `sim_adrf()`), this is a plot of the average marginal mean of the outcome against the requested values of the focal predictor; for the average marginal effects function (AMEF, requested with `contrast = "amef"` in `sim_adrf()`), this is a plot of the instantaneous average marginal effect of the focal predictor on the outcome against the requested values of the focal predictor.

Usage

```
## S3 method for class 'clarify_adrf'
plot(
  x,
  ci = TRUE,
  level = 0.95,
  method = "quantile",
  baseline,
  color = "black",
  ...
)
```

Arguments

`x`	a `clarify_adrf` object resulting from a call to `sim_adrf()`.
`ci`	`logical`; whether to display confidence bands for the estimates. Default is `TRUE`.
`level`	the confidence level desired. Default is .95 for 95% confidence intervals.
`method`	the method used to compute confidence bands. Can be `"wald"` to use a Normal approximation or `"quantile"` to use the simulated sampling distribution (default). See `summary.clarify_est()` for details. Abbreviations allowed.

baseline	logical; whether to include a horizontal line at y = 0 on the plot. Default is FALSE for the ADRF (since 0 might not be in the range of the outcome) and TRUE for the AMEF.
color	the color of the line and confidence band in the plot.
...	for plot(), further arguments passed to ggplot2::geom_density().

Details

These plots are produced using ggplot2::geom_line() and ggplot2::geom_ribbon(). The confidence bands should be interpreted pointwise (i.e., they do not account for simultaneous inference).

Value

A ggplot object.

See Also

summary.clarify_est() for computing p-values and confidence intervals for the estimated quantities.

Examples

```
## See help("sim_adrf") for examples
```

---

plot.clarify_est *Plotting and inference for* clarify_est *objects*

Description

summary() tabulates the estimates and confidence intervals and (optionally) p-values from a clarify_est object. confint() computes confidence intervals. plot() plots the "posterior" distribution of estimates.

Usage

```
## S3 method for class 'clarify_est'
plot(
  x,
  parm,
  ci = TRUE,
  level = 0.95,
  method = "quantile",
  reference = FALSE,
  ...
)
```

```
## S3 method for class 'clarify_est'
summary(object, parm, level = 0.95, method = "quantile", null = NA, ...)

## S3 method for class 'clarify_est'
confint(object, parm, level = 0.95, method = "quantile", ...)
```

Arguments

parm	a vector of the names or indices of the estimates to plot. If unspecified, all estimates will be displayed.
ci	logical; whether to display confidence interval limits for the estimates. Default is TRUE.
level	the confidence level desired. Default is .95 for 95% confidence intervals.
method	the method used to compute p-values and confidence intervals. Can be "wald" to use a Normal approximation or "quantile" to use the simulated sampling distribution (default). See Details. Abbreviations allowed.
reference	logical; whether to overlay a normal density reference distribution over the plots. Default is FALSE.
...	for plot(), further arguments passed to ggplot2::geom_density().
object, x	a clarify_est object; the output of a call to sim_apply() or its wrappers.
null	the values of the parameters under the null hypothesis for the p-value calculations. Should have length equal to the number of quantities estimated, or one, in which case it will be recycled, or it can be a named vector with just the names of quantities for which null values are to be set. Set values to NA to omit p-values for those quantities. When all values are NA, the default, no p-values are produced.

Details

summary() uses the estimates computed from the original model as its estimates and uses the simulated parameters for inference only.

When method = "wald", the standard deviation of the simulation estimates is used as the standard error, which is used in the z-statistics and the confidence intervals. The p-values and confidence intervals are valid only when the sampling distribution of the resulting statistic is normal (which can be assessed using plot()). When method = "quantile", the confidence interval is calculated using the quantiles of the simulation estimates corresponding to level, and the p-value is calculated as twice the proportion of simulation estimates less than or greater than null, whichever is smaller; this is equivalent to inverting the confidence interval but is only truly valid when the true sampling distribution is only a location shift from the sampling distribution under the null hypothesis and should therefore be interpreted with caution. Using "method = "quantile" (the default) is recommended because the confidence intervals will be valid even if the sampling distribution is not Normally distributed. The precision of the p-values and confidence intervals depends on the number of simulations requested (the value of n supplied to sim()).

The plots are produced using ggplot2::geom_density() and can be customized with **ggplot2** functions.

Value

For `summary()`, a `summary.clarify_est` object, which is a matrix containing the coefficient estimates, standard errors, test statistics, p-values, and confidence intervals. Not all columns will be present depending on the arguments supplied to `summary()`.

For `confint()`, a matrix containing the confidence intervals for the requested quantities.

For `plot()`, a `ggplot` object.

See Also

- `sim_apply()` for applying a function to each set of simulated coefficients

Examples

```
data("lalonde", package = "MatchIt")
fit <- glm(I(re78 > 0) ~ treat + age + race + nodegree + re74,
          data = lalonde)

s <- sim(fit, n = 100)

# Compute average marginal means for `treat`
est <- sim_ame(s, var = "treat", verbose = FALSE)
coef(est)

# Compute average marginal effects on risk difference
# (RD) and risk ratio (RR) scale
est <- transform(est,
                 RD = `E[Y(1)]` - `E[Y(0)]`,
                 RR = `E[Y(1)]` / `E[Y(0)]`)

# Compute confidence intervals and p-values,
# using given null values for computing p-values
summary(est, null = c(`RD` = 0, `RR` = 1))

# Same tests using normal approximation and alternate
# syntax for `null`
summary(est, null = c(NA, NA, 0, 1),
        normal = TRUE)

# Plot the RD and RR with a reference distribution
plot(est, parm = c("RD", "RR"), reference = TRUE,
     ci = FALSE)

# Plot the RD and RR with quantile confidence bounds
plot(est, parm = c("RD", "RR"), ci = TRUE)
```

plot.clarify_setx *Plot marginal predictions from* sim_setx()

Description

plot.clarify_sext() plots the output of sim_setx(), providing graphics similar to those of plot.clarify_est() but with features specifically for plot marginal predictions. For continues predictors, this is a plot of the marginal predictions and their confidence bands across levels of the predictor. Otherwise, this is is a plot of simulated sampling distribution of the marginal predictions.

Usage

```
## S3 method for class 'clarify_setx'
plot(
  x,
  var = NULL,
  ci = TRUE,
  level = 0.95,
  method = "quantile",
  reference = FALSE,
  ...
)
```

Arguments

x	a clarify_est object resulting from a call to sim_setx().
var	the name of the focal varying predictor, i.e., the variable to be on the x-axis of the plot. All other variables with varying set values will be used to color the resulting plot. See Details. Ignored if no predictors vary or if only one predictor varies in the reference grid or if x1 was specified in sim_setx(). If not set, will use the predictor with the greatest number of unique values specified in the reference grid.
ci	logical; whether to display confidence intervals or bands for the estimates. Default is TRUE.
level	the confidence level desired. Default is .95 for 95% confidence intervals.
method	the method used to compute confidence intervals or bands. Can be "wald" to use a Normal approximation or "quantile" to use the simulated sampling distribution (default). See summary.clarify_est() for details. Abbreviations allowed.
reference	logical; whether to overlay a normal density reference distribution over the plots. Default is FALSE. Ignored when variables other than the focal varying predictor vary.
...	for plot(), further arguments passed to ggplot2::geom_density().

Details

plot() creates one of two kinds of plots depending on how the reference grid was specified in the call to sim_setx() and what var is set to. When the focal varying predictor (i.e., the one set in var) is numeric and takes on three or more unique values in the reference grid, the produced plot is a line graph displaying the value of the marginal prediction (denoted as E[Y|X]) across values of the focal varying predictor, with confidence bands displayed when ci = TRUE. If other predictors also vary, lines for different values will be displayed in different colors. These plots are produced using ggplot2::geom_line() and ggplot2::geom_ribbon()

When the focal varying predictor is a factor or character or only takes on two or fewer values in the reference grid, the produced plot is a density plot of the simulated predictions, similar to the plot resulting from plot.clarify_est(). When other variables vary, densities for different values will be displayed in different colors. These plots are produced using ggplot2::geom_density().

Marginal predictions are identified by the corresponding levels of the predictors that vary. The user should keep track of whether the non-varying predictors are set at specified or automatically set "typical" levels.

Value

A ggplot object.

See Also

summary.clarify_est() for computing p-values and confidence intervals for the estimated quantities.

Examples

```
## See help("sim_setx") for examples
```

---

sim *Simulate model parameters*

Description

sim() simulates model parameters from a multivariate normal or t distribution that are then used by sim_apply() to calculate quantities of interest.

Usage

```
sim(fit, n = 1000, vcov = NULL, coefs = NULL, dist = NULL)
```

Arguments

fit	a model fit, such as the output of a call to lm() or glm(). Can be left unspecified if coefs and vcov are not functions.
n	the number of simulations to run; default is 1000. More is always better but resulting calculations will take longer.
vcov	either a square covariance matrix of the coefficient covariance estimates or a function to use to extract it from fit. By default, uses stats::vcov() or insight::get_varcov() if that doesn't work.
coefs	either a vector of coefficient estimates or a function to use to extract it from fit. By default, uses stats::coef() or insight::get_parameters() if that doesn't work.
dist	a string containing the name of the multivariate distribution to use to draw simulated coefficients. Should be one of "normal" (multivariate normal distribution) or "t({#})" (multivariate t distribution), where {#} corresponds to the desired degrees of freedom (e.g., "t(100)"). If NULL, the right distribution to use will be determined based on heuristics; see Details.

Details

When dist is NULL, sim() samples from a multivariate normal or t distribution depending on the degrees of freedom extracted from insight::get_df(., type = "wald"). If Inf, a normal distribution will be used; otherwise, a t-distribution with the returned degrees of freedom will be used. Models not supported by insight will use a normal distribution.

When a multivariate normal is used, it is sampled from with means equal to the estimated coefficients and the parameter covariance matrix as the covariance matrix using mvnfast::rmvn(). When a multivariate t distribution is used, it is sampled from with means equal to the estimated coefficients and scaling matrix equal to cov*(df - 2)/df, where cov is the parameter covariance matrix and df is the residual degrees of freedom for the model, using mvnfast::rmvt().

Value

A clarify_sim object, which has the following components:

sim.coefs	a matrix containing the simulated coefficients with a column for each coefficient and a row for each simulation
coefs	the original coefficients extracted from fit or supplied to coefs.
vcov	the covariance matrix of the coefficients extracted from fit or supplied to vcov
fit	the original model fit supplied to fit

The "dist" attribute contains "normal" if the coefficients were sampled from a multivariate normal distribution and "t(df)" if sampled from a multivariate t distribution. The "clarify_hash" attribute contains a unique hash generated by rlang::hash().

See Also

- misim() for simulating model coefficients after multiple imputation
- sim_apply() for applying a function to each set of simulated coefficients

- sim_ame() for computing average marginal effects in each simulation draw
- sim_setx() for computing marginal predictions and first differences at typical values in each simulation draw
- sim_adrf() for computing average dose-response functions in each simulation draw

Examples

```
data("lalonde", package = "MatchIt")
fit <- lm(re78 ~ treat * (age + race + nodegree + re74), data = lalonde)

# Simulate coefficients
s <- sim(fit)
s

## Could also use a robust covariance matrix, e.g.,
s <- sim(fit, vcov = "HC3")

# Simulated coefficients assuming a normal distribution
# for coefficients; default for `lm` objects is a t-
# distribution
s <- sim(fit, dist = "normal")
s
```

---

sim_adrf *Compute an average dose-response function*

Description

sim_adrf() is a wrapper for sim_apply() that computes average dose-response functions (ADRFs) and average marginal effect functions (AMEFs). An ADRF describes the relationship between values a focal variable can take and the expected value of the outcome were all units to be given each value of the variable. An AMEF describes the relationship between values a focal variable can take and the derivative of ADRF at each value.

Usage

```
sim_adrf(
  sim,
  var,
  subset = NULL,
  contrast = "adrf",
  at = NULL,
  n = 21,
  outcome = NULL,
  type = NULL,
  eps = 1e-05,
```

```
  verbose = TRUE,
  cl = NULL
)

## S3 method for class 'clarify_adrf'
print(x, digits = NULL, max.ests = 6, ...)
```

Arguments

sim	a clarify_sim object; the output of a call to sim() or misim().
var	the name of a variable for which the ADRF or AMEF is to be computed. This variable must be present in the model supplied to sim() and must be a numeric variable taking on more than two unique values.
subset	optional; a vector used to subset the data used to compute the ADRF or AMEF. This will be evaluated within the original dataset used to fit the model using subset(), so nonstandard evaluation is allowed.
contrast	a string naming the type of quantity to be produced: "adrf" for the ADRF (the default) or "amef" for the AMEF.
at	the levels of the variable named in var at which to evaluate the ADRF or AMEF. Should be a vector of numeric values corresponding to possible levels of var. If NULL, will be set to a range from slightly below the lowest observed value of var to slightly above the largest value.
n	when at = NULL, the number of points to evaluate the ADRF or AMEF. Default is 21. Ignored when at is not NULL.
outcome	a string containing the name of the outcome or outcome level for multivariate (multiple outcomes) or multi-category outcomes. Ignored for univariate (single outcome) and binary outcomes.
type	a string containing the type of predicted values (e.g., the link or the response). Passed to marginaleffects::get_predict() and eventually to predict() in most cases. The default and allowable option depend on the type of model supplied, but almost always corresponds to the response scale (e.g., predicted probabilities for binomial models).
eps	when contrast = "amef", the value by which to shift the value of var to approximate the derivative. See Details.
verbose	logical; whether to display a text progress bar indicating progress and estimated time remaining for the procedure. Default is TRUE.
cl	a cluster object created by parallel::makeCluster(), or an integer to indicate the number of child-processes (integer values are ignored on Windows) for parallel evaluations. See pbapply::pblapply() for details. If NULL, no parallelization will take place.
x	a clarify_adrf object.
digits	the minimum number of significant digits to be used; passed to print.data.frame().
max.ests	the maximum number of estimates to display.
...	optional arguments passed to FUN.

Details

The ADRF is composed of average marginal means across levels of the focal predictor. For each level of the focal predictor, predicted values of the outcome are computed after setting the value of the predictor to that level, and those values of the outcome are averaged across all units in the sample to arrive at an average marginal mean. Thus, the ADRF represent the relationship between the "dose" (i.e., the level of the focal predictor) and the average "response" (i.e., the outcome variable). It is the continuous analog to the average marginal effect computed for a binary predictor, e.g., using `sim_ame()`. Although inference can be at each level of the predictor or between two levels of the predictor, typically a plot of the ADRF is the most useful relevant quantity. These can be requested using `plot.clarify_adrf()`.

The AMEF is the derivative of the ADRF; if we call the derivative of the ADRF at each point a "treatment effect" (i.e., the rate at which the outcome changes corresponding to a small change in the predictor, or "treatment"), the AMEF is a function that relates the size of the treatment effect to the level of the treatment. The shape of the AMEF is usually of less importance than the value of the AMEF at each level of the predictor, which corresponds to the size of the treatment effect at the corresponding level. The AMEF is computed by computing the ADRF at each level of the focal predictor specified in `at`, shifting the predictor value by a tiny amount (control by `eps`), and computing the ratio of the change in the outcome to the shift, then averaging this value across all units. This quantity is related the the average marginal effect of a continuous predictor as computed by `sim_ame()`, but rather than average these treatment effects across all observed levels of the treatment, the AMEF is a function evaluated at each possible level of the treatment. The "tiny amount" used is `eps` times the standard deviation of `var`.

If unit-level weights are included in the model fit (and discoverable using `insight::get_weights()`), all means will be computed as weighted means.

Value

A `clarify_adrf` object, which inherits from `clarify_est` and is similar to the output of `sim_apply()`, with the additional attributes `"var"` containing the variable named in `var`, `"at"` containing values at which the ADRF or AMEF is evaluated, and `"contrast"` containing the argument supplied to `contrast`. For an ADRF, the average marginal means will be named `E[Y({v})]`, where `{v}` is replaced with the values in `at`. For an AMEF, the average marginal effects will be named `dY/d({x})|{a}` where `{x}` is replaced with `var` and `{a}` is replaced by the values in `at`.

See Also

`plot.clarify_adrf()` for plotting the ADRF or AMEF; `sim_ame()` for computing average marginal effects; `sim_apply()`, which provides a general interface to computing any quantities for simulation-based inference; `summary.clarify_est()` for computing p-values and confidence intervals for the estimated quantities.

`marginaleffects::marginaleffects()` and `marginaleffects::predictions()` for delta method-based implementations of computing average marginal effects and average marginal means.

Examples

```
data("lalonde", package = "MatchIt")

# Fit the model
```

```
fit <- glm(I(re78 > 0) ~ treat + age + race + re74,
           data = lalonde, family = binomial)

# Simulate coefficients
set.seed(123)
s <- sim(fit, n = 100)

# ADRF for `age`
est <- sim_adrf(s, var = "age",
                at = seq(15, 55, length.out = 6),
                verbose = FALSE)
est
plot(est)

# AMEF for `age`
est <- sim_adrf(s, var = "age", contrast = "amef",
               at = seq(15, 55, length.out = 6),
               verbose = FALSE)
est
summary(est)
plot(est)
```

`sim_ame` *Compute average marginal effects*

Description

`sim_ame()` is a wrapper for `sim_apply()` that computes average marginal effects, the average effect of changing a single variable from one value to another (i.e., from one category to another for categorical variables or a tiny change for continuous variables).

Usage

```
sim_ame(
  sim,
  var,
  subset = NULL,
  contrast = NULL,
  outcome = NULL,
  type = NULL,
  eps = 1e-05,
  verbose = TRUE,
  cl = NULL
)

## S3 method for class 'clarify_ame'
print(x, digits = NULL, max.ests = 6, ...)
```

Arguments

sim	a clarify_sim object; the output of a call to sim() or misim().
var	either the name of a variable for which marginal effects are to be computed or a named list of length one containing the values the variable should take. If a list is supplied or the named variables is categorical (factor, character, or having two values), categorical calculations will be triggered. Otherwise, continuous calculations will be triggered. See Details.
subset	optional; a vector used to subset the data used to compute the marginal effects. This will be evaluated within the original dataset used to fit the model using subset(), so nonstandard evaluation is allowed.
contrast	a string containing the name of a contrast between the average marginal means when the variable named in var is categorical and takes on two values. Allowed options include "diff" for the difference in means (also "rd"), "rr" for the risk ratio (also "irr"), "log(rr)" for the log risk ratio (also "log(irr)"), "or" for the odds ratio, "log(or)" for the log odds ratio, and "nnt" for the number needed to treat. These options are not case sensitive, but the parentheses must be included if present.
outcome	a string containing the name of the outcome or outcome level for multivariate (multiple outcomes) or multi-category outcomes. Ignored for univariate (single outcome) and binary outcomes.
type	a string containing the type of predicted values (e.g., the link or the response). Passed to marginaleffects::get_predict() and eventually to predict() in most cases. The default and allowable option depend on the type of model supplied, but almost always corresponds to the response scale (e.g., predicted probabilities for binomial models).
eps	when the variable named in var is continuous, the value by which to change the variable values to approximate the derivative. See Details.
verbose	logical; whether to display a text progress bar indicating progress and estimated time remaining for the procedure. Default is TRUE.
cl	a cluster object created by parallel::makeCluster(), or an integer to indicate the number of child-processes (integer values are ignored on Windows) for parallel evaluations. See pbapply::pblapply() for details. If NULL, no parallelization will take place.
x	a clarify_ame object.
digits	the minimum number of significant digits to be used; passed to print.data.frame().
max.ests	the maximum number of estimates to display.
...	optional arguments passed to FUN.

Details

sim_ame() operates differently depending on whether continuous or categorical calculations are triggered. To trigger categorical calculations, var should be a string naming a factor, character, or binary variable or a named list with specific values given (e.g., var = list(x1 = c(1, 2 ,3))). Otherwise, continuous calculations are triggered.

Categorical calculations involve computing average marginal means at each level of `var`. The average marginal mean is the average predicted outcome value after setting all units' value of `var` to one level. (This quantity has several names, including the average potential outcome, average adjusted prediction, and standardized mean). When `var` only takes on two levels (or it is supplied as a list and only two values are specified), a contrast between the average marginal means can be computed by supplying an argument to `contrast`. Contrasts can be manually computed using `transform()` afterward as well.

Continuous calculations involve computing the average of marginal effects of `var` across units. A marginal effect is the instantaneous rate of change corresponding to changing a unit's observed value of `var` by a tiny amount and considering to what degree the predicted outcome changes. The ratio of the change in the predicted outcome to the change in the value of `var` is the marginal effect; these are averaged across the sample to arrive at an average marginal effect. The "tiny amount" used is `eps` times the standard deviation of the focal variable.

If unit-level weights are included in the model fit (and discoverable using `insight::get_weights()`), all means will be computed as weighted means.

Effect measures:

The effect measures specified in `contrast` are defined below. Typically only `"diff"` is appropriate for continuous outcomes and `"diff"` or `"irr"` are appropriate for count outcomes; the rest are appropriate for binary outcomes. For a focal variable with two levels, `0` and `1`, and an outcome `Y`, the average marginal means will be denoted in the below formulas as `E[Y(0)]` and `E[Y(1)]`, respectively.

contrast	Formula
`"diff"`	`E[Y(1)] - E[Y(0)]`
`"rr"`	`E[Y(1)] / E[Y(0)]`
`"or"`	`O[Y(1)] / O[Y(0)]`, where `O[Y(.)] = E[Y(.)] / (1 - E[Y(.)])`
`"nnt"`	`1 / (E[Y(1)] - E[Y(0)])`

The `log(.)` versions are defined by taking the `log()` (natural log) of the corresponding effect measure.

Value

A `clarify_ame` object, which inherits from `clarify_est` and is similar to the output of `sim_apply()`, with the additional attribute `"var"` containing the variable named in `var`. The average marginal means will be named `E[Y({v})]`, where `{v}` is replaced with the values the focal variable (`var`) takes on. The average marginal effect for a continuous `var` will be named `dY/d({x})` where `{x}` is replaced with `var`.

See Also

`sim_apply()`, which provides a general interface to computing any quantities for simulation-based inference; `plot.clarify_est()` for plotting the output of a call to `sim_ame()`; `summary.clarify_est()` for computing p-values and confidence intervals for the estimated quantities.

`marginaleffects::marginaleffects()`, `marginaleffects::comparisons()`, and `margins::margins()` for delta method-based implementations of computing average marginal effects.

Examples

```
data("lalonde", package = "MatchIt")

# Fit the model
fit <- glm(I(re78 > 0) ~ treat + age + race + re74,
           data = lalonde, family = binomial)

# Simulate coefficients
set.seed(123)
s <- sim(fit, n = 100)

# Average marginal effect of `age`
est <- sim_ame(s, var = "age", verbose = FALSE)
summary(est)

# Contrast between average marginal means for `treat`
est <- sim_ame(s, var = "treat", contrast = "rr",
               verbose = FALSE)
summary(est)

# Average marginal means for `race`; need to follow up
# with contrasts for specific levels
est <- sim_ame(s, var = "race", verbose = FALSE)

est <- transform(est,
                 `RR(h,b)` = `E[Y(hispan)]` / `E[Y(black)]`)

summary(est)
```

`sim_apply` *Apply a function to simulated parameter values*

Description

`sim_apply()` applies a function that produces quantities of interest to each set of simulated coefficients produced by `sim()`; these calculated quantities form the posterior sampling distribution for the quantities of interest. Capabilities are available for parallelization.

Usage

```
sim_apply(sim, FUN, verbose = TRUE, cl = NULL, ...)
```

Arguments

`sim`	a `clarify_sim` object; the output of a call to `sim()` or `misim()`.
`FUN`	a function to be applied to each set of simulated coefficients. See Details.
`verbose`	`logical`; whether to display a text progress bar indicating progress and estimated time remaining for the procedure. Default is `TRUE`.

cl	a cluster object created by parallel::makeCluster(), or an integer to indicate the number of child-processes (integer values are ignored on Windows) for parallel evaluations. See pbapply::pblapply() for details. If NULL, no parallelization will take place.
...	optional arguments passed to FUN.

Details

sim_apply() applies a function, FUN, to each set of simulated coefficients, similar to apply(). This function should return a numeric vector containing one or more estimated quantities. This should be a named vector to more easily keep track of the meaning of each estimated quantity. Care should be taken to ensure that the returned vector is the same length each time FUN is called. NAs are allowed in the output but should be avoided if possible.

The arguments to FUN can be specified in a few ways. If FUN has an argument called coefs, a simulated set of coefficients will be passed to this argument, and FUN should compute and return a quantity based on the coefficients (e.g., the difference between two coefficients if one wants to test whether two coefficients are equal). If FUN has an argument called fit, a model fit object of the same type as the one originally supplied to sim() (e.g., an lm or glm object) will be passed to this argument, where the coefficients of the fit object have been replaced by the simulated coefficients generated by sim(), and FUN should compute and return a quantity based on the model fit (e.g., a computation based on the output of predict()). If neither coefs nor fit are the names of arguments to FUN, the model fit object with replaced coefficients will be supplied to the first argument of FUN.

When custom coefficients are supplied to sim(), i.e., when the coefs argument to sim() is not left at its default value, FUN must accept a coefs argument and a warning will be thrown if it accepts a fit argument. This is because sim_apply() does not know how to reconstruct the original fit object with the new coefficients inserted. The quantities computed by sim_apply() must therefore be computed directly from the coefficients.

sim_apply() with multiply imputed data:

When using misim() and sim_apply() with multiply imputed data, the coefficients are supplied to the model fit corresponding to the imputation identifier associated with each set of coefficients, which means if FUN uses a dataset extracted from a model, it will do so from the model fit in the corresponding imputation.

The original estimates (see Value below) are computed as the mean of the estimates across the imputations using the original coefficients averaged across imputations. That is, first, the coefficients estimated in the models in the imputed datasets are combined to form a single set of pooled coefficients; then, for each imputation, the quantities of interest are computed using the pooled coefficients; finally, the mean of the resulting estimates across the imputations are taken as the "original" estimates. Note this procedure is only valid for quantities with symmetric sampling distributions, which excludes quantities like risk ratios and odds ratios, but includes log risk ratios and log odds ratios. The desired quantities can be transformed from their log versions using transform().

Value

A clarify_est object, which is a matrix with a column for each estimated quantity and a row for each simulation. The original estimates (FUN applied to the original coefficients or model fit object)

are stored in the attribute "original". The "sim_hash" attributes contained the simulation hash produced by sim().

See Also

- sim() for generating the simulated coefficients
- summary.clarify_est() for computing p-values and confidence intervals for the estimated quantities
- plot.clarify_est() for plotting estimated quantities and their simulated posterior sampling distribution.

Examples

```
data("lalonde", package = "MatchIt")
fit <- lm(re78 ~ treat + age + race + nodegree + re74,
          data = lalonde)
coef(fit)

set.seed(123)
s <- sim(fit, n = 100)

# Function to compare predicted values for two units
# using `fit` argument
sim_fun <- function(fit) {
  pred1 <- unname(predict(fit, newdata = lalonde[1,]))
  pred2 <- unname(predict(fit, newdata = lalonde[2,]))
  c(pred1 = pred1, pred2 = pred2)
}

est <- sim_apply(s, sim_fun, verbose = FALSE)

# Add difference between predicted values as
# additional quantity
est <- transform(est, `diff 1-2` = pred1 - pred2)

# Examine estimates and confidence intervals
summary(est)

# Function to compare coefficients using `coefs`
# argument
sim_fun <- function(coefs) {
  c(`wh - his` = coefs["racewhite"] - coefs["racehispan"])
}

est <- sim_apply(s, sim_fun, verbose = FALSE)

# Examine estimates and confidence intervals
summary(est)
```

sim_setx *Compute predictions and first differences at set values*

Description

`sim_setx()` is a wrapper for `sim_apply()` that computes predicted values of the outcome at specified values of the predictors, sometimes called marginal predictions. One can also compute the difference between two marginal predictions (the "first difference"). Although any function that accepted `clarify_est` objects can be used with `sim_setx()` output objects, a special plotting function, `plot.clarify_setx()`, can be used to plot marginal predictions.

Usage

```
sim_setx(
  sim,
  x = list(),
  x1 = list(),
  outcome = NULL,
  type = NULL,
  verbose = TRUE,
  cl = NULL
)

## S3 method for class 'clarify_setx'
print(x, digits = NULL, max.ests = 6, ...)
```

Arguments

`sim`	a `clarify_sim` object; the output of a call to `sim()` or `misim()`.
`x`	a named list of values each predictor should take defining a reference grid of predictor values, e.g., `list(v1 = 1:4, v2 = c("A", "B"))`. Any omitted predictors are fixed at a "typical" value. See Details. When `x1` is specified, `x` should identify a single reference unit. For `print()`, a `clarify_setx` object.
`x1`	a named list of the value each predictor should take to compute the first difference from the predictor combination specified in `x`. `x1` can only identify a single unit. See Details.
`outcome`	a string containing the name of the outcome or outcome level for multivariate (multiple outcomes) or multi-category outcomes. Ignored for univariate (single outcome) and binary outcomes.
`type`	a string containing the type of predicted values (e.g., the link or the response). Passed to `marginaleffects::get_predict()` and eventually to `predict()` in most cases. The default and allowable option depend on the type of model supplied, but almost always corresponds to the response scale (e.g., predicted probabilities for binomial models).

verbose	logical; whether to display a text progress bar indicating progress and estimated time remaining for the procedure. Default is TRUE.
cl	a cluster object created by parallel::makeCluster(), or an integer to indicate the number of child-processes (integer values are ignored on Windows) for parallel evaluations. See pbapply::pblapply() for details. If NULL, no parallelization will take place.
digits	the minimum number of significant digits to be used; passed to print.data.frame().
max.ests	the maximum number of estimates to display.
...	optional arguments passed to FUN.

Details

x should be a named list of predictor values that will be crossed to form a reference grid for the marginal predictions. Any predictors not set in x are assigned their "typical" value, which, for factor, character, logical, and binary variables is the mode, for numeric variables is the mean, and for ordered variables is the median. These values can be seen in the "setx" attribute of the output object. If x is empty, a prediction will be made at a point corresponding to the typical value of every predictor. Estimates are identified (in summary(), etc.) only by the variables that differ across predictions.

When x1 is supplied, the first difference is computed, which here is considered as the difference between two marginal predictions. One marginal prediction must be specified in x and another, ideally with a single predictor changed, specified in x1.

Value

a clarify_setx object, which inherits from clarify_est and is similar to the output of sim_apply(), with the following additional attributes:

- "setx" - a data frame containing the values at which predictions are to be made
- "fd" - whether or not the first difference is to be computed; set to TRUE if x1 is specified and FALSE otherwise

See Also

sim_apply(), which provides a general interface to computing any quantities for simulation-based inference; plot.clarify_setx() for plotting the output of a call to sim_setx(); summary.clarify_est() for computing p-values and confidence intervals for the estimated quantities.

Examples

```
data("lalonde", package = "MatchIt")

fit <- lm(re78 ~ treat + age + educ + married + race + re74,
          data = lalonde)

# Simulate coefficients
set.seed(123)
s <- sim(fit, n = 100)
```

```
# Predicted values at specified values of treat, typical
# values for other predictors
est <- sim_setx(s, x = list(treat = 0:1,
                            re74 = c(0, 10000)),
                verbose = FALSE)
summary(est)
plot(est)

# Predicted values at specified grid of values, typical
# values for other predictors
est <- sim_setx(s, x = list(age = c(20, 25, 30, 35),
                            married = 0:1),
                verbose = FALSE)
summary(est)
plot(est)

# First differences of treat at specified value of
# race, typical values for other predictors
est <- sim_setx(s, x = list(treat = 0, race = "hispan"),
                x1 = list(treat = 1, race = "hispan"),
                verbose = FALSE)
summary(est)
plot(est)
```

Index

apply(), *18*

confint.clarify_est *(*plot.clarify_est*)*, *5*

ggplot2::geom_density(), *5, 6*, *8, 9*
ggplot2::geom_line(), *5*, *9*
ggplot2::geom_ribbon(), *5*, *9*
glm(), *10*

insight::get_parameters(), *2*, *10*
insight::get_varcov(), *2*, *10*
insight::get_weights(), *13*, *16*

lm(), *10*
log(), *16*

marginaleffects::get_predict(), *12*, *15*, *20*
misim, 2
misim(), *10*, *12*, *15*, *17, 18*, *20*
mvnfast::rmvn(), *10*
mvnfast::rmvt(), *10*

parallel::makeCluster(), *12*, *15*, *18*, *21*
pbapply::pblapply(), *12*, *15*, *18*, *21*
plot.clarify_adrf, 4
plot.clarify_adrf(), *13*
plot.clarify_est, 5
plot.clarify_est(), *8, 9*, *16*, *19*
plot.clarify_setx, 8
plot.clarify_setx(), *20, 21*
print.clarify_adrf *(*sim_adrf*)*, 11
print.clarify_ame *(*sim_ame*)*, 14
print.clarify_setx *(*sim_setx*)*, 20
print.data.frame(), *12*, *15*, *21*

rlang::hash(), *3*, *10*

sim, 9
sim(), *3*, *6*, *12*, *15*, *17*, *19, 20*
sim_adrf, 11
sim_adrf(), *4*, *11*
sim_ame, 14
sim_ame(), *3*, *11*, *13*
sim_apply, 17
sim_apply(), *2, 3*, *6, 7*, *9–11*, *13, 14*, *16*, *20, 21*
sim_setx, 20
sim_setx(), *3*, *8*, *11*
stats::coef(), *2*, *10*
stats::vcov(), *2*, *10*
subset(), *12*, *15*
summary.clarify_est *(*plot.clarify_est*)*, *5*
summary.clarify_est(), *4, 5*, *8, 9*, *13*, *16*, *19*, *21*

transform(), *16*, *18*