Package: switchSelection (via r-universe)

October 18, 2026

Type Package

Title Endogenous Switching and Sample Selection Regression Models

Version 2.1.0

Date 2026-04-15

Description Estimate the parameters of multivariate endogenous switching and sample selection models using methods described in Newey (2009) <doi:10.1111/j.1368-423X.2008.00263.x>, E. Kossova, B. Potanin (2018) <https://ideas.repec.org/a/ris/apltrx/0346.html>, E. Kossova, L. Kupriianova, B. Potanin (2020) <https://ideas.repec.org/a/ris/apltrx/0391.html> and E. Kossova, B. Potanin (2022) <https://ideas.repec.org/a/ris/apltrx/0455.html>.

License GPL (>= 2)

Imports Rcpp (>= 1.1.1), hpa (>= 1.3.4), mnorm (>= 1.2.3), gena (>= 1.0.1), dplyr (>= 1.1.4), methods

LinkingTo Rcpp, RcppArmadillo, hpa, mnorm

Depends R (>= 3.5.0)

RoxygenNote 7.3.2

NeedsCompilation yes

Author Bogdan Potanin [aut, cre, ctb], Sofiia Dolgikh [ctb]

Maintainer Bogdan Potanin <bogdanpotanin@gmail.com>

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

Date/Publication 2026-04-15 17:36:12 UTC

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

RemoteRef HEAD

RemoteSha 7f9b6ee23a465388459e4398073dddc4342b8b6b

Contents

boot . . . 2
bootstrap . . . 3
coef.msel . . . 7
cps . . . 9
exogenous_fn . . . 11
fitted.msel . . . 11
formula.msel . . . 12
formula_merge . . . 13
formula_split . . . 14
grad_msel . . . 14
lnL_msel . . . 15
logLik.msel . . . 16
loocv . . . 16
lrtest_msel . . . 17
msel . . . 18
nobs.msel . . . 62
predict.msel . . . 63
print.lrtest_msel . . . 67
print.msel . . . 67
print.struct_msel . . . 68
print.summary.lrtest_msel . . . 68
print.summary.msel . . . 69
print.summary.test_msel . . . 69
sigma.msel . . . 70
starsVector . . . 70
struct_msel . . . 71
summary.lrtest_msel . . . 72
summary.msel . . . 72
summary.test_msel . . . 73
test_msel . . . 74
update_msel . . . 85
vcov.msel . . . 86
vcov_combine . . . 87

Index **88**

`boot` *Bootstrap covariance matrix for least squares estimates of linear regression*

Description

This function calculates bootstrapped covariance matrix for least squares estimates of linear regression. The estimates should be obtained via `lm` function.

Usage

```
boot(model, iter = 100)
```

Arguments

model	object of class lm.
iter	positive integer representing the number of bootstrap iterations.

Details

Calculations may take long time for high iter value.

Value

This function returns a bootstrapped covariance matrix of the least squares estimator.

Examples

```
set.seed(123)
# Generate data according to linear regression
n <- 20
eps <- rnorm(n)
x <- runif(n)
y <- x + eps
# Estimate the model
model <- lm(y ~ x)
# Calculate bootstrap covariance matrix
boot(model, iter = 50)
```

---

bootstrap *Bootstrap for msel function*

Description

Function bootstrap_msel provides bootstrap estimates of the parameters of the model estimated via the msel function. Function bootstrap_combine_msel allows to combine several objects of class 'bootstrap_msel'.

Usage

```
bootstrap_msel(
  object,
  iter = 100,
  opt_type = "optim",
  opt_args = NULL,
  is_ind = FALSE,
  n_sim = 1000,
  n_cores = 1
```

```
)

bootstrap_combine_msel(...)
```

Arguments

`object`	an object of class 'msel' or a list of such objects.
`iter`	the number of bootstrap iterations.
`opt_type`	the same as the `opt_type` argument of the `msel` function.
`opt_args`	the same as the `opt_args` argument of the `msel` function.
`is_ind`	logical; if `TRUE` then the function also returns a numeric matrix of indexes of observations used in the bootstrap samples.
`n_sim`	the same as the `n_sim` argument of the `msel` function.
`n_cores`	the same as the `n_cores` argument of the `msel` function.
`...`	objects returned by the function `bootstrap_msel` to be combined into a single object.

Details

The function generates `iter` bootstrap samples and estimates the parameters θ of the model by using each of these samples. The estimate $\hat{\theta}^{(b)}$ from the b-th of these samples is stored as the `b`-th row of the numeric matrix `par` which is an element of the returned object.

Use the `update_msel` function to transfer the bootstrap estimates to the object of class `'msel'`.

Argument `object` may be a list of objects of class `'msel'`. Then the function returns an object of class `'crossmodel_bootstrap_msel'` which is a list `out` such that `out[[j]]` is an object of class `'bootstrap_msel'`. Notice that all the models will be estimated on the same bootstrap samples. That is, `out[[j]]$ind` is the same for each model `j`. Specifically, it may be useful for cross-model hypothesis testing.

See `test_msel` for hypothesis testing.

Value

Function `bootstrap_msel` returns an object of class `"bootstrap_msel"`. This object is a list which may contain the following elements:

- `par` - a numeric matrix such that `par[b, ]` is a vector of the estimates of the parameters of the model estimated via the `msel` function on the `b`-th bootstrap sample.
- `iter` - the number of the bootstrap iterations.
- `cov` - bootstrap estimate of the covariance matrix which equals `cov(par)`
- `ind` - a numeric matrix such that `ind[, b]` stores the indexes of the observations from `object$data` included in the `b`-th bootstrap sample.

Function `bootstrap_combine_msel` returns the object which combines several objects returned by the `bootstrap_msel` function into a single object.

Examples

```
# ------------------------------
# Bootstrap for the probit model
# ------------------------------

# ---
# Step 1
# Simulation of data
# ---

# Load required package
library("mnorm")

# Set seed for reproducibility
set.seed(123)

# The number of observations
n <- 100

# Regressors (covariates)
w1 <- runif(n = n, min = -1, max = 1)
w2 <- runif(n = n, min = -1, max = 1)

# Random errors
u <- rnorm(n = n, mean = 0, sd = 1)

# Coefficients
gamma <- c(-1, 2)

# Linear index
li <- gamma[1] * w1 + gamma[2] * w2

# Latent variable
z_star <- li + u

# Cuts
cuts <- c(-1, 0.5, 2)

# Observable ordered outcome
z                                          <- rep(0, n)
z[(z_star > cuts[1]) & (z_star <= cuts[2])] <- 1
z[(z_star > cuts[2]) & (z_star <= cuts[3])] <- 2
z[z_star > cuts[3]]                        <- 3
table(z)

# Data
data <- data.frame(w1 = w1, w2 = w2, z = z, crossind = 1:n)

# ---
# Step 2
# Estimation of the parameters
# ---
```

```
# Estimation
model <- msel(formula = z ~ w1 + w2, data = data)
summary(model)

# ---
# Step 3
# Bootstrap
# ---

# Perform the bootstrap
bootstrap <- bootstrap_msel(model)

# Test the hypothesis that H0: gamma[2] = -2gamma[1]
# by using the t-test and with bootstrap p-values
fn_test <- function(object)
{
  gamma <- coef(object, eq = 1)
  return(gamma[2] + 2 * gamma[1])
}
b <- test_msel(object    = model,
               fn        = fn_test,
               test      = "t",
               method    = "bootstrap",
               ci        = "percentile",
               se_type   = "bootstrap",
               bootstrap = bootstrap)
summary(b)

# Replicate the analysis with an additional 20 bootstrap iterations
bootstrap2    <- bootstrap_msel(model, iter = 20)
bootstrap_new <- bootstrap_combine_msel(bootstrap, bootstrap2)
b2 <- test_msel(object    = model,
                fn        = fn_test,
                test      = "t",
                method    = "bootstrap",
                ci        = "percentile",
                se_type   = "bootstrap",
                bootstrap = bootstrap_new)
summary(b2)

# ---
# Step 4
# Cross-model test
# ---

# Estimate restricted model
model2 <- msel(formula = z ~ w1, data = data)

# Perform a bootstrap for two models
bootstrap2 <- bootstrap_msel(list(model, model2), iter = 10)

# Test the hypothesis that the coefficients in both models are the same
```

```
fn_test2 <- function(object)
{
  coef1 <- coef(object[[1]], eq = 1)[1]
  coef2 <- coef(object[[2]], eq = 1)[1]
  return(coef1 - coef2)
}
b <- test_msel(object    = list(model, model2),
               fn        = fn_test2,
               test      = "t",
               method    = "bootstrap",
               ci        = "percentile",
               se_type   = "bootstrap",
               bootstrap = bootstrap2)
summary(b)
```

`coef.msel` *Coefficients extraction method for msel.*

Description

Extract coefficients and other estimates from msel object.

Usage

```
## S3 method for class 'msel'
coef(
  object,
  ...,
  eq = NULL,
  eq2 = NULL,
  eq3 = NULL,
  regime = NULL,
  type = "coef"
)
```

Arguments

`object`	an object of class "msel".
`...`	further arguments (currently ignored).
`eq`	an integer representing the index of the ordered equation.
`eq2`	an integer representing the index of the continuous equation.
`eq3`	an integer representing the index of the alternative of the multinomial equation.
`regime`	an integer representing a regime of the continuous equation.

type — a character representing a type of the output. Possible options are "coef", "coef2", "coef_lambda", "coef_var", "coef3", "cuts", "cov", "cov1", "cov12", "var", "cov2", "cov3", and "marginal". See 'Details' for additional information.

Details

Consider the notations from the 'Details' section of msel.

Mean coefficients of the ordinal equations

Suppose that type = "coef". Then estimates of the γ_j coefficients are returned for each $j \in \{1, ..., J\}$. If eq = j then only estimates of the γ_j coefficients are returned.

Variance coefficients of the ordinal equations

Suppose that type = "coef_var". Then estimates of the γ_j^* coefficients are returned for each $j \in \{1, ..., J\}$. If eq = j then only estimates of γ_j^* coefficients are returned.

Coefficients of the continuous equations

Suppose that type = "coef2". Then estimates of the β_r coefficients are returned for each $r \in \{0, ..., R-1\}$. If eq2 = k then only estimates for the k-th continuous equation are returned. If regime = r then estimates of the β_r coefficients are returned for the eq2-th continuous equation. Herewith if regime is not NULL and eq2 is NULL it is assumed that eq2 = 1.

Selectivity terms

Suppose that type = "coef_lambda". Then estimates of the coefficients associated with the selectivity terms are returned for each $r \in \{0, ..., R-1\}$. If eq2 = k then only estimates for the k-th continuous equation are returned. If regime = r then estimates of the coefficients of the selectivity terms are returned for the eq2-th continuous equation.

Thresholds of the ordinal equations

Suppose that type = "cuts" or type = "thresholds". Then estimates of the c_j cuts (thresholds) are returned for each $j \in \{1, ..., J\}$. If eq = j then only estimates of the c_j cuts are returned.

Covariances between the random errors of the ordinal equations

Suppose that type = "cov1". Then the estimate of the covariance matrix of u_i is returned. If eq = c(a, b) then the function returns (a, b)-th element of this matrix i.e. an element from the a-th row and the b-th column which represents an estimate of $Cov(u_{ai}, u_{bi})$.

Covariances between the random errors of the ordinal and continuous equations

Suppose that type = "cov12". Then estimates of the covariances between random errors of the ordinal u_i and continuous ε_i equations are returned. If eq2 = k then covariances with random errors of the k-th continuous equation are returned. If in addition eq = j and regime = r then the function returns an estimate of $Cov(u_{ji}, \varepsilon_{ri})$ for the k-th continuous equation. If eq2 = NULL it is assumed that eq2 = 1.

Variances of the random errors of the continuous equations

Suppose that type = "var". Then estimates of the variances of ε_i are returned. If eq2 = k then estimates only for the k-th continuous equation are returned. If in addition regime = r then the estimate of the $Var(\varepsilon_{ri})$ is returned. Herewith if regime is not NULL and eq2 is NULL it is assumed that eq2 = 1.

Covariances between the random errors of the continuous equations

Suppose that `type = "cov2"`. Then estimates of the covariances between random errors of different continuous equations in different regimes are returned. If `eq2 = c(a, b)` and `regime = c(c, d)` then the function returns an estimate of the covariance of random errors of the `a`-th and `b`-th continuous equations in the regimes `c` and `d` correspondingly. If this covariance is not identifiable then an `NA` value is returned.

Coefficients of the multinomial equation

Suppose that `type = "coef3"`. Then estimates of the $\tilde{\gamma}_j$ coefficients are returned for each $j \in \{0, ..., \tilde{J} - 1\}$. If `eq3 = j` then only estimates of the $\tilde{\gamma}_j$ coefficients are returned.

Covariances between the random errors of the multinomial equations

Suppose that `type = "cov3"`. Then the estimate of the covariance matrix of $\tilde{u}_i$ is returned. If `eq3 = c(a, b)` then the function returns (a, b)-th element of this matrix i.e. an element from the `a`-th row and the `b`-th column which represents an estimate of $Cov(\tilde{u}_{(a+1)i}, \tilde{u}_{(b+1)i})$.

Parameters of the marginal distributions

Suppose that `type = "marginal"`. Then a list is returned whose `j`-th element is a numeric vector of estimates of the parameters of the marginal distribution of u_{ji}.

Asymptotic covariance matrix

Suppose that `type = "cov"`. Then the estimate of the asymptotic covariance matrix of the estimator is returned. Note that this estimate depends on the `cov_type` argument of the `msel` function.

Value

See 'Details' section.

---

cps *A subset of data from Current Population Survey (CPS).*

Description

Labor market data on 18,253 middle age (25-54 years) married women in the year 2022.

Usage

```
data(cps)
```

Format

A data frame with 18,253 rows and 25 columns. It contains information on wages and some socio-demographic characteristics of the middle age (25-54 years) married women:

age the age measured in years.

sage the same as age but for a spouse.

work a binary variable for the employment status (0 - unemployed, 1 - employed).

swork the same as work but for a spouse.

nchild the number of children under age 5.

snchild the same as nchild but for a spouse.

health subjective health status (1 - poor, 2 - fair, 3 - good, 4 - very good, 5 - excellent).

shealth the same as health but for a spouse.

basic a binary variable which equals 1 for those who have graduated from high school or has at least some college or has an associated degree and does not have any higher level of education, 0 - otherwise.

bachelor a binary variable which equals 1 for those whose highest education level is a bachelor degree.

master a binary variable which equals 1 for those whose highest education level is a master degree.

sbasic the same as basic but for a spouse.

sbachelor the same as bachelor but for a spouse.

smaster the same as master but for a spouse.

educ a categorical variable for the level of education such that educ = 0 if basic = 1, educ = 1 if bachelor = 1 and educ = 2 if master = 1.

seduc the same as educ but for a spouse.

weeks a total number of weeks worked during the year.

sweeks the same as weeks but for a spouse.

hours a usual number of working hours per week.

shours the same as hours but for a spouse.

wage the wage of the individual.

swage the same as wage but for a spouse.

lwage an inverse hyperbolic sine transformation of the hourly wage.

slwage the same as lwage but for a spouse.

state a state of residence. ...

Source

<https://www.census.gov/programs-surveys/cps.html>

References

Flood S, King M, Rodgers R, Ruggles S, Warren R, Westberry M (2022). Integrated Public Use Microdata Series, Current Population Survey: Version 10.0 [dataset]. doi: 10.18128/D030.V10.0.

Examples

```
data(cps)
model <- msel(work ~ age + bachelor + master, data = cps)
summary(model)
```

exogenous_fn *Modify exogenous variables in data frame*

Description

Change some values of the exogenous variables in a data frame.

Usage

```
exogenous_fn(exogenous, newdata)
```

Arguments

exogenous	list such that exogenous[[i]] represents the value (or a vector of values of the same size as nrow(newdata)) which will be exogenously assigned to the variable names(exogenous)[[i]] in newdata i.e., newdata[, names(exogenous)[i]] <- exogenous[[i]]. If newdata is NULL and exogenous is not NULL then newdata is set to object$data. This argument is especially useful for causal inference when some endogenous (dependent) variables should be exogenously assigned with some values i.e., in the right hand side of the formula and formula2. The purpose of the exogenous argument is just a convenience so equivalently it is possible to exogenously provide the values to variables via newdata argument.
newdata	data frame.

Details

This function changes exogenous variables in newdata.

Value

The function returns data frame which is similar to newdata but some values of this data frame are set according to exogenous.

fitted.msel *Extract Model Fitted Values*

Description

Extracts fitted values from 'msel' object

Usage

```
## S3 method for class 'msel'
fitted(object, ..., newdata = NULL)
```

Arguments

object	object of class 'msel'.
...	further arguments (currently ignored).
newdata	an optional data frame in which to look for variables with which to predict. If omitted, the original data frame is used. This data frame should contain values of dependent variables even if they are not actually needed for prediction (simply assign them with 0 values).

Value

Returns a numeric matrix. Its columns whose names coincide with the names of the ordinal and multinomial equations provide the index of the most probable category for each observation. Columns whose names coincide with the names of the continuous equations provide conditional expectations of the dependent variables in observable regimes for each observation.

---

formula.msel *Formulas of msel model.*

Description

Provides formulas associated with the object of class 'msel'.

Usage

```
## S3 method for class 'msel'
formula(x, ..., type = "formula", eq = NULL)
```

Arguments

x	object of class 'msel'.
...	further arguments (currently ignored).
type	character; if type = "formula" or type = 1 then function returns formulas of the ordinal equations. If type = "formula2" or type = 2 then function returns formulas of the continuous equations. If type = "formula3" or type = 3 then function returns formula of the multinomial equation.
eq	positive integer representing the index of the equation which formula should be returned. If NULL (default) then formulas for each equation will be returned as a list which i-th element associated with i-th equation.

Value

Returns a formula or a list of formulas depending on eq value.

`formula_merge` *Merge formulas*

Description

This function merges all variables of several formulas into a single formula.

Usage

```
formula_merge(..., type = "all")
```

Arguments

`...`	formulas to be merged such that there is a single element on the left hand side and various elements on the right hand side.
`type`	string representing the type of merge to be used. If `type = "all"` then both right hand side and left hand side elements of the formulas will be merged on the right hand side. If `type = "terms"` then only right hand side elements of the formulas will be merged on the right hand side. If `type = "var-terms"` then the result is the same as in case when `type = "terms"` but there will be left hand side element of the first formula on the left hand side of the merged formula.

Details

Merged formulas should have a single element on the left hand side and arbitrary number of elements on the right hand side.

Value

This function returns a formula which form depends on `type` input argument value. See 'Details' for additional information.

Examples

```
# Consider three formulas
f1 <- as.formula("y1 ~ x1 + x2")
f2 <- as.formula("y2 ~ x2 + x3")
f3 <- as.formula("y3 ~ y2 + x6")
# Merge these formulas in a various ways
formula_merge(f1, f2, f3, type = "all")
formula_merge(f1, f2, f3, type = "terms")
formula_merge(f1, f2, f3, type = "var-terms")
```

formula_split *Split formula by symbol*

Description

This function splits one formula into two formulas by symbol.

Usage

```
formula_split(formula, symbol = "|")
```

Arguments

formula	an object of class formula.
symbol	a string that is used to split formula into two formulas.

Details

The symbol should be on the right hand side of the formula.

Value

This function returns a list of two formulas.

Examples

```
formula_split("y ~ x1 + x2 | x2 + x3")
formula_split("y ~ x1 + x2 : x2 + x3", symbol = ":")
```

grad_msel *Gradient of the Log-likelihood Function of Multivariate Ordered Probit Model*

Description

Calculates gradient of the log-likelihood function of multivariate ordered probit model.

Usage

```
grad_msel(
  par,
  control_lnL,
  out_type = "grad",
  n_sim = 1000L,
  n_cores = 1L,
  regularization = NULL
)
```

Arguments

par	vector of parameters.
control_lnL	list with some additional parameters.
out_type	string representing the output type of the function.
n_sim	the number of random draws for multivariate normal probabilities.
n_cores	the number of cores to be used.
regularization	list of regularization parameters.

---

lnL_msel *Log-likelihood Function of Multivariate Ordered Probit Model*

Description

Calculates log-likelihood function of multivariate ordered probit model.

Usage

```
lnL_msel(
  par,
  control_lnL,
  out_type = "val",
  n_sim = 1000L,
  n_cores = 1L,
  regularization = NULL
)
```

Arguments

par	vector of parameters.
control_lnL	list with some additional parameters.
out_type	string representing the output type of the function.
n_sim	the number of random draws for multivariate normal probabilities.
n_cores	the number of cores to be used.
regularization	list of regularization parameters.

logLik.msel *Extract Log-Likelihood from a Fit of the msel Function.*

Description

Extract Log-Likelihood from a model fit of the `msel` function.

Usage

```
## S3 method for class 'msel'
logLik(object, ...)
```

Arguments

`object`	object of class "msel"
`...`	further arguments (currently ignored)

Details

If `estimator == "2step"` in `msel` then function may return `NA` value since the two-step estimator of the covariance matrix may not be positively defined.

Value

Returns an object of class 'logLik'.

loocv *Leave-one-out cross-validation*

Description

This function calculates root mean squared error (RMSE) for leave-one-out cross-validation of linear regression estimated via least squares method.

Usage

```
loocv(fit)
```

Arguments

`fit`	object of class `lm`.

Details

Fast analytical formula is used.

Value

This function returns a numeric value representing root mean squared error (RMSE) of leave-one-out cross-validation (LOOCV).

Examples

```
set.seed(123)
# Generate data according to linear regression
n <- 100
eps <- rnorm(n)
x <- runif(n)
y <- x + eps
# Estimate the model
model <- lm(y ~ x)
# Perform cross-validation
loocv(model)
```

---

lrtest_msel *Likelihood ratio test*

Description

This function performs chi-squared test for nested models.

Usage

```
lrtest_msel(model1, model2)
```

Arguments

model1	the first model.
model2	the second model.

Details

Arguments model1 and model2 should be objects of class that has implementations of logLik and nobs methods. It is assumed that either model1 is nested into model2 or vice versa. More precisely it is assumed that the model with smaller log-likelihood value is nested into the model with greater log-likelihood value.

Arguments model1 and model2 may be the lists of models. If model1 is a list of models then it is assumed that the number of degrees of freedom and log-likelihood of the first model are just a sum of degrees of freedom and log-likelihoods of the models in this list. Similarly for model2.

If model1 or model2 is a list then the number of observations of the associated models is calculated as the sum of the numbers of observations of the models in corresponding lists. However sometimes it may be misleading. For example, when bivariate probit model (full) is tested against two independent probit models (restricted). Then it will be assumed that the number of observations in the restricted model is twice the number of observations in the full model that is not the case. Fortunately it will not affect the results of the likelihood ratio test.

Value

The function returns an object of class `'lrtest_msel'` that is a list with the following elements:

- `n1` - the number of observations in the first model.
- `n2` - the number of observations in the second model.
- `ll1` - log-likelihood value of the first model.
- `ll2` - log-likelihood value of the second model.
- `df1` - the number of parameters in the first model.
- `df2` - the number of parameters in the second model.
- `restrictions` - the number of restrictions in the nested model.
- `value` - chi-squared (likelihood ratio) test statistic value.
- `p_value` - p-value of the chi-squared (likelihood ratio) test.

Examples

```
set.seed(123)
# Generate data according to linear regression
n   <- 100
eps <- rnorm(n)
x1  <- runif(n)
x2  <- runif(n)
y   <- x1 + 0.2 * x2 + eps
# Estimate full model
model1 <- lm(y ~ x1 + x2)
# Estimate restricted (nested) model
model2 <- lm(y ~ x1)
# Likelihood ratio test results
lrtest_msel(model1, model2)
```

`msel` *Multivariate and multinomial sample selection and endogenous switching models with multiple outcomes.*

Description

This function allows to estimate parameters of the multivariate and multinomial sample selection and endogenous switching models with multiple outcomes. Both maximum-likelihood and two-step estimators are implemented.

Usage

```
msel(
  formula = NA,
  formula2 = NA,
  formula3 = NA,
  data = NULL,
```

```
  groups = NA,
  groups2 = NA,
  groups3 = NA,
  marginal = list(),
  opt_type = "optim",
  opt_args = NA,
  start = NULL,
  estimator = "ml",
  cov_type = "mm",
  degrees = NA,
  degrees3 = NA,
  n_sim = 1000,
  n_cores = 1,
  control = list(),
  regularization = list(),
  type3 = "logit",
  cluster = NA
)
```

Arguments

formula	a list which i-th element is an object of class "formula" describing the form of the linear predictor (index) of the i-th ordinal equation. Mean and variance equations should be separated by the 'I' symbol.
formula2	a list which i-th element is an object of class "formula" describing the form of the linear predictor (index) of the i-th continuous equation.
formula3	an object of class "formula" describing the form of the linear predictor (index) of the multinomial equation.
data	a data frame containing the variables in the model.
groups	a matrix which (i, j)-th element is the j-th ordinal category (value starting from 0) of the i-th dependent ordinal variable. Each row of this matrix describes observable (in data) combination of categories - values of the ordinal dependent variables i.e., from the left hand side of formula. Special category '-1' means that variable in the j-th column is unobservable when other dependent variables have particular values i.e., given in the same row. See 'Details' for additional information.
groups2	the same as the groups argument but for the continuous dependent variables from formula2. See 'Details' for additional information.
groups3	the same as the groups argument but for the multinomial dependent variable from formula3. See 'Details' for additional information.
marginal	a list such that marginal[[i]] represents parameters of the marginal distribution of the random error of the i-th ordered equation and names(marginal)[i] is a name of this distribution. Marginal distributions are the same as in pmnorm.
opt_type	a character representing the optimization function to be used. If opt_type = "optim" then optim will be used. If opt_type = "gena" then gena will be applied i.e., a genetic algorithm. If opt_type = "pso" then pso will be used i.e., a particle swarm optimization.

`opt_args` a list of input arguments for the optimization function selected via the `opt_type` argument. See 'Details' for additional information.

`start` a numeric vector of initial parameters' values. It will be used as a starting point for the optimization purposes. It is also possible to provide an object of class `'msel'`; then its `'par'` element will be used as a starting point.

`estimator` a character determining estimation method. If `estimator = "ml"` then maximum-likelihood method will be used. If `estimator = "2step"` then two-step estimation procedure similar to Heckman's method will be applied. See 'Details' for additional information.

`cov_type` a character determining the type of the covariance matrix estimate to be returned. First, suppose that `estimator = "ml"` then the following estimators are available. If `cov_type = "hessian"` then negative inverse of Hessian matrix will be applied. If `cov_type = "gop"` then inverse of Jacobian outer products will be used. If `cov_type = "sandwich"` then sandwich covariance matrix estimator will be applied. If `cov_type = "no"` then a matrix of `NA` values will be used. If `cov_type = "mm"` (default) then sandwich estimator is used along with the penalized log-likelihood function. Therefore if there is no regularization then `"mm"` and `"sandwich"` estimators are the same. Second, suppose that `estimator = "2step"`. Then only available options are `cov_type = "mm"` and `cov_type = "no"`.

`degrees` a vector of non-negative integers such that `degrees[i]` represents the degree of the polynomial whose elements are selectivity terms associated with the `i`-th ordered equation. See 'Details' for additional information.

`degrees3` a vector of non-negative integers such that `degrees3[i]` represents the degree of the polynomial whose elements are selectivity terms associated with the `i`-th multinomial equation. See 'Details' for additional information.

`n_sim` an integer representing the number of GHK draws when there are more than 3 ordered equations. Otherwise alternative (much more efficient) algorithms will be used to calculate multivariate normal probabilities.

`n_cores` a positive integer representing the number of CPU cores used for the parallel computing. If possible it is highly recommended to set it equal to the number of available physical cores.

`control` a list of control parameters. See 'Details' for additional information.

`regularization` a list of control parameters for regularization. Element `ridge_ind` is a vector of indexes of parameters subject to regularization according to the quadratic (ridge) penalty function. These indexes correspond to parameters from `par` output element. Set `show_ind` argument of `summary.msel` to `TRUE` to see these indexes. Element `ridge_scale` is a numeric vector of weights of the ridge penalty function. Element `ridge_location` is a numeric vector of values to be subtracted from the parameters before they pass into the penalty function. Elements `lasso_ind`, `lasso_scale` and `lasso_location` are the same but for the lasso penalty term.

`type3` a character determining the type of the multinomial model to be considered. If `type3 = "logit"` then multinomial logit model will be used. If `type3 = "probit"` then multinomial probit model will be applied. See 'Details' for additional information.

`cluster` an object of class `"formula"` describing the variables which interactions determine the clusters used to estimate the asymptotic covariance matrix when `cov_type = "sandwich"` or `cov_type = "mm"`. Alternatively this argument may be specified as a character vector of these variables.

Details

This function allows the estimation of multivariate and multinomial sample selection and endogenous switching models with multiple outcomes. These models are the systems of ordinal, continuous and multinomial equations described by `formula`, `formula2` and `formula3` respectively.

Ordinal equations

Argument `formula` determines the regressors of the multivariate ordered probit model with the heteroscedastic random errors:

$$z^*_{ji} = w_{ji}\gamma_j + \sigma^*_{ji}u_{ji},$$

$$\sigma^*_{ji} = \exp(w^*_{ji}\gamma^*_j), \quad u_i \sim N\left(\begin{bmatrix}0\\ \vdots \\ 0\end{bmatrix}, \Sigma\right), i.i.d.,$$

$$z_{ji} = \begin{cases} 0, \text{ if } z^*_{ji} \leq c_{j1} \\ 1, \text{ if } c_{j1} < z^*_{ji} \leq c_{j2} \\ 2, \text{ if } c_{j2} < z^*_{ji} \leq c_{j3} \\ \vdots \\ m_{J-1}, \text{ if } z^*_{(J-1)i} > c_{(J-1)m_{j-1}} \end{cases},$$

$$z_i = (z_{1i}, ..., z_{Ji})^T, \quad u_i = (u_{1i}, u_{2i}, ..., u_{Ji})^T,$$

$$i \in \{1, 2, ..., n\}, \quad j \in \{1, 2, ..., J\},$$

where:

- n - the number of observations. If there are no omitted observations then n equals `nrow(data)`.
- J - the number of ordinal equations which equals `length(formula)`.
- z^*_{ji} - an unobservable (latent) value of the `j`-th dependent ordinal variable.
- z_{ji} - an observable (ordinal) value of the `j`-th dependent ordinal variable.
- m_j - the number of categories of $z_{ji} \in \{0, 1, ..., m_j\}$.
- c_{jk} - the k-th cut (threshold) of the `j`-th dependent ordinal variable.
- w_{ji} - the regressors of the j-th mean equation which should be described in `formula[[j]]`.
- γ_j - the regression coefficients of the j-th mean equation.
- $w_{ji}\gamma_j$ - the linear predictor (index) of the j-th mean equation.
- u_i - a multivariate normal random vector whose elements are standard normal random variables.
- Σ - a correlation matrix of u_i so $\Sigma_{t_1t_2} = \text{corr}\,(u_{it_1}, u_{it_2})$.
- σ^*_{ji} - a heteroscedastic standard deviation.
- $\sigma^*_{ji}u_{ji}$ - the heteroscedastic random errors.

- w^*_{ji} - the regressors of the j-th variance equation which should be described in `formula[[j]]` after 'I' symbol.
- γ^*_j - the regression coefficients of the j-th variance equation.
- $w^*_{ji}\gamma^*_j$ - the linear predictor (index) of the j-th variance equation.

Constant terms (intercepts) are excluded from the model for identification purposes. If z_{ji} is a binary variable then $-c_{j1}$ may be interpreted as a constant term of the j-th ordinal equation. If all z_{ji} are binary variables then the model becomes a multivariate probit.

By default the joint distribution of u_i is multivariate normal. However, by using the `marginal` argument, it is possible to consider the joint distribution that is determined by the Gaussian copula and possibly non-normal marginal distributions. Specifically, `names(marginal)[i]` is the name of the marginal distribution of u_{ji} and `marginal[[i]]` is the number of parameters of this distribution. The marginal distributions are the same as in `pmnorm`.

Multinomial equation

Argument `formula3` determines the regressors of the multinomial equation:

$$\tilde{z}^*_{ji} = \tilde{w}_i\tilde{\gamma}_j + \tilde{u}_{ji}, \qquad j \in \{0, 1, ..., \tilde{J}-1\},$$

$$\tilde{z}_i = \underset{t\in\{0,...,\tilde{J}-1\}}{\text{argmax}} \tilde{z}^*_{ti}, \qquad \tilde{u}_i = (\tilde{u}_{0i}, \tilde{u}_{1i}, ..., \tilde{u}_{(\tilde{J}-1)i})^T,$$

where:

- $\tilde{J}$ - the number of the alternatives i.e., possible values of the dependent variable of the multinomial equation.
- $\tilde{z}^*_{ji}$ - an unobservable (latent) value of the `j`-th alternative. Usually $\tilde{z}^*_{ji}$ is interpreted as a utility of the `j`-th alternative.
- $\tilde{z}_i$ - a selected alternative i.e., one which provides the greatest utility $\tilde{z}^*_{ji}$.
- $\tilde{w}_i$ - the regressors of the multinomial equation which should be described in `formula3` and are assumed to be the same for all the alternatives.
- $\tilde{\gamma}_j$ - the regression coefficients of the j-th alternative's equation.
- $\tilde{w}_i\tilde{\gamma}_j$ - the linear predictor (index) of the j-th alternative's equation.
- $\tilde{u}_i$ - a vector of random errors.

For identification purposes it is assumed that the regression coefficients of the last alternative are zero $\tilde{\gamma}_{\tilde{J}-1} = (0, ..., 0)^T$.

The joint distribution of $\tilde{u}_i$ depends on the value of the `type3` argument. If `type3 = "logit"` then a multinomial logit model is considered. It assumes that $\tilde{u}_{ji}$ are independent and their marginal distribution is Gumbel (error value distribution). If `type3 = "probit"` then a multinomial probit model is used, so it is assumed that the joint distribution of $\tilde{u}_i$ is multivariate normal with zero mean and the covariance matrix $\tilde{\Sigma}$. For identification purposes it is also assumed that $\tilde{\Sigma}_{11} = 1$ so $Var(\tilde{u}_{0i}) = 1$. In addition, $\tilde{u}_{(\tilde{J}-1)i} = 0$ which implies $\tilde{\Sigma}_{t(\tilde{J})} = \tilde{\Sigma}_{(\tilde{J})t} = 0$ for all $t \in \{0, ..., \tilde{J}-1\}$.

Continuous equations

Argument `formula3` determines the regressors of the continuous equations:

$$y^{(v)}_{r^{(v)}i} = x^{(v)}_i \beta^{(v)}_{r^{(v)}} + \varepsilon^{(v)}_{r^{(v)}i},$$

$$r^{(v)} \in \{0, 1, ..., R^{(v)} - 1\}, \quad v \in \{1, ..., V\},$$

where:

- V - the number of continuous equations.
- $r^{(v)}$ - the regime of the v-th continuous equation.
- $y^{(v)}_{r^{(v)}i}$ - the $r^{(v)}$-th potential outcome (in the sense of the Neyman-Rubin framework) of the v-th continuous equation.
- $R^{(v)}$ - the number of potential outcomes of the v-th continuous equation.
- $x^{(v)}_i$ - the regressors of the v-th continuous equation. They are provided via `formula2[[v]]`.
- $\beta^{(v)}_{r^{(v)}}$ - regression coefficients of the v-th continuous equation in the regime $r^{(v)}$.
- $\varepsilon^{(v)}_{r^{(v)}i}$ - a random error of the v-th continuous equation in the regime $r^{(v)}$.

Estimation of the model with multivariate ordinal and multinomial equations

If `formula2` is not provided then maximum-likelihood estimator is used with `estimator = "ml"` to estimate the parameters of the multivariate ordinal and multinomial equations.

If both `formula` and `formula3` are provided then parameters of the multivariate ordered and multinomial equations are estimated under the assumption that u_i is independent of $\tilde{u}_i$. Therefore the estimates are identical to those obtained by separate estimation of these models. We may relax the independence assumption during future updates.

Estimation of the model with continuous equations

If `formula` and `formula3` are not provided then it is assumed that each continuous equation has only one regime so $R^{(v)} = 1$ for each $v \in \{1, ..., V\}$.

If `estimator = "ml"` then the maximum-likelihood estimator is used under the assumption that the distribution of random errors is multivariate normal. If `estimator = "2step"` then the V-stage least squares estimator is used. In the latter case v-th equation is estimated by the least squares estimator and for every v_0 such that $v_0 < v$ if $y^{(v_0)}_{0i}$ is included in $x^{(v)}_i$ then $y^{(v_0)}_{0i}$ is substituted with its estimate $\hat{y}^{(v_0)}_{0i}$ obtained on the v_0-th step. Therefore if $v_0 < v$ then if some endogenous variable appears on the left hand side of `formula2[[v]]` it should not appear on the right hand side of `formula2[[v0]]`.

Estimation of the model with ordinal outcomes and multivariate ordered selection

Suppose that z_i represents the ordinal potential outcomes while the observable values are $z^{(o)}_i = g^{(1)}(z_i)$, where the function $g^{(1)}(z_i)$ converts unobservable values of z_i to -1. Therefore $z^{(o)}_{ji}$ instead of z_i appears on the left hand side of the `formula[[j]]`.

For example, consider a binary variable for the employment status (0 - unemployed, 1 - employed) and an ordinal variable z_{2i} (ranging from 0 to 2) for job satisfaction (0 - unsatisfied, 1- satisfied, 2 - highly satisfied). Then z_{2i} is observable only when z_{1i} equals 1 since job satisfaction is observable only for working individuals. Consequently $z^{(o)}_{2i}$ should be equal to -1 (minus one) whenever z_{1i} equals 0:

$$z^{(o)}_i = g^{(1)}(z_i) = \begin{cases} (1,2), \text{ if } z_i = (1,2) \\ (1,1), \text{ if } z_i = (1,1) \\ (1,0), \text{ if } z_i = (1,0) \\ (0,-1), \text{ otherwise} \end{cases}$$

Rows of the matrix groups determine all possible values of the function $g^{(1)}(z_i)$. In this particular example matrix groups should have the following form:

$$\text{groups} = \begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

Notice that in this particular example it is necessary to ensure that in data if $z_{1i}^{(o)}$ equals 0 then $z_{2i}^{(o)}$ equals -1. Also in this case matrix groups will be created automatically so there is no need to provide it manually.

By using the groups argument it is straightforward to consider various other models with ordinal outcomes and multivariate ordered selection mechanisms.

If some values of the ordinal variables z_{ji} are missing (at random) i.e., take an NA value then the contribution of other ordinal dependent variables (for the i-th observation) still may be included into the likelihood function by manually substituting NA with -1 in the data. However, ensure that this particular (missing) z_{ji} is not a regressor for another dependent variable that may happen in the hierarchical systems.

It is useful to use the groups argument to consider causal inference models. For example, suppose that z_{1i} represents a binary treatment variable for the university diploma (0 - no diploma, 1 - has diploma). Also, z_{2i} is a binary potential outcome for the employment status (0 - unemployed, 1 - employed) of the individual if she has university diploma. Finally, z_{3i} is a binary potential outcome for the employment status (0 - unemployed, 1 - employed) if the individual does not have a university diploma. Since z_{2i} is observable only if $z_{1i} = 1$ and z_{3i} is observable only when $z_{2i} = 0$ we get:

$$z_i^{(o)} = \begin{cases} (1, 1, -1), \text{ if } z_{1i} = 1 \text{ and } z_{2i} = 1 \\ (1, 0, -1), \text{ if } z_{1i} = 1 \text{ and } z_{2i} = 0 \\ (0, -1, 1), \text{ if } z_{1i} = 0 \text{ and } z_{3i} = 1 \\ (0, -1, 0), \text{ if } z_{1i} = 0 \text{ and } z_{3i} = 0 \end{cases}.$$

Therefore to estimate this model it is necessary to ensure that in data we have $z_{2i}^{(o)} = -1$ when $z_{1i}^{(o)} = 0$ and $z_{3i}^{(o)} = -1$ if $z_{1i}^{(o)} = 1$. Also the groups argument should include all possible values of $z_i^{(o)}$:

$$\text{groups} = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 0 & -1 \\ 0 & -1 & 1 \\ 0 & -1 & 0 \end{bmatrix}.$$

Estimation of the model with continuous outcomes and multivariate ordered selection

To simplify the notation, suppose that there is only one continuous equation with multiple regimes:

$$y_{ri} = x_i \beta_r + \varepsilon_{ri},$$

$$y_i = \begin{cases} \text{unobservable, if } g^{(2)}\left(z_i^{(o)}\right) = -1 \\ y_{0i}, \text{ if } g^{(2)}\left(z_i^{(o)}\right) = 0 \\ y_{1i}, \text{ if } g^{(2)}\left(z_i^{(o)}\right) = 1 \\ \vdots \\ y_{(R^*-1)i}, \text{ if } g^{(2)}\left(z_i^{(o)}\right) = R^* - 1 \end{cases},$$

$$\varepsilon_i = (\varepsilon_{0i}, \varepsilon_{1i}, ..., \varepsilon_{(R^*-1)i}), \quad r \in \{0, 1, ..., R^* - 1\},$$

where:

- y_i - an observable continuous outcome.
- r - the index of the potential outcome.
- R^* - the number of the regimes.
- y_{ri} - a continuous potential outcome i.e., the value of y_i in the regime r.
- x_i - the vector of regressors provided in `formula2`.
- β_r - the regression coefficients in the r-th regime.
- $g^{(2)}(z_i^{(o)})$ - a function determining which potential outcome is observable depending on the observable values of the ordinal variables $z_i^{(o)}$.
- ε_i - a vector of random errors.

The value of the `groups2[i]` argument specifies the value of $g^{(2)}(z_i^{(o)})$ when $z_i^{(o)}$ equals `groups[i, ]`. The values of y_i in `data` such that $g^{(2)}(z_i^{(o)}) = -1$ should be set to `NA`.

For example, consider a system of equations for wage y_i, employment status z_{1i} (0 - unemployed, 1 - employed) and job satisfaction z_{2i} (0 - unsatisfied, 1- satisfied, 2 - highly satisfied). Notice that wage and job satisfaction are observable only for working individuals. Also suppose that wage is unobservable for unsatisfied workers and observable in different regimes for other workers. Namely, for satisfied workers $z_{2i} = 1$ we observe y_{0i} and for highly satisfied workers $z_{2i} = 2$ we observe y_{1i}.

To estimate this model it is necessary to manually specify the structure of the equations via the `groups` and `groups2` arguments by providing all possible combinations of the ordinal variables and the regimes of the continuous equation:

$$\text{groups} = \begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 1 & 0 \\ 0 & -1 \end{bmatrix}, \text{groups2} = \begin{bmatrix} 1 \\ 0 \\ -1 \\ -1 \end{bmatrix}.$$

Notice that `groups2[1] = 1` indicates that when `groups[1, ] = c(1, 2)` i.e. $z_{1i} = 1$ and $z_{2i} = 2$ we observe y_i in regime `1` corresponding to the wage of highly satisfied workers y_{1i}. Similarly `groups2[2] = 0` indicates that when `groups[2, ] = c(1, 1)` i.e., $z_{1i} = 1$ and $z_{2i} = 1$ we observe y_i in the regime `0` corresponding to the wage of the satisfied workers y_{0i}. Also, `groups2[3] = -1` means that when `groups[3, ] = c(1, 0)` i.e., $z_{1i} = 1$ and $z_{2i} = 0$ we do not observe the wage of the worker y_i since he is unsatisfied. Finally, `groups2[4] = -1` means that when `groups[4, ] = c(0, -1)` i.e., $z_{1i} = 0$ and $z_{2i}^{(o)} = -1$ we do not observe the wage of the worker y_i since he is unemployed.

If the joint distribution of ε_i and u_i is multivariate normal then according to Kossova and Potanin (2018):

$$y_{ri} = x_i\beta_r + \sum_{j=1}^{J}\Sigma_{j(r+1)}^{(12)}\lambda_{ji} + \varepsilon_{ri}^{*},$$

where:

$$\varepsilon_{ri}^{*} = \varepsilon_{ri} - E(\varepsilon_i|z_{1i},...z_{Ji}) = \varepsilon_{ri} - \sum_{j=1}^{J}\Sigma_{j(r+1)}^{(12)}\lambda_{ji},$$

$$\lambda_{ji} = \lambda_{ji}^{(1)} + \lambda_{ji}^{(2)}, \qquad \Sigma_{j(r+1)}^{(12)} = cov(u_{ji}, \varepsilon_{r+1}),$$

$$\lambda_{ji}^{(1)} = \begin{cases} 0, \text{ if } z_{ji} = 0 \\ -\frac{\partial \ln P_i^{*}}{\partial a_{ji}}, \text{ otherwise} \end{cases}, \qquad \lambda_{ji}^{(2)} = \begin{cases} 0, \text{ if } z_{ji} = m_j - 1 \\ -\frac{\partial \ln P_i^{*}}{\partial b_{ji}}, \text{ otherwise} \end{cases},$$

$$P_i^{*}(a_{1i},...,a_{Ji};b_{1i},...,b_{Ji}) = P(a_{1i} \le u_{1i} \le b_{1i},....,a_{Ji} \le u_{Ji} \le b_{Ji}),$$

$$a_{ji} = \begin{cases} -\infty, \text{ if } z_{ji} = 0 \\ \frac{c_{jz_{ji}} - w_{ji}\gamma_j}{\sigma_{ji}^{*}}, \text{ otherwise} \end{cases}, \qquad b_{ji} = \begin{cases} \infty, \text{ if } z_{ji} = m_j - 1 \\ \frac{c_{j(z_{ji}+1)} - w_{ji}\gamma_j}{\sigma_{ji}^{*}}, \text{ otherwise} \end{cases}.$$

Notice that the regression equation has selectivity terms λ_{ji} which may be correlated with x_i. Therefore if random errors u_i and ε_i are correlated, the least squares estimator of x_i on y_{ri} is inconsistent. To get consistent estimates of β_r it is possible to use maximum-likelihood `estimator = "ml"` or two-step (method of moments) `estimator = "2step"` estimator.

If the two-step estimator is used then on the first step λ_{ji} are estimated as the functions of the estimates of the multinomial heteroscedastic ordered probit model. On the second step least squares regression of y_{ri} on x_i and the first step estimates $\hat{\lambda}_{ji}$ is used to estimate β_r and $\Sigma_{j(r+1)}^{(12)}$.

If the joint distribution of random errors ε_i, u_i is not multivariate normal then λ_{ji} terms may enter the continuous equation non-linearly. Following Newey (2009) and E. Kossova, L. Kupriianova, and B. Potanin (2020) it is assumed that:

$$y_{ri} = x_i\beta_r + \zeta_i\tau_r + \varepsilon_i^{*},$$

where τ_r is a n_λ-dimensional column vector and:

$$\zeta_i = (\zeta_1(\lambda_i),...,\zeta_{n_\lambda}(\lambda_i)), \qquad \lambda_i = (\lambda_{1i},...,\lambda_{Ji}).$$

Functions $\zeta_t(\lambda_i)$ are specified manually by the user in the `formula2` inside `I()`. For example, to specify $\zeta_t(\lambda_i) = \lambda_{1i} \times \lambda_{2i}$ it is sufficient to have a term `I(lambda1 * lambda2)` in `formula2`. Notice that to avoid confusion, no variables in `data` should have the names containing `"lambda"`. Otherwise these variables will be dropped.

It is possible to specify $\zeta_t(\lambda_i)$ functions as a polynomial without interaction terms by using the `degrees` argument. Specifically, if `degrees[j] = t` then `lambdaj`, `I(lambdaj ^ 2)`,...,`I(lambdaj ^ t)` terms are added to `formula2`. However, if the `degrees` argument is used then no functions of `lambdaj` should be provided manually in `formula2`. Otherwise it will be assumed that `degrees` is a vector of zeros. Also, if `estimator = "2step"`, there are no `lambdaj` terms in `formula2` and `degrees` is `NA`, then `degrees` will be converted to a vector of ones.

If there are multiple continuous equations then `formula2` should be a list of formulas. Further, if `estimator = "2step"` then the second step is a V-stage least squares estimator with `lambda`

terms. If they are provided via the `degrees` argument, then it should be a matrix whose `v`-th row corresponds to the `v`-th continuous equation.

For example, consider the previous example with additional continuous equation for working hours $y_i^{(2)}$ which does not vary with the satisfaction of the workers z_{2i} but is observable only for the employed individuals $z_{1i} = 1$. To estimate the system with this additional continuous equation simply substitute all $y_i^{(2)}$ (such that $z_{1i} = 0$) in `data` with `NA` and specify:

$$\text{groups} = \begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 1 & 0 \\ 0 & -1 \end{bmatrix}, \text{groups2} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ -1 & 0 \\ -1 & -1 \end{bmatrix}.$$

Notice that `groups2[, 1]` describes the regimes of the wage equation $y_i^{(1)}$ while `groups[, 2]` contains the regimes of the hours equation $y_i^{(2)}$. Note that the formula of the first equation (wage) should be specified in `formula2[[1]]` and the formula of the second equation should be provided via `formula2[[2]]` i.e., as the first and the second elements in a `formula2` list correspondingly.

If the `marginal` argument is used, then aforementioned formula of λ_{ji} is slightly modified to address the non-normal marginal distribution of u_{ji}.

Estimation of the model with continuous outcomes and multinomial selection

The only difference with the previous model is that the observable value of the continuous equation depends on the value of the multinomial equation described in `formula3`. Therefore $g^{(2)}(z_i^{(o)})$ is substituted with $g^{(3)}(\tilde{z}_i)$. Also `groups3` argument instead of `groups` is used. Since there is only a single multinomial equation argument `groups3` is a vector. If `groups3[k] = q` and `groups2[k, v] = r` then $\tilde{z}_i = q$ implies $y_i^{(v)} = y_{ri}^{(v)}$. Recall that a special value `r = -1` implies that $y_i^{(v)}$ is unobservable.

Only the two-step `estimator = "2step"` estimator of this model is available, which is similar to the one described above. The only difference is the formula used to estimate selectivity terms. If `type = "logit"`, then the two-step estimator of Dubin-McFadden (1984) is used, so selectivity terms are as follows:

$$\lambda_{ji} = \begin{cases} -\ln P(\tilde{z}_i = j), \text{if } \tilde{z}_i = j \\ \frac{P(\tilde{z}_i = j) \ln P(\tilde{z}_i = j)}{1 - P(\tilde{z}_i = j)}, \text{otherwise} \end{cases},$$

where $j \in \{0, ..., \tilde{J} - 1\}$ and:

$$P(\tilde{z}_i = j) = \begin{cases} \frac{e^{(\tilde{w}_i \tilde{\gamma}_j)}}{1 + \sum_{q=0}^{\tilde{J}-2} e^{(\tilde{w}_i \tilde{\gamma}_q)}}, \text{if } j \neq \tilde{J} - 1 \\ \frac{1}{1 + \sum_{q=0}^{\tilde{J}-2} e^{(\tilde{w}_i \tilde{\gamma}_q)}}, \text{otherwise} \end{cases}.$$

If `type = "probit"`, then the two-step estimator of Kossova and Potanin (2022) is used with the selectivity terms of the form:

$$\lambda_{ji} = \left(A^{(\tilde{z}_i)} \lambda_i^* \right)_j,$$

where $j \in \{0, ..., \tilde{J}-2\}$ and:

$$A^{(j)}_{t_1 t_2} = \begin{cases} 1, \text{ if } t_1 = j+1 \\ -1, \text{ if } t_1 < j+1 \text{ and } t_1 = t_2 \\ -1, \text{ if } t_1 > j+1 \text{ and } t_1 = t_2 + 1 \\ 0, \text{ otherwise} \end{cases},$$

$$t_1, t_2 \in \{1, ..., \tilde{J}-1\},$$

$$\lambda^*_i = \nabla \ln P(\tilde{z}_i) = \nabla \ln F_{\tilde{u}^{(ji)}}\left(\tilde{z}^{(ji)}_1, ..., \tilde{z}^{(ji)}_{\tilde{J}-1}\right) = \left(\lambda^*_{1i}, ..., \lambda^*_{(\tilde{J}-1)i}\right)^T,$$

$$\tilde{u}^{(ji)} = \left(\tilde{u}_{0i} - \tilde{u}_{ji}, \tilde{u}_{1i} - \tilde{u}_{ji}, ..., \tilde{u}_{(j-1)i} - \tilde{u}_{ji}, \tilde{u}_{(j+1)i} - \tilde{u}_{ji}, ..., \tilde{u}_{(\tilde{J}-1)i} - \tilde{u}_{ji}\right),$$

$$\tilde{z}^{(ji)} = \tilde{w}_i \left((\tilde{\gamma}_j - \tilde{\gamma}_0), (\tilde{\gamma}_j - \tilde{\gamma}_1), ..., (\tilde{\gamma}_j - \tilde{\gamma}_{j-1}), (\tilde{\gamma}_j - \tilde{\gamma}_{j+1}), ..., (\tilde{\gamma}_j - \tilde{\gamma}_{\tilde{J}-1})\right).$$

Probabilities $P(z_i)$ are calculated by using a cumulative distribution function of the multivariate normal distribution with zero mean and the covariance matrix of $\tilde{u}^{(ji)}$.

In the `formula2`, selectivity terms associated with the multinomial equation should be named `lambdaj_mn` instead of `lambdaj`. Argument `degrees3` is similar to `degrees`.

Consider a simple example of this model. Suppose that $\tilde{z}_i$ is a multinomial variable for the employment status (0 - unemployed, 1 - working in IT sector, 2 - working in other sector). Wage y_i is unobservable for the unemployed $\tilde{z}_i = 0$, equals y_{0i} in the IT sector $\tilde{z}_i = 1$, and equals y_{1i} in other sectors $\tilde{z}_i = 2$. To estimate this model set `groups3 = (0, 1, 2)` and `groups2 = (-1, 0, 1)`. Then substitute all y_i such that $\tilde{z}_i = 0$ with `NA`.

Estimation of the model with continuous outcomes and mixed selection

It is possible to consider the model with continuous outcomes and both multinomial and ordinal selection equations. Recall that it is assumed that random errors of the ordered and multinomial equations are independent. Therefore if `formula` and `formula3` are provided then both `lambdaj` and `lambdaj_mn` are included in `formula2`. Only the two-step estimator `estimator = "2step"` is available for this model.

Missing values

If any of the left hand side variables (regressors) of `formula[[j]]` is missing then the right hand side variable of `formula[[j]]` will be set to `NA` in `data`. Similar is true for `formula2` and `formula3`.

Additional information

Functions `pmnorm` and `dmnorm` are used internally for calculation of multivariate normal probabilities, densities and their derivatives.

Currently `control` has no input arguments intended for the users. This argument is used for some internal purposes of the package.

Optimization always starts with `optim`. If `opt_type = "gena"` or `opt_type = "pso"` then `gena` or `pso` is used to proceed with optimization starting from the initial point provided by `optim`. Manual arguments to `optim` should be provided in the form of a list through `opt_args$optim`. Similarly `opt_args$gena` and `opt_args$pso` provide manual arguments to `gena` and `pso`. For example, to provide the Nelder-Mead optimizer to `optim` and restrict the number of genetic algorithm iterations to 10 make `opt_args = list(optim = list(method = "Nelder-Mead"),gena = list(maxiter = 10))`.

If `estimator = "2step"` then it is possible to precalculate the first step model with `msel` function and pass it through the `formula` argument. It allows one to experiment with various `formula2` and `degrees` specifications without extra computational burden associated with the first step estimation.

If `estimator = "2step"` then the method of moments estimator of the asymptotic covariance matrix is used as described in Meijer and Wansbeek (2007).

Value

This function returns an object of class `"msel"`. It is a list which may contain the following elements:

- `par` - a vector of the estimates of the parameters.
- `estimator` - the same as the `estimator` input argument.
- `type3` - the same as the `type3` input argument.
- `formula` - the same as `formula` input argument but all elements are coerced to `"formula"` type.
- `formula2` - the same as `formula2` input argument but all elements are coerced to `"formula"` type.
- `formula3` - the same as `formula3` input argument but all elements are coerced to `"formula"` type.
- `model1` - an object of class `"msel"` with the first step estimation results.
- `data` - the same as `data` input argument but without missing (at random) values.
- `W_mean` - a list such that `W_mean[[j]]` is a matrix of the regressors w_{ji} of the mean equation of z^*_{ji}.
- `W_var` - a list such that `W_var[[j]]` is a matrix of the regressors w^*_{ji} of the variance equation of z^*_{ji}.
- `X` - a list such that `X[[v]]` is a numeric matrix of regressors $x_i^{(v)}$ of the `v`-th continuous equation $y_i^{(v)}$.
- `W_mn` - a matrix of the regressors $\tilde{w}_i$ of the multinomial equation of $\tilde{z}^*_{ji}$.
- `dependent` - a numeric matrix which j-th column `dependent[, j]` is a vector of the ordinal dependent variable $z_{ji}^{(o)}$ values.
- `dependent2` - a numeric matrix which v-th column `dependent2[, v]` is a vector of the continuous dependent variable $y_i^{(v)}$ values.
- `dependent3` - a numeric vector of values of the multinomial dependent variable $\tilde{z}_j$.
- `groups` - the same as `groups` input argument or automatically generated matrix representing the structure of the system of equations. Please, see 'Details' section above for more information.
- `groups2` - the same as `groups2` input argument or automatically generated matrix representing the structure of the system of equations. Please, see 'Details' section above for more information.
- `groups3` - the same as `groups3` input argument or automatically generated matrix representing the structure of the system of equations. Please, see 'Details' section above for more information.

- `marginal` - the same as `marginal` input argument.
- `ind` - a list containing some indexes partition of the model (not intended for the users).
- `start` - the same as the `start` input argument.
- `twostep` - a list such that `twostep[[v]][[r + 1]]` is an object of class `"lm"` associated with the second step estimates of the `v`-th equation in the regime `r`.
- `y_pred` - a numeric matrix such that `y_pred[, v]` is a second step prediction of the $y_i^{(v)}$.
- `coef` - a list which j-th element `coef[[j]]` is a numeric vector representing $\hat{\gamma}_j$.
- `coef_var` - a list which j-th element `coef_var[[j]]` is a numeric vector representing $\hat{\gamma}_j^*$.
- `cuts` - a list which j-th element `cuts[[j]]` is a numeric vector representing $\hat{c}_j$.
- `coef2` - a list of numeric matrices such that `coef2[[v]][, r + 1]` is a numeric vector representing $\hat{\beta}^{(v)}_{r^{(v)}}$.
- `sigma` - a numeric matrix such that `sigma[j, t]` is a numeric value representing $\widehat{Cov}(u_{ji}, u_{ti})$.
- `var2` - a list of numeric vectors such that `var2[[v]][r + 1]` represents $\widehat{Var}(\varepsilon_{ri}^{(v)})$.
- `cov2` - a list of numeric matrices which element `sigma2[[v]][j, r + 1]` represents $\widehat{Cov}(u_{ji}, \varepsilon_{ri}^{(v)})$.
- `sigma2` - a list of numeric matrices representing the estimates of the covariances between random errors of the continuous equations in different regimes $\widehat{Cov}(\varepsilon^{(v)}_{r^{(v)}i}, \varepsilon^{(t)}_{r^{(t)}i})$.
- `marginal_par` - a list such that `marginal_par[[j]]` is a numeric vector of estimates of the parameters of the marginal distribution of u_{ji}.
- `coef3` - a numeric matrix such that `coef3[j + 1, ]` is a numeric vector representing $\hat{\tilde{\gamma}}_j$.
- `sigma3` - a numeric matrix such that `sigma3[j + 1, t + 1]` is a numeric value representing $\widehat{Cov}(\tilde{u}_{ji}, \tilde{u}_{ti})$.
- `lambda` - a numeric matrix such that `lambda[i, j]` corresponds to $\hat{\lambda}_{ji}$ of the ordinal equations.
- `lambda_mn` - a numeric matrix such that `lambda_mn[i, j]` corresponds to $\hat{\lambda}_{ji}$ of the multinomial equation.
- `H` - if `estimator = "ml"` then `H` is a Hessian matrix of the log-likelihood function. If `estimator = "2step"` then `H` is a numeric matrix of the derivatives of mean sample scores with respect to the estimated parameters.
- `J` - if `estimator = "ml"` then `J` is a Jacobian matrix of the log-likelihood function. If `estimator = "2step"` then `J` is a numeric matrix such that `J[, s]` is a vector of sample scores associated with the parameter `par[s]`.
- `cov_type` - the same as `cov_type` input argument.
- `cov` - an estimate of the asymptotic covariance matrix of the parameters' estimator.
- `tbl` - a list of special tables used to create a summary (not intended for the users).
- `se` - a numeric vector of standard errors of the estimates.
- `p_value` - a numeric vector of the p-values of the tests on the significance of the estimated parameters where the null hypothesis is that corresponding parameter is zero.
- `logLik` - the value of log-likelihood function at `par`.
- `other` - a list of additional variables not intended for the users.
- `cluster` - the same as `cluster` input argument.

It is highly recommended to get the estimates of the parameters via `coef.msel` function.

References

W. K. Newey (2009). Two-step series estimation of sample selection models. The Econometrics Journal, vol. 12(1), pages 217-229.

E. Kossova, B. Potanin (2018). Heckman method and switching regression model multivariate generalization. Applied Econometrics, vol. 50, pages 114-143.

E. Kossova, L. Kupriianova, B. Potanin (2020). Parametric and semiparametric multivariate sample selection models estimators' accuracy: Comparative analysis on simulated data. Applied Econometrics, vol. 57, pages 119-139.

E. Kossova, B. Potanin (2022). Estimation of Gaussian multinomial endogenous switching model. Applied Econometrics, vol. 67, pages 121-143.

E. Meijer, T. Wansbeek (2007). The sample selection model from a method of moments perspective. Econometric Reviews, vol. 26(1), pages 25-51.

Examples

```
# ---------------------------------------
# CPS data example
# ---------------------------------------

# Set seed for reproducibility
set.seed(123)

# Load data
data(cps)

# Prepare the variable for education
cps$educ                    <- NA
cps$educ[cps$basic == 1]    <- 0
cps$educ[cps$bachelor == 1] <- 1
cps$educ[cps$master == 1]   <- 2

# Labor supply (probit) model
f_work <- work ~ age + I(age ^ 2) + bachelor + master + health +
                 slwage + nchild
model1 <- msel(f_work, data = cps)
summary(model1)

# Education choice (ordered probit) model
f_educ <- educ ~ age + I(age ^ 2) + sbachelor + smaster
model2 <- msel(f_educ, data = cps)
summary(model2)

# Education choice (multinomial logit) model
model3 <- msel(formula3 = f_educ, data = cps, type3 = "logit")
summary(model3)

# Education choice (multinomial probit) model
model4 <- msel(formula3 = f_educ, data = cps, type3 = "probit")
summary(model4)
```

```
# Labor supply with endogenous ordinal education
# treatment (recursive or hierarchical ordered probit) model
model5 <- msel(list(f_work, f_educ), data = cps)
summary(model5)

# Sample selection (on employment) Heckman's model
f_lwage                  <- lwage ~ age + I(age ^ 2) +
                                    bachelor + master + health
model6                   <- msel(f_work, f_lwage, data = cps)
summary(model6)

# Ordinal endogenous education treatment with non-random
# sample selection into employment
model7 <- msel(list(f_work, f_educ), f_lwage, data = cps)
summary(model7)

# Ordinal endogenous switching on education model with
# non-random selection into employment
groups   <- cbind(c(1, 1, 1, 0, 0, 0),
                  c(0, 1, 2, 0, 1, 2))
groups2  <- matrix(c(0, 1, 2, -1, -1, -1), ncol = 1)
f_lwage2 <- lwage ~ age + I(age ^ 2) + health
model8   <- msel(list(f_work, f_educ), f_lwage2,
                     groups = groups, groups2 = groups2,
                     data   = cps)
summary(model8)

# Multinomial endogenous switching on education model with
# non-random selection into employment
groups  <- matrix(c(1, 1, 1, 0, 0, 0), ncol = 1)
groups2 <- matrix(c(0, 1, 2, -1, -1, -1), ncol = 1)
groups3 <- c(0, 1, 2, 0, 1, 2)
model9  <- msel(f_work, f_lwage2, f_educ,
                groups    = groups,  groups2 = groups2,
                groups3   = groups3, data    = cps,
                estimator = "2step")
summary(model9)


# ---------------------------------------
# Simulated data example 1
# Ordered probit and other univariate
# ordered choice models
# ---------------------------------------

# ---
# Step 1
# Simulation of the data
# ---

# Load required package
library("mnorm")
```

```
# Set seed for reproducibility
set.seed(123)

# The number of observations
n <- 1000

# Regressors (covariates)
w1 <- runif(n = n, min = -1, max = 1)
w2 <- runif(n = n, min = -1, max = 1)

# Random errors
u <- rnorm(n = n, mean = 0, sd = 1)

# Coefficients
gamma <- c(-1, 2)

# Linear predictor (index)
li <- gamma[1] * w1 + gamma[2] * w2

# Latent variable
z_star <- li + u

# Cuts
cuts <- c(-1, 0.5, 2)

# Observable ordinal outcome
z                                          <- rep(0, n)
z[(z_star > cuts[1]) & (z_star <= cuts[2])] <- 1
z[(z_star > cuts[2]) & (z_star <= cuts[3])] <- 2
z[z_star > cuts[3]]                        <- 3
table(z)

# Data
data <- data.frame(w1 = w1, w2 = w2, z = z)

# ---
# Step 2
# Estimation of the parameters
# ---

# Estimation
model <- msel(z ~ w1 + w2, data = data)
summary(model)

# Compare the estimates and true values of the parameters
  # regression coefficients
gamma_est <- coef(model, type = "coef", eq = 1)
cbind(true = gamma, estimate = gamma_est)
  # cuts
cuts_est <- coef(model, type = "cuts", eq = 1)
cbind(true = cuts, estimate = cuts_est)

# ---
```

```
# Step 3
# Estimation of the probabilities and marginal effects
# ---

# Predict conditional probability of the dependent variable
# equals 2 for every observation in a sample.
# P(z = 2 | w)
prob <- predict(model, group = 2, type = "prob")
head(prob)

# Calculate mean marginal effect of w2 on P(z = 1 | w)
mean(predict(model, group = 1, type = "prob", me = "w2"))

# Calculate probabilities and marginal effects
# for manually provided observations.
  # new data
newdata <- data.frame(z = c(1, 1),
                      w1 = c(0.5, 0.2),
                      w2 = c(-0.3, 0.8))
  # probability P(z = 2 | w)
predict(model, group = 2, type = "prob", newdata = newdata)
  # linear predictor (index)
predict(model, type = "li", newdata = newdata)
  # marginal effect of w1 on P(z = 2 | w)
predict(model, group = 2, type = "prob", newdata = newdata, me = "w1")
  # marginal effect of w1 and w2 on P(z = 3 | w)
predict(model, group   = 3,       type = "prob",
               newdata = newdata, me   = c("w1", "w2"))
  # marginal effect of w2 on the linear predictor (index)
predict(model, group = 2, type = "li", newdata = newdata, me = "w2")
  # discrete marginal effect:
  # P(z = 2 | w1 = 0.5, w2) - P(z = 2 | w1 = 0.2, w2)
predict(model, group = 2,    type = "prob", newdata = newdata,
               me    = "w2", eps  = c(0.2, 0.5))

# ---
# Step 4
# Ordered logit model
# ---

# Estimate ordered logit model with a unit variance
# that is just a matter of reparametrization i.e.,
# does not affect signs and significance of the coefficients
# and does not affect at all the marginal effects
logit <- msel(z ~ w1 + w2, data = data, marginal = list("logistic" = 0))
summary(logit)

# Compare ordered probit and ordered logit models
# using Akaike and Bayesian information criteria
  # AIC
c(probit = AIC(model), logit = AIC(logit))
  # BIC
c(probit = BIC(model), logit = BIC(logit))
```

```
# Estimate some probabilities and marginal effects
  # probability P(z = 1 | w)
predict(logit, group = 1, type = "prob", newdata = newdata)
  # marginal effect of w2 on P(z = 1 | w)
predict(logit, group = 1, type = "prob", newdata = newdata, me = "w2")

# ---
# Step 5
# Semiparametric ordered choice model with
# Gallant and Nychka distribution
# ---

# Estimate semiparametric model
pgn <- msel(z ~ w1 + w2, data = data, marginal = list("PGN" = 2))
summary(pgn)

# Estimate some probabilities and marginal effects
  # probability P(z = 3 | w)
predict(pgn, group = 3, type = "prob", newdata = newdata)
  # marginal effect of w2 on P(z = 3 | w)
predict(pgn, group = 3, type = "prob", newdata = newdata, me = "w2")

# Test the normality assumption via the likelihood ratio test
summary(lrtest_msel(model, pgn))

# Test the normality assumption via the Wald test
test_fn <- function(object)
{
  marginal_par <- coef(object, type = "marginal", eq = 1)
  return(marginal_par)
}
test_result <- test_msel(object = pgn, test = "wald", fn = test_fn)
summary(test_result)



# ----------------------------------------
# Simulated data example 2
# Heteroscedastic ordered probit model
# ----------------------------------------

# Load required package
library("mnorm")

# ---
# Step 1
# Simulation of the data
# ---

# Set seed for reproducibility
set.seed(123)
```

```
# The number of observations
n <- 1000

# Regressors (covariates)
w1 <- runif(n = n, min = -1, max = 1)
w2 <- runif(n = n, min = -1, max = 1)
w3 <- runif(n = n, min = -1, max = 1)

# Random errors
u <- rnorm(n, mean = 0, sd = 1)

# Coefficients of the mean equation
gamma <- c(-1, 2)

# Coefficients of the variance equation
gamma_het <- c(0.5, -1)

# Linear predictor (index) of the mean equation
li <- gamma[1] * w1 + gamma[2] * w2

# Linear predictor (index) of the variance equation
li_het <- gamma_het[1] * w2 + gamma_het[2] * w3

# Heteroscedastic standard deviation
# i.e., value of the variance equation
sd_het <- exp(li_het)

# Latent variable
z_star <- li + u * sd_het

# Cuts
cuts <- c(-1, 0.5, 2)

# Observable ordinal outcome
z                                          <- rep(0, n)
z[(z_star > cuts[1]) & (z_star <= cuts[2])] <- 1
z[(z_star > cuts[2]) & (z_star <= cuts[3])] <- 2
z[z_star > cuts[3]]                        <- 3
table(z)

# Data
data <- data.frame(w1 = w1, w2 = w2, w3 = w3, z = z)

# ---
# Step 2
# Estimation of the parameters
# ---

# Estimation
model <- msel(z ~ w1 + w2 | w2 + w3,
                   data = data)
summary(model)
```

```
# Compare the estimates and true values of the parameters
  # regression coefficients of the mean equation
gamma_est <- coef(model, type = "coef", eq = 1)
cbind(true = gamma, estimate = gamma_est)
  # regression coefficients of the variance equation
gamma_het_est <- coef(model, type = "coef_var", eq = 1)
cbind(true = gamma_het, estimate = gamma_het_est)
  # cuts
cuts_est <- coef(model, type = "cuts", eq = 1)
cbind(true = cuts, estimate = cuts_est)

# Likelihood-ratio test for the homoscedasticity
model0 <- msel(z ~ w1 + w2, data = data)
summary(lrtest_msel(model, model0))

# Wald test for the homoscedasticity
test_fn <- function(object)
{
  val <- coef(object, type = "coef_var", eq = 1)
  return(val)
}
test_result <- test_msel(model, test = "wald", fn = test_fn)
summary(test_result)

# ---
# Step 3
# Estimation of the probabilities and marginal effects
# ---

# Predict probability that the dependent variable
# equals 2 for every observation in a sample
# P(z = 2 | w)
prob <- predict(model, group = 2, type = "prob")
head(prob)

# Calculate mean marginal effect of w2 on P(z = 1 | w)
mean(predict(model, group = 1, type = "prob", me = "w2"))

# Estimate conditional probabilities, linear predictors (indexes) and
# heteroscedastic standard deviations for manually
# provided observations.
  # new data
newdata <- data.frame(z = c(1, 1),
                      w1 = c(0.5, 0.2),
                      w2 = c(-0.3, 0.8),
                      w3 = c(0.6, 0.1))
  # probability P(z = 2 | w)
predict(model, group = 2, type = "prob", newdata = newdata)
  # linear predictor (index)
predict(model, type = "li", newdata = newdata, group = 0)
  # standard deviation
predict(model, type = "sd", newdata = newdata, group = 0)
  # marginal effect of w3 on P(z = 3 | w)
```

```
predict(model, group = 3, type = "prob", newdata = newdata, me = "w3")
  # marginal effect of w2 on the standard error
predict(model, group = 2, type = "sd", newdata = newdata, me = "w2")
  # discrete marginal effect:
  # P(Z = 2 | w1 = 0.5, w2) - P(Z = 2 | w1 = 0.2, w2)
predict(model, group = 2, type = "prob", newdata = newdata,
        me = "w2", eps = c(0.2, 0.5))



# ----------------------------------------
# Simulated data example 3
# Bivariate ordered probit model with
# heteroscedastic second equation
# ----------------------------------------

# Load required package
library("mnorm")

# ---
# Step 1
# Simulation of the data
# ---

# Set seed for reproducibility
set.seed(123)

# The number of observations
n <- 1000

# Regressors (covariates)
w1 <- runif(n = n, min = -1, max = 1)
w2 <- runif(n = n, min = -1, max = 1)
w3 <- runif(n = n, min = -1, max = 1)
w4 <- runif(n = n, min = -1, max = 1)

# Covariance matrix of random errors
rho <- 0.5
sigma <- matrix(c(1,   rho,
                  rho, 1),
                nrow = 2)

# Random errors
u <- mnorm::rmnorm(n = n, mean = c(0, 0), sigma = sigma)

# Coefficients
gamma1 <- c(-1, 2)
gamma2 <- c(1, 1.5)

# Coefficients of the variance equation
gamma2_het <- c(0.5, -1)

# Linear predictors (indexes)
```

```
li1 <- gamma1[1] * w1 + gamma1[2] * w2
li2 <- gamma2[1] * w2 + gamma2[2] * w3

# Linear predictor (index) of the variance equation
li2_het <- gamma2_het[1] * w2 + gamma2_het[2] * w4

# Heteroscedastic standard deviation
# i.e., the value of the variance equation
sd2_het <- exp(li2_het)

# Latent variables
z1_star <- li1 + u[, 1]
z2_star <- li2 + u[, 2] * sd2_het

# Cuts
cuts1 <- c(-1, 0.5, 2)
cuts2 <- c(-2, 0)

# Observable ordinal outcome
  # first outcome
z1                                              <- rep(0, n)
z1[(z1_star > cuts1[1]) & (z1_star <= cuts1[2])] <- 1
z1[(z1_star > cuts1[2]) & (z1_star <= cuts1[3])] <- 2
z1[z1_star > cuts1[3]]                          <- 3
  # second outcome
z2                                              <- rep(0, n)
z2[(z2_star > cuts2[1]) & (z2_star <= cuts2[2])] <- 1
z2[z2_star > cuts2[2]]                          <- 2
  # distribution
table(z1, z2)

# Data
data <- data.frame(w1 = w1, w2 = w2, w3 = w3,
                   w4 = w4, z1 = z1, z2 = z2)

# ---
# Step 2
# Estimation of the parameters
# ---

# Estimation
model <- msel(list(z1 ~ w1 + w2,
                   z2 ~ w2 + w3 | w2 + w4),
              data = data)
summary(model)

# Compare the estimates and true values of the parameters
  # regression coefficients of the first equation
gamma1_est <- coef(model, type = "coef", eq = 1)
cbind(true = gamma1, estimate = gamma1_est)
  # regression coefficients of the second equation
gamma2_est <- coef(model, type = "coef", eq = 2)
cbind(true = gamma2, estimate = gamma2_est)
```

```
  # cuts of the first equation
cuts1_est <- coef(model, type = "cuts", eq = 1)
cbind(true = cuts1, estimate = cuts1_est)
  # cuts of the second equation
cuts2_est <- coef(model, type = "cuts", eq = 2)
cbind(true = cuts2, estimate = cuts2_est)
  # correlation coefficients
rho_est <- coef(model, type = "cov1", eq = c(1, 2))
cbind(true = rho, estimate = rho_est)
  # regression coefficients of the variance equation
gamma2_het_est <- coef(model, type = "coef_var", eq = 2)
cbind(true = gamma2_het, estimate = gamma2_het_est)

# ---
# Step 3
# Estimation of the probabilities and linear predictors (indexes)
# ---

# Predict probability P(z1 = 2, z2 = 0 | w)
prob <- predict(model, group = c(2, 0), type = "prob")
head(prob)

# Calculate mean marginal effect of w2 on:
  # P(z1 = 1 | w)
mean(predict(model, group = c(1, -1), type = "prob", me = "w2"))
  # P(z1 = 1, z2 = 0 | w)
mean(predict(model, group = c(1, 0), type = "prob", me = "w2"))

# Calculate conditional probabilities and linear predictors (indexes)
# for the manually provided observations.
  # new data
newdata <- data.frame(z1 = c(1, 1),
                      z2 = c(1, 1),
                      w1 = c(0.5, 0.2),
                      w2 = c(-0.3, 0.8),
                      w3 = c(0.6, 0.1),
                      w4 = c(0.3, -0.5))
  # probability P(z1 = 2, z2 = 0 | w)
predict(model, group = c(2, 0), type = "prob", newdata = newdata)
  # linear predictor (index)
predict(model, type = "li", newdata = newdata, group = c(0, 0))
  # marginal probability P(z2 = 1 | w)
predict(model, group = c(-1, 1), type = "prob", newdata = newdata)
  # marginal probability P(z1 = 3 | w)
predict(model, group = c(3, -1), type = "prob", newdata = newdata)
  # conditional probability P(z1 = 2 | z2 = 0, w)
predict(model, group = c(2, 0), given_ind = 2,
        type = "prob", newdata = newdata)
  # conditional probability P(z2 = 1 | z1 = 3, w)
predict(model, group = c(3, 1), given_ind = 1,
        type = "prob", newdata = newdata)
  # marginal effect of w4 on P(Z2 = 2 | w)
predict(model, group = c(-1, 2),
```

```
        type = "prob", newdata = newdata, me = "w4")
  # marginal effect of w4 on P(z1 = 3, Z2 = 2 | w)
predict(model, group = c(3, 2),
        type = "prob", newdata = newdata, me = "w4")
  # marginal effect of w4 on P(z1 = 3 | z2 = 2, w)
predict(model, group = c(3, 2), given_ind = 2,
        type = "prob", newdata = newdata, me = "w4")

# ---
# Step 4
# Replication under the non-random sample selection
# ---

# Suppose that z2 is unobservable when z1 = 1 or z1 = 3
z2[(z1 == 1) | (z1 == 3)] <- -1
data$z2 <- z2

# Replicate the estimation procedure
model <- msel(list(z1 ~ w1 + w2,
                   z2 ~ w2 + w3 | w2 + w4),
              cov_type = "gop", data = data)
summary(model)

# Compare the estimates and true values of the parameters
  # regression coefficients of the first equation
gamma1_est <- coef(model, type = "coef", eq = 1)
cbind(true = gamma1, estimate = gamma1_est)
  # regression coefficients of the second equation
gamma2_est <- coef(model, type = "coef", eq = 2)
cbind(true = gamma2, estimate = gamma2_est)
  # cuts of the first equation
cuts1_est <- coef(model, type = "cuts", eq = 1)
cbind(true = cuts1, estimate = cuts1_est)
  # cuts of the second equation
cuts2_est <- coef(model, type = "cuts", eq = 2)
cbind(true = cuts2, estimate = cuts2_est)
  # correlation coefficient
rho_est <- coef(model, type = "cov1", eq = c(1, 2))
cbind(true = rho, estimate = rho_est)
  # regression coefficients of the variance equation
gamma2_het_est <- coef(model, type = "coef_var", eq = 2)
cbind(true = gamma2_het, estimate = gamma2_het_est)

# ---
# Step 5
# Semiparametric model with marginal logistic and PGN distributions
# ---

# Estimate the model
model <- msel(list(z1 ~ w1 + w2,
                   z2 ~ w2 + w3 | w2 + w4),
              data = data, marginal = list(PGN = 3, logistic = NULL))
summary(model)
```

```
# ---------------------------------------
# Simulated data example 4
# Heckman model with ordinal
# selection mechanism
# ---------------------------------------

# Load required package
library("mnorm")

# ---
# Step 1
# Simulation of the data
# ---

# Set seed for reproducibility
set.seed(123)

# The number of observations
n <- 1000

# Regressors (covariates)
w1 <- runif(n = n, min = -1, max = 1)
w2 <- runif(n = n, min = -1, max = 1)
w3 <- runif(n = n, min = -1, max = 1)

# Random errors
rho    <- 0.5
var.y  <- 0.3
sd.y   <- sqrt(var.y)
sigma  <- matrix(c(1,          rho * sd.y,
                   rho * sd.y, var.y),
                 nrow = 2)
errors <- mnorm::rmnorm(n = n, mean = c(0, 0), sigma = sigma)
u      <- errors[, 1]
eps    <- errors[, 2]

# Coefficients
gamma <- c(-1, 2)
beta  <- c(1, -1, 1)

# Linear predictor (index)
li   <- gamma[1] * w1 + gamma[2] * w2
li.y <- beta[1] + beta[2] * w1 + beta[3] * w3

# Latent variable
z_star <- li + u
y_star <- li.y + eps

# Cuts
cuts <- c(-1, 0.5, 2)
```

```
# Observable ordered outcome
z                                          <- rep(0, n)
z[(z_star > cuts[1]) & (z_star <= cuts[2])] <- 1
z[(z_star > cuts[2]) & (z_star <= cuts[3])] <- 2
z[z_star > cuts[3]]                        <- 3
table(z)

# Observable continuous outcome such
# that outcome 'y' is observable only
# when 'z > 1' and unobservable otherwise
# i.e., when 'z <= 1' we code 'y' as 'NA'
y         <- y_star
y[z <= 1] <- NA

# Data
data <- data.frame(w1 = w1, w2 = w2, w3 = w3,
                   z = z, y = y)

# ---
# Step 2
# Estimation of the parameters
# ---

# Estimation
model <- msel(z ~ w1 + w2, y ~ w1 + w3, data = data)
summary(model)

# Compare estimates and true values of the parameters
  # regression coefficients of the ordinal equation
gamma_est <- coef(model, type = "coef", eq = 1)
cbind(true = gamma, estimate = gamma_est)
  # cuts
cuts_est <- coef(model, type = "cuts", eq = 1)
cbind(true = cuts, estimate = cuts_est)
  # regression coefficients of the continuous equation
beta_est <- coef(model, type = "coef2", eq2 = 1, regime = 0)
cbind(true = beta, estimate = beta_est)
  # variance
var.y_est <- coef(model, type = "var", eq2 = 1, regime = 0)
cbind(true = var.y, estimate = var.y_est)
  # covariance
cov_est <- coef(model, type = "cov12", eq = 1, eq2 = 1)
cbind(true = rho * sd.y, estimate = cov_est)

# ---
# Step 3
# Estimation of the expectations and marginal effects
# ---

# New data
newdata <- data.frame(z = 1,
                      y = 1,
```

```
                    w1 = 0.1,
                    w2 = 0.2,
                    w3 = 0.3)

# Predict the unconditional expectation of the continuous outcome
# E(y | w)
predict(model, group = -1, group2 = 0, newdata = newdata)

# Predict the conditional expectations of the continuous outcome
  # E(y | z = 2, w)
predict(model, group = 2, group2 = 0, newdata = newdata)
  # E(y | z = 0, w)
predict(model, group = 0, group2 = 0, newdata = newdata)

# ---
# Step 4
# The classic Heckman two-step estimation procedure
# ---

# Estimate the model by using the two-step estimator
model_ts <- msel(z ~ w1 + w2, y ~ w1 + w3,
                 data = data, estimator = "2step")
summary(model_ts)

# Check the accuracy of the estimates
tbl <- cbind(true    = beta,
             ml      = model$coef2[[1]][1, ],
             twostep = model_ts$coef2[[1]][1, -4])
print(tbl)

# ---
# Step 5
# Semiparametric estimation procedure
# ---

# Estimate the model using Lee's method
# assuming logistic distribution of the
# random errors of the selection equation
model_Lee <- msel(z ~ w1 + w2,
                  y ~ w1 + w3,
                  data = data, estimator = "2step",
                  marginal = list(logistic = NULL))
summary(model_Lee)

# One-step estimation is available as well as
# more complex marginal distributions.
# Consider random errors in the selection equation
# following a PGN distribution with three parameters.
model_sp <- msel(z ~ w1 + w2,
                 y ~ w1 + w3,
                 data = data, marginal = list(PGN = 3))
summary(model_sp)
```

```
# To additionally relax the normality assumption of
# the random error of the continuous equation, it is possible
# to use Newey's two-step procedure.
model_Newey <- msel(z ~ w1 + w2,
                    y ~ w1 + w3,
                    data      = data,    marginal = list(logistic = 0),
                    estimator = "2step", degrees  = 2)
summary(model_Newey)



# ----------------------------------------
# Simulated data example 5
# Endogenous switching model with
# heteroscedastic ordered selection
# mechanism
# ----------------------------------------

# Load required package
library("mnorm")

# ---
# Step 1
# Simulation of the data
# ---

# Set seed for reproducibility
set.seed(123)

# The number of observations
n <- 1000

# Regressors (covariates)
w1 <- runif(n = n, min = -1, max = 1)
w2 <- runif(n = n, min = -1, max = 1)
w3 <- runif(n = n, min = -1, max = 1)

# Random errors
rho_0    <- -0.8
rho_1    <- -0.7
var2_0   <- 0.9
var2_1   <- 1
sd_y_0   <- sqrt(var2_0)
sd_y_1   <- sqrt(var2_1)
cor_y_01 <- 0.7
cov2_01  <- sd_y_0 * sd_y_1 * cor_y_01
cov2_z_0 <- rho_0 * sd_y_0
cov2_z_1 <- rho_1 * sd_y_1
sigma    <- matrix(c(1,        cov2_z_0, cov2_z_1,
                     cov2_z_0, var2_0,   cov2_01,
                     cov2_z_1, cov2_01,  var2_1),
                   nrow = 3)
errors   <- mnorm::rmnorm(n = n, mean = c(0, 0, 0), sigma = sigma)
```

```
u        <- errors[, 1]
eps_0    <- errors[, 2]
eps_1    <- errors[, 3]

# Coefficients
gamma     <- c(-1, 2)
gamma_het <- c(0.5, -1)
beta_0    <- c(1, -1, 1)
beta_1    <- c(2, -1.5, 0.5)

# Linear predictor (index) of the ordinal equation
  # mean
li <- gamma[1] * w1 + gamma[2] * w2
  # variance
li_het <- gamma_het[1] * w2 + gamma_het[2] * w3

# Linear predictor (index) of the continuous equation
  # regime 0
li_y_0 <- beta_0[1] + beta_0[2] * w1 + beta_0[3] * w3
  # regime 1
li_y_1 <- beta_1[1] + beta_1[2] * w1 + beta_1[3] * w3

# Latent variable
z_star   <- li + u * exp(li_het)
y_0_star <- li_y_0 + eps_0
y_1_star <- li_y_1 + eps_1

# Cuts
cuts <- c(-1, 0.5, 2)

# Observable ordinal outcome
z                                          <- rep(0, n)
z[(z_star > cuts[1]) & (z_star <= cuts[2])] <- 1
z[(z_star > cuts[2]) & (z_star <= cuts[3])] <- 2
z[z_star > cuts[3]]                        <- 3
table(z)

# Observable continuous outcome such that y is
# observable in regime 1 when 'z = 1',
# observable in regime 0 when 'z <= 1',
# unobservable           when 'z = 0'
y         <- rep(NA, n)
y[z == 0] <- NA
y[z == 1] <- y_0_star[z == 1]
y[z > 1]  <- y_1_star[z > 1]

# Data
data <- data.frame(w1 = w1, w2 = w2, w3 = w3,
                   z  = z,  y  = y)

# ---
# Step 2
# Estimation of the parameters
```

```
# ---

# Assign groups
groups <- matrix(0:3, ncol = 1)
groups2 <- matrix(c(-1, 0, 1, 1), ncol = 1)

# Estimation
model <- msel(list(z ~ w1 + w2 | w2 + w3),
                  list(y ~ w1 + w3),
                  groups = groups, groups2 = groups2,
                  data = data)
summary(model)

# Compare the estimates and true values of the parameters
  # regression coefficients of the ordinal equation
gamma_est     <- coef(model, type = "coef",     eq = 1)
gamma_het_est <- coef(model, type = "coef_var", eq = 1)
cbind(true = gamma,     estimate = gamma_est)
cbind(true = gamma_het, estimate = gamma_het_est)
  # cuts
cuts_est <- coef(model, type = "cuts", eq = 1)
cbind(true = cuts, estimate = cuts_est)
  # regression coefficients of the continuous equation
beta_0_test <- coef(model, type = "coef2", eq2 = 1, regime = 0)
beta_1_test <- coef(model, type = "coef2", eq2 = 1, regime = 1)
cbind(true = beta_0, estimate = beta_0_test)
cbind(true = beta_1, estimate = beta_1_test)
  # variances
var2_0_est <- coef(model, type = "var", eq2 = 1, regime = 0)
var2_1_est <- coef(model, type = "var", eq2 = 1, regime = 1)
cbind(true = c(var2_0, var2_1), estimate = c(var2_0_est, var2_1_est))
  # covariances between the random errors
cov2_z_0_est <- coef(model, type = "cov12", eq = 1, eq2 = 1, regime = 0)
cov2_z_1_est <- coef(model, type = "cov12", eq = 1, eq2 = 1, regime = 1)
cbind(true     = c(cov2_z_0,     cov2_z_1),
      estimate = c(cov2_z_0_est, cov2_z_1_est))

# ---
# Step 3
# Estimation of the expectations and marginal effects
# ---

# New data
newdata <- data.frame(z  = 1,   y  = 1,
                      w1 = 0.1, w2 = 0.2, w3 = 0.3)

# Predict the unconditional expectation of the
# continuous outcome E(yr | w)
  # regime 0
predict(model, group = -1, group2 = 0, newdata = newdata)
  # regime 1
predict(model, group = -1, group2 = 1, newdata = newdata)
```

```
# Predict the conditional expectations of the continuous outcome
# given condition 'z == 0' for regime 1 i.e., E(y1 | z = 0, w)
predict(model, group = 0, group2 = 1, newdata = newdata)



# ----------------------------------------
# Simulated data example 6
# Endogenous switching model with
# multivariate heteroscedastic ordered
# selection mechanism
# ----------------------------------------

# Load required package
library("mnorm")

# ---
# Step 1
# Simulation of the data
# ---

# Set seed for reproducibility
set.seed(123)

# The number of observations
n <- 1000

# Regressors (covariates)
w1 <- runif(n = n, min = -1, max = 1)
w2 <- runif(n = n, min = -1, max = 1)
w3 <- runif(n = n, min = -1, max = 1)
w4 <- runif(n = n, min = -1, max = 1)

# Random errors
rho_z1_z2 <- 0.5
rho_y0_z1 <- 0.6
rho_y0_z2 <- 0.7
rho_y1_z1 <- 0.65
rho_y1_z2 <- 0.75
var20     <- 0.9
var21     <- 1
sd_y0     <- sqrt(var20)
sd_y1     <- sqrt(var21)
cor_y01   <- 0.7
cov201    <- sd_y0 * sd_y1 * cor_y01
cov20_z1  <- rho_y0_z1 * sd_y0
cov21_z1  <- rho_y1_z1 * sd_y1
cov20_z2  <- rho_y0_z2 * sd_y0
cov21_z2  <- rho_y1_z2 * sd_y1
sigma     <- matrix(c(1,         rho_z1_z2,  cov20_z1,  cov21_z1,
                      rho_z1_z2, 1,          cov20_z2,  cov21_z2,
                      cov20_z1,  cov20_z2,   var20,     cov201,
                      cov21_z1,  cov21_z2,   cov201,    var21),
```

```
                    nrow = 4)
errors    <- mnorm::rmnorm(n = n, mean = c(0, 0, 0, 0), sigma = sigma)
u1        <- errors[, 1]
u2        <- errors[, 2]
eps0      <- errors[, 3]
eps1      <- errors[, 4]

# Coefficients
gamma1     <- c(-1, 2)
gamma1_het <- c(0.5, -1)
gamma2     <- c(1, 1)
beta0      <- c(1, -1, 1, -1.2)
beta1      <- c(2, -1.5, 0.5, 1.2)

# Linear index (predictor) of the ordinal equation
  # mean
li1 <- gamma1[1] * w1 + gamma1[2] * w2
li2 <- gamma2[1] * w1 + gamma2[2] * w3
  # variance
li1_het <- gamma1_het[1] * w2 + gamma1_het[2] * w3

# Linear predictor (index) of the continuous equation
  # regime 0
li_y0 <- beta0[1] + beta0[2] * w1 + beta0[3] * w3 + beta0[4] * w4
  # regime 1
li_y1 <- beta1[1] + beta1[2] * w1 + beta1[3] * w3 + beta1[4] * w4

# Latent variables
z1_star <- li1 + u1 * exp(li1_het)
z2_star <- li2 + u2
y0_star <- li_y0 + eps0
y1_star <- li_y1 + eps1

# Cuts
cuts1 <- c(-1, 1)
cuts2 <- c(0)

# Observable ordered outcome
  # first
z1                                              <- rep(0, n)
z1[(z1_star > cuts1[1]) & (z1_star <= cuts1[2])] <- 1
z1[z1_star > cuts1[2]]                          <- 2
  # second
z2                     <- rep(0, n)
z2[z2_star > cuts2[1]] <- 1
table(z1, z2)

# Observable continuous outcome such
# that outcome 'y' is
# in regime 0  when 'z1 == 1',
# in regime 1  when 'z1 == 0' or 'z1 == 2',
# unobservable when 'z2 == 0'
y          <- rep(NA, n)
```

```
y[z1 == 1] <- y0_star[z1 == 1]
y[z1 != 1] <- y1_star[z1 != 1]
y[z2 == 0] <- NA

# Data
data <- data.frame(w1 = w1, w2 = w2, w3 = w3, w4 = w4,
                   z1 = z1, z2 = z2, y  = y)

# ---
# Step 2
# Estimation of the parameters
# ---

# Assign groups
groups <- matrix(c(0, 0,
                   0, 1,
                   1, 0,
                   1, 1,
                   2, 0,
                   2, 1),
                 byrow = TRUE, ncol = 2)
groups2 <- matrix(c(-1, 1, -1, 0, -1, 1), ncol = 1)

# Estimation
model <- msel(list(z1 ~ w1 + w2 | w2 + w3,
                   z2 ~ w1 + w3),
              y ~ w1 + w3 + w4,
              groups = groups, groups2 = groups2,
              data   = data)
summary(model)

# Compare estimates and true values of the parameters
  # regression coefficients of the first ordered equation
gamma1_est      <- coef(model, type = "coef",     eq = 1)
gamma1_het_est <- coef(model, type = "coef_var", eq = 1)
cbind(true = gamma1,     estimate = gamma1_est)
cbind(true = gamma1_het, estimate = gamma1_het_est)
  # regression coefficients of the second ordered equation
gamma2_est <- coef(model, type = "coef", eq = 2)
cbind(true = gamma2, estimate = gamma2_est)
  # cuts
cuts1_est <- coef(model, type = "cuts", eq = 1)
cuts2_est <- coef(model, type = "cuts", eq = 2)
cbind(true = cuts1, estimate = cuts1_est)
cbind(true = cuts2, estimate = cuts2_est)
  # regression coefficients of the continuous equation
beta0_est <- coef(model, type = "coef2", eq2 = 1, regime = 0)
beta1_est <- coef(model, type = "coef2", eq2 = 1, regime = 1)
cbind(true = beta0, estimate = beta0_est)
cbind(true = beta1, estimate = beta1_est)
  # variances
var20_est <- coef(model, type = "var", eq2 = 1, regime = 0)
var21_est <- coef(model, type = "var", eq2 = 1, regime = 1)
```

```
cbind(true = c(var20, var21), estimate = c(var20_est, var21_est))
  # covariances
cov_y0_z1_est <- coef(model, type = "cov12", eq = 1, eq2 = 1, regime = 0)
cov_y0_z2_est <- coef(model, type = "cov12", eq = 2, eq2 = 1, regime = 0)
cov_y1_z1_est <- coef(model, type = "cov12", eq = 1, eq2 = 1, regime = 1)
cov_y1_z2_est <- coef(model, type = "cov12", eq = 2, eq2 = 1, regime = 1)
cbind(true     = c(cov20_z1, cov20_z2),
      estimate = c(cov_y0_z1_est, cov_y0_z2_est))
cbind(true     = c(cov21_z1, cov21_z2),
      estimate = c(cov_y1_z1_est,  cov_y1_z2_est))

# ---
# Step 3
# Estimation of the expectations and marginal effects
# ---

# New data
newdata <- data.frame(z1 = 1,   z2 = 1,   y  = 1,
                      w1 = 0.1, w2 = 0.2, w3 = 0.3, w4 = 0.4)

# Predict the unconditional expectation of the continuous outcome
  # regime 0
predict(model, group = c(-1, -1), group2 = 0, newdata = newdata)
  # regime 1
predict(model, group = c(-1, -1), group2 = 1, newdata = newdata)

# Predict the conditional expectations of the continuous outcome
# E(y1 | z1 = 2, z2 = 1, w)
predict(model, group = c(2, 1), group2 = 1, newdata = newdata)

# Marginal effect of w3 on E(y1 | z1 = 2, z2 = 1, w)
predict(model, group = c(2, 1), group2 = 1, newdata = newdata, me = "w3")

# ---
# Step 4
# Two-step estimation procedure
# ---

# For comparison reasons, let's estimate the model
# via the least squares
model.ls.0 <- lm(y ~ w1 + w3 + w4,
                 data = data[!is.na(data$y) & (data$z1 == 1), ])
model.ls.1 <- lm(y ~ w1 + w3 + w4,
                 data = data[!is.na(data$y) & (data$z1 != 1), ])


# Apply the two-step procedure
model_ts <-  msel(list(z1 ~ w1 + w2 | w2 + w3,
                       z2 ~ w1 + w3),
                  y ~ w1 + w3 + w4,
                  groups    = groups,  groups2 = groups2,
                  estimator = "2step", data    = data)
summary(model_ts)
```

```
# Use the two-step procedure with logistic marginal distributions
# that is a multivariate generalization of Lee's method
model_Lee <-  msel(list(z1 ~ w1 + w2 | w2 + w3,
                        z2 ~ w1 + w3),
                   y ~ w1 + w3 + w4,
                   marginal = list(logistic = NULL, logistic = NULL),
                   groups    = groups,  groups2 = groups2,
                   estimator = "2step", data    = data)

# Apply Newey's method
model_Newey <-  msel(list(z1 ~ w1 + w2 | w2 + w3,
                          z2 ~ w1 + w3),
                     y ~ w1 + w3 + w4,
                     marginal = list(logistic = NULL, logistic = NULL),
                     degrees = c(2, 3), groups = groups, groups2 = groups2,
                     estimator = "2step", data = data)

# Compare the accuracy of the methods
  # beta0
tbl0 <- cbind(true    = beta0,
              ls      = coef(model.ls.0),
              ml      = model$coef2[[1]][1, 1:length(beta0)],
              twostep = model_ts$coef2[[1]][1, 1:length(beta0)],
              Lee     = model_Lee$coef2[[1]][1, 1:length(beta0)],
              Newey   = model_Newey$coef2[[1]][1, 1:length(beta0)])
print(tbl0)
  # beta1
tbl1 <- cbind(true    = beta1,
              ls      = coef(model.ls.1),
              ml      = model$coef2[[1]][2, 1:length(beta1)],
              twostep = model_ts$coef2[[1]][2, 1:length(beta1)],
              Lee     = model_Lee$coef2[[1]][2, 1:length(beta1)],
              Newey   = model_Newey$coef2[[1]][2, 1:length(beta1)])
print(tbl1)



# ---------------------------------------
# Simulated data example 7
# Endogenous multivariate switching model
# with multivariate heteroscedastic
# ordered selection mechanism
# ---------------------------------------

# Load required package
library("mnorm")

# ---
# Step 1
# Simulation of the data
# ---
```

```
# Set seed for reproducibility
set.seed(123)

# The number of observations
n <- 1000

# Regressors (covariates)
w1 <- runif(n = n, min = -1, max = 1)
w2 <- runif(n = n, min = -1, max = 1)
w3 <- runif(n = n, min = -1, max = 1)
w4 <- runif(n = n, min = -1, max = 1)
w5 <- runif(n = n, min = -1, max = 1)

# Random errors
var_y0 <- 0.9
var_y1 <- 1
var_g0 <- 1.1
var_g1 <- 1.2
var_g2 <- 1.3
A      <- rWishart(1, 7, diag(7))[, , 1]
B      <- diag(sqrt(c(1, 1, var_y0, var_y1,
                      var_g0, var_g1, var_g2)))
sigma  <-  B  %*% cov2cor(A) %*% B
errors <- mnorm::rmnorm(n = n, mean = rep(0, nrow(sigma)), sigma = sigma)
u1     <- errors[, 1]
u2     <- errors[, 2]
eps0_y <- errors[, 3]
eps1_y <- errors[, 4]
eps0_g <- errors[, 5]
eps1_g <- errors[, 6]
eps2_g <- errors[, 7]

# Coefficients
gamma1     <- c(-1, 2)
gamma1_het <- c(0.5, -1)
gamma2     <- c(1, 1)
beta0_y    <- c(1, -1, 1, -1.2)
beta1_y    <- c(2, -1.5, 0.5, 1.2)
beta0_g    <- c(-1, 1, 1, 1)
beta1_g    <- c(1, -1, 1, 1)
beta2_g    <- c(1, 1, -1, 1)

# Linear predictor (index) of the ordinal equation
  # mean
li1 <- gamma1[1] * w1 + gamma1[2] * w2
li2 <- gamma2[1] * w1 + gamma2[2] * w3
  # variance
li1_het <- gamma1_het[1] * w2 + gamma1_het[2] * w3

# Linear predictor (index) of the first continuous equation
  # regime 0
li_y0 <- beta0_y[1] + beta0_y[2] * w1 + beta0_y[3] * w3 + beta0_y[4] * w4
  # regime 1
```

```
li_y1 <- beta1_y[1] + beta1_y[2] * w1 + beta1_y[3] * w3 + beta1_y[4] * w4

# Linear predictor (index) of the second continuous equation
  # regime 0
li_g0 <- beta0_g[1] + beta0_g[2] * w2 + beta0_g[3] * w3 + beta0_g[4] * w5
  # regime 1
li_g1 <- beta1_g[1] + beta1_g[2] * w2 + beta1_g[3] * w3 + beta1_g[4] * w5
  # regime 2
li_g2 <- beta2_g[1] + beta2_g[2] * w2 + beta2_g[3] * w3 + beta2_g[4] * w5

# Latent variables
z1_star <- li1 + u1 * exp(li1_het)
z2_star <- li2 + u2
y0_star <- li_y0 + eps0_y
y1_star <- li_y1 + eps1_y
g0_star <- li_g0 + eps0_g
g1_star <- li_g1 + eps1_g
g2_star <- li_g2 + eps2_g

# Cuts
cuts1 <- c(-1, 1)
cuts2 <- c(0)

# Observable ordered outcome
  # first
z1                                              <- rep(0, n)
z1[(z1_star > cuts1[1]) & (z1_star <= cuts1[2])] <- 1
z1[z1_star > cuts1[2]]                          <- 2
  # second
z2                     <- rep(0, n)
z2[z2_star > cuts2[1]] <- 1
table(z1, z2)

# Observable continuous outcome such that outcome 'y' is
# in regime 0 when 'z1 == 1',
# in regime 1 when 'z1 == 0' or 'z1 == 2',
# unobservable when 'z2 == 0'
y          <- rep(NA, n)
y[z1 == 1] <- y0_star[z1 == 1]
y[z1 != 1] <- y1_star[z1 != 1]
y[z2 == 0] <- NA

# Observable continuous outcome such that outcome 'g' is
# in regime 0 when 'z1 == z2',
# in regime 1 when 'z1 > z2',
# in regime 2 when 'z1 < z2',
g           <- rep(NA, n)
g[z1 == z2] <- g0_star[z1 == z2]
g[z1 > z2]  <- g1_star[z1 > z2]
g[z1 < z2]  <- g2_star[z1 < z2]

# Data
data <- data.frame(w1 = w1, w2 = w2, w3 = w3, w4 = w4, w5 = w5,
```

```
                  z1 = z1, z2 = z2, y  = y,  g  = g)

# ---
# Step 2
# Estimation of the parameters
# ---

# Assign groups
groups <- matrix(c(0, 0,
                   0, 1,
                   1, 0,
                   1, 1,
                   2, 0,
                   2, 1),
                 byrow = TRUE, ncol = 2)

# Assign groups 2
  # prepare the matrix
groups2 <- matrix(NA, nrow = nrow(groups), ncol = 2)
  # fill the matrix
groups2[groups[, 1] == 1, 1]                        <- 0
groups2[(groups[, 1] == 0) | (groups[, 1] == 2), 1] <- 1
groups2[groups[, 2] == 0, 1]                        <- -1
groups2[groups[, 1] == groups[, 2], 2]              <- 0
groups2[groups[, 1] > groups[, 2], 2]               <- 1
groups2[groups[, 1] < groups[, 2], 2]               <- 2

# The structure of the model
cbind(groups, groups2)

# Estimation
model <- msel(list(z1 ~ w1 + w2 | w2 + w3, z2 ~ w1 + w3),
              list(y ~ w1 + w3 + w4, g ~ w2 + w3 + w5),
              groups = groups, groups2 = groups2, data = data)
summary(model)

# Compare the estimates and true values of the parameters
  # regression coefficients of the first ordered equation
gamma1_est     <- coef(model, type = "coef", eq = 1)
gamma1_het_est <- coef(model, type = "coef_var", eq = 1)
cbind(true = gamma1,     estimate = gamma1_est)
cbind(true = gamma1_het, estimate = gamma1_het_est)
  # regression coefficients of the second ordered equation
gamma2_est <- coef(model, type = "coef", eq = 2)
cbind(true = gamma2, estimate = gamma2_est)
  # cuts
cuts1_est <- coef(model, type = "cuts", eq = 1)
cuts2_est <- coef(model, type = "cuts", eq = 2)
cbind(true = cuts1, estimate = cuts1_est)
cbind(true = cuts2, estimate = cuts2_est)
  # regression coefficients of the first continuous equation
beta0_y_est <- coef(model, type = "coef2", eq2 = 1, regime = 0)
beta1_y_est <- coef(model, type = "coef2", eq2 = 1, regime = 1)
```

```
cbind(true = beta0_y, estimate = beta0_y_est)
cbind(true = beta1_y, estimate = beta1_y_est)
  # regression coefficients of the second continuous equation
beta0_g_est <- coef(model, type = "coef2", eq2 = 2, regime = 0)
beta1_g_est <- coef(model, type = "coef2", eq2 = 2, regime = 1)
beta2_g_est <- coef(model, type = "coef2", eq2 = 2, regime = 2)
cbind(true = beta0_g, estimate = beta0_g_est)
cbind(true = beta1_g, estimate = beta1_g_est)
cbind(true = beta2_g, estimate = beta2_g_est)
  # variances of the first continuous equation
var_y0_est <- coef(model, type = "var", eq2 = 1, regime = 0)
var_y1_est <- coef(model, type = "var", eq2 = 1, regime = 1)
cbind(true = c(var_y0, var_y1), estimate = c(var_y0_est, var_y1_est))
  # variances of the second continuous equation
var_g0_est <- coef(model, type = "var", eq2 = 2, regime = 0)
var_g1_est <- coef(model, type = "var", eq2 = 2, regime = 1)
var_g2_est <- coef(model, type = "var", eq2 = 2, regime = 2)
cbind(true     = c(var_g0, var_g1, var_g2),
      estimate = c(var_g0_est, var_g1_est, var_g2_est))
  # correlation between the ordinal equations
sigma12_est <- coef(model, type = "cov1", eq = c(1, 2))
cbind(true = c(sigma[1, 2]), estimate = sigma12_est)
  # covariances between the continuous and ordinal equations
cbind(true = sigma[1:2, 3], estimate = model$cov2[[1]][1, ])
cbind(true = sigma[1:2, 4], estimate = model$cov2[[1]][2, ])
cbind(true = sigma[1:2, 5], estimate = model$cov2[[2]][1, ])
cbind(true = sigma[1:2, 6], estimate = model$cov2[[2]][2, ])
cbind(true = sigma[1:2, 7], estimate = model$cov2[[2]][3, ])
  # covariances between the continuous equations
sigma2_est <- coef(model, type = "cov2")[[1]]
cbind(true     = c(sigma[4, 7], sigma[3, 5], sigma[4, 6]),
      estimate = sigma2_est)

# ---
# Step 3
# Estimation of the expectations and marginal effects
# ---

# New data
newdata <- data.frame(z1 = 1,   z2 = 1,   y  = 1,   g  = 1,
                      w1 = 0.1, w2 = 0.2, w3 = 0.3, w4 = 0.4, w5 = 0.5)

# Predict unconditional expectation of the dependent variable
  # regime 0 for 'y' and regime 1 for 'g' i.e. E(y0 | w), E(g1 | w)
predict(model, group = c(-1, -1), group2 = c(0, 1), newdata = newdata)

# Predict conditional expectations of the dependent variable
# E(y0 | z1 = 2, z2 = 1, w), E(g1 | z1 = 2, z2 = 1, w)
predict(model, group = c(2, 1), group2 = c(0, 1), newdata = newdata)

# Marginal effect of w3 on
# E(y1 | z1 = 2, z2 = 1, w) and E(g1 | z1 = 2, z2 = 1, w)
predict(model, group   = c(2, 1), group2 = c(0, 1),
```

```
               newdata = newdata, me     = "w3")

# ---
# Step 4
# Two-step estimation procedure
# ---

# Manually provide selectivity terms
model2 <- msel(list(z1 ~ w1 + w2 | w2 + w3, z2 ~ w1 + w3),
               list(y ~ w1 + w3 + w4 +
                        lambda1 + lambda2 + I(lambda1 * lambda2),
                    g ~ w2 + w3 + w5 + lambda1 + lambda2),
               groups = groups, groups2   = groups2,
               data   = data,   estimator = "2step")
summary(model2)



# ----------------------------------------
# Simulated data example 8
# Multinomial endogenous switching and
# selection model (probit)
# ----------------------------------------

# Load required package
library("mnorm")

# ---
# Step 1
# Simulation of the data
# ---

# Set seed for reproducibility
set.seed(123)

# The number of observations
n <- 10000

# Random errors
  # variances and correlations
sd.z2    <- sqrt(0.9)
cor.z    <- 0.3
sd.y0    <- sqrt(2)
cor.z1y0 <- 0.4
cor.z2y0 <- 0.7
sd.y1    <- sqrt(1.8)
cor.z1y1 <- 0.3
cor.z2y1 <- 0.6
cor.y    <- 0.8
  # the covariance matrix
sigma <- matrix(c(
1,                cor.z * sd.z2,           cor.z1y0 * sd.y0,         cor.z1y1 * sd.y1,
cor.z * sd.z2,   sd.z2 ^ 2,            cor.z2y0 * sd.z2 * sd.y0, cor.z2y1 * sd.z2 * sd.y1,
```

```
cor.z1y0 * sd.y0, cor.z2y0 * sd.z2 * sd.y0, sd.y0 ^ 2,              cor.y * sd.y0 * sd.y1,
cor.z1y1 * sd.y1, cor.z2y1 * sd.z2 * sd.y1, cor.y * sd.y0 * sd.y1,    sd.y1 ^ 2),
                ncol = 4, byrow = TRUE)
colnames(sigma) <- c("z1", "z2", "y0", "y1")
rownames(sigma) <- colnames(sigma)

# Simulate the random errors
errors <- rmnorm(n, c(0, 0, 0, 0), sigma)
u      <- errors[, 1:2]
eps    <- errors[, 3:4]

# Regressors (covariates)
x1 <- runif(n, -1, 1)
x2 <- runif(n, -1, 1)
x3 <- (x2 + runif(n, -1, 1)) / 2
W  <- cbind(1, x1, x2)
X  <- cbind(1, x1, x3)

# Coefficients
gamma0 <- c(0.1,    1,   1)
gamma1 <- c(0.2, -1.2, 0.8)
beta0  <- c(1,     -3,   4)
beta1  <- c(1,      4,  -3)

# Linear predictors (indexes)
z1.li <- W %*% gamma0
z2.li <- W %*% gamma1
y0.li <- X %*% beta0
y1.li <- X %*% beta1

# Latent variables
z1.star <- z1.li + u[, 1]
z2.star <- z2.li + u[, 2]
y0.star <- y0.li + eps[, 1]
y1.star <- y1.li + eps[, 2]

# Observable variable as a dummy
z1 <- (z1.star > z2.star) & (z1.star > 0)
z2 <- (z2.star > z1.star) & (z2.star > 0)
z3 <- (z1 != 1) & (z2 != 1)

# Observable multinomial variable
z     <- rep(0, n)
z[z1] <- 0
z[z2] <- 1
z[z3] <- 2
table(z)

# Make unobservable values of the continuous outcome
y         <-  rep(NA, n)
y[z == 1] <- y0.star[z == 1]
y[z == 2] <- y1.star[z == 2]
```

```
# Data
data <- data.frame(z = z, y = y, x1 = x1, x2 = x2, x3 = x3)

# ---
# Step 2
# Estimation of the parameters
# ---

# Define the groups
groups3 <- c(0, 1, 2)
groups2 <- matrix(c(-1, 0, 1), ncol = 1)

# Two-step method
model <- msel(formula3  = z ~ x1 + x2, formula2 = y ~ x1 + x3,
              groups3   = groups3,     groups2   = groups2,
              data      = data,        estimator = "2step",
              type3     = "probit")
summary(model)

# Get the coefficients of the selectivity terms
coef_lambda0_est <- coef(model, type = "coef_lambda", eq2 = 1, regime = 0)
coef_lambda1_est <- coef(model, type = "coef_lambda", eq2 = 1, regime = 1)

# Compare the estimates and true values of the parameters
  # regression coefficients of the continuous equation
beta0_est <- coef(model, type = "coef2", eq2 = 1, regime = 0)
beta1_est <- coef(model, type = "coef2", eq2 = 1, regime = 1)
cbind(true = beta0, est = beta0_est[1:length(beta0)])
cbind(true = beta1, est = beta1_est[1:length(beta1)])
  # regression coefficients of the multinomial equations
gamma0_est <- coef(model, type = "coef3", eq3 = 0)
gamma1_est <- coef(model, type = "coef3", eq3 = 1)
cbind(true = gamma0, est = gamma0_est)
cbind(true = gamma1, est = gamma1_est)
  # compare the covariances between
  # z1 and z2
cbind(true = cor.z * sd.z2,
      est  = coef(model, type = "cov3", eq3 = c(0, 1)))
  # z1 and y0
cbind(true = cor.z1y0 * sd.y0,
      est  = coef_lambda0_est["lambda1_mn"])
  # z2 and y0
cbind(true = cor.z2y0 * sd.y0,
      est  = coef_lambda0_est["lambda2_mn"])
  # z1 and y1
cbind(true = cor.z1y1 * sd.y1,
      est  = coef_lambda1_est["lambda1_mn"])
  # z2 and y1
cbind(true = cor.z2y1 * sd.y1,
      est  = coef_lambda1_est["lambda2_mn"])

# ---
# Step 3
```

```
# Predictions and marginal effects
# ---

# Unconditional expectation E(y1 | w) for every observation in the sample
predict(model, type = "val", group2 = 1, group3 = -1)

# The marginal effect of x1 on conditional expectation E(y0 | z = 1, w)
# for every observation in a sample
predict(model, type = "val", group2 = 0, group3 = 1, me = "x1")

# Calculate predictions and marginal effects
# for manually provided observations
# using the aforementioned models
newdata <- data.frame(z  = c(1, 1),
                      y  = c(1, 1),
                      x1 = c(0.5, 0.2),
                      x2 = c(-0.3, 0.8),
                      x3 = c(0.6, -0.7))

# Unconditional expectation E(y0 | w)
predict(model, type = "val", group2 = 0, group3 = -1, newdata = newdata)

# Conditional expectation E(y1 | z=2, w)
predict(model, type = "val", group2 = 1, group3 = 2, newdata = newdata)

# Marginal effect of x2 on E(y0 | z = 1, w)
predict(model, type = "val", group2  = 0, group3 = 1,
               me   = "x2",  newdata = newdata)

# ---
# Step 4
# Multinomial logit selection
# ---

# Two-step method
model2 <- msel(formula3  = z ~ x1 + x2, formula2 = y ~ x1 + x3,
               groups3   = groups3,     groups2   = groups2,
               data      = data,        estimator = "2step",
               type3     = "logit")
summary(model2)

# Compare the estimates
beta0_est2 <- coef(model2, type = "coef2", eq2 = 1, regime = 0)[]
beta1_est2 <- coef(model2, type = "coef2", eq2 = 1, regime = 1)

# beta0 coefficients
cbind(true = beta0, probit = beta0_est[1:3], logit = beta0_est2[1:3])

# beta1 coefficients
cbind(true = beta1, probit = beta1_est[1:3], logit = beta1_est2[1:3])
```

```
# ---------------------------------------
# Simulated data example 9
# Multinomial endogenous switching and
# selection model (logit)
# ---------------------------------------

# ---
# Step 1
# Simulation of the data
# ---

# Set seed for reproducibility
set.seed(123)

# The number of observations
n <- 10000

# Random errors
u      <- matrix(-log(-log(runif(n * 3))), nrow = n, ncol = 3)
tau0   <- matrix(c(0.6, -0.4, 0.3), ncol = 1)
tau1   <- matrix(c(-0.3, 0.5, 0.2), ncol = 1)
eps0   <- (u - 0.57721656649) %*% tau0 + rnorm(n)
eps1   <- (u - 0.57721656649) %*% tau1 + rnorm(n)

# Regressors (covariates)
x1 <- runif(n = n, min = -1, max = 1)
x2 <- runif(n = n, min = -1, max = 1)
x3 <- runif(n = n, min = -1, max = 1)

# Coefficients
gamma.0 <- c(0.2, -2,  2)
gamma.1 <- c(0.1,  2, -2)
beta.0  <- c(2,    2,  2)
beta.1  <- c(1,   -2,  2)

# Linear predictors (indexes)
z0.li <- gamma.0[1] + gamma.0[2] * x1 + gamma.0[3] * x2
z1.li <- gamma.1[1] + gamma.1[2] * x1 + gamma.1[3] * x2

# Latent variables
z0.star <- z0.li + u[, 1]
z1.star <- z1.li + u[, 2]
z2.star <- u[, 3]
y0.star <- beta.0[1] + beta.0[2] * x1 + beta.0[3] * x3 + eps0
y1.star <- beta.1[1] + beta.1[2] * x1 + beta.1[3] * x3 + eps1

# Observable multinomial variable
z                                          <- rep(2, n)
z[(z0.star > z1.star) & (z0.star > z2.star)] <- 0
z[(z1.star > z0.star) & (z1.star > z2.star)] <- 1
table(z)

# Unobservable values of the continuous outcome
```

```
y         <- rep(NA, n)
y[z == 0] <- y0.star[z == 0]
y[z == 1] <- y1.star[z == 1]

# Data
data <- data.frame(x1 = x1, x2 = x2, x3 = x3, z = z, y = y)

# ---
# Step 2
# Estimation of the parameters
# ---

# Define the groups
groups3 <- c(0, 1, 2)
groups2 <- c(0, 1, -1)

# Two-step estimator of Dubin-McFadden
model <- msel(formula3  = z ~ x1 + x2, formula2 = y ~ x1 + x3,
              groups3   = groups3,     groups2   = groups2,
              data      = data,        estimator = "2step",
              type3     = "logit")
summary(model)

# Least squares estimates (benchmark)
lm0 <- lm(y ~ x1 + x3, data = data[data$z == 0, ])
lm1 <- lm(y ~ x1 + x3, data = data[data$z == 1, ])


# Compare the estimates of beta0
cbind(true = beta.0,
      DMF  = coef(model, type = "coef2", eq2 = 1, regime = 0),
      LS   = coef(lm0))

# Compare the estimates of beta1
cbind(true = beta.1,
      DMF  = coef(model, type = "coef2", eq2 = 1, regime = 1),
      LS   = coef(lm1))
```

`nobs.msel` *Extract the Number of Observations from a Fit of the msel Function.*

Description

Extract the number of observations from a model fit of the `msel` function.

Usage

```
## S3 method for class 'msel'
nobs(object, ...)
```

Arguments

object	object of class "msel"
...	further arguments (currently ignored)

Details

Unobservable values of continuous equations are included in the number of observations.

Value

A single positive integer number.

---

predict.msel *Predict method for msel function*

Description

Predicted values based on the object of class 'msel'.

Usage

```
## S3 method for class 'msel'
predict(
  object,
  ...,
  newdata = NULL,
  given_ind = numeric(),
  group = NA,
  group2 = NA,
  group3 = NA,
  type = ifelse(any(is.na(group2)), "prob", "val"),
  me = NULL,
  eps = NULL,
  control = list(),
  test = FALSE,
  exogenous = NULL,
  factors = NULL
)
```

Arguments

object	an object of class "msel".
...	further arguments (currently ignored)
newdata	an optional data frame in which to look for variables with which to predict. If omitted, the original data frame used. This data frame should contain values of dependent variables even if they are not actually needed for prediction (simply assign them with 0 values).

given_ind	a numeric vector of indexes of conditioned components.
group	a numeric vector which i-th element represents a value of the i-th dependent variable. If this value equals -1 then this component will be ignored (useful for estimation of marginal probabilities).
group2	a numeric vector which i-th element represents a value of the i-th dependent variable of the continuous equation. If this value equals -1 then this component will be ignored.
group3	an integer representing the index of the alternative of the multinomial equation. If this value equals -1 then this component will be ignored.
type	a string representing a type of the prediction. See 'Details' for more information.
me	a string representing the name of the variable for which marginal effect should be estimated. It may also equal to a character vector. Then the function returns a numeric matrix which i-th column contains estimates of the marginal effects with respect to marginal[i]. See 'Details' for more information.
eps	a numeric vector of length 1 or 2 used for calculation of the marginal effects. See 'Details' for more information.
control	a list of additional arguments. Currently is not intended for the users.
test	a logical, function or integer. If test = TRUE then the output of the function is supplied to test_msel before return to perform a t-test. If test is a function it will be applied to the output of the predict before test_msel is called. If test is an integer then test_msel will be applied only to the test-th column of the output.
exogenous	a list such that exogenous[[i]] represents the value (or a vector of values of the same size as nrow(newdata)) which will be exogenously assigned to the variable names(exogenous)[[i]] in newdata i.e., newdata[, names(exogenous)[i]] <- exogenous[[i]]. If newdata is NULL and exogenous is not NULL then newdata is set to object$data. This argument is especially useful for the causal inference when some endogenous (dependent) variables should be exogenously assigned with some values i.e., in the right hand side of the formula, formula2 and formula3. The purpose of the exogeneous argument is just a convenience so equivalently it is possible to exogenously provide the values to variables via the newdata argument.
factors	a list which elements are character vectors representing the variables which should be treated as factors when estimation of marginal effects is performed. It is assumed that if one of these variables equals 1 then other are set to 0. Specifically, if me[i] is an element of factors[[j]] for some j, then when estimating marginal effect respect to me[i], the value of eps set to c(0, 1) and the values of other variables in factors[[j]] are set (internally by the function) to 0. If length(me) = 1 then factors may be just a character vector (not a list).

Details

See 'Examples' section of msel for the examples of this function application.

Probabilities of the multivariate ordinal equations

If type = "prob" then the function returns a joint probability that the ordinal outcomes will have values assigned in group. To calculate marginal probabilities set unnecessary group values to -1.

To estimate conditional probabilities provide indexes of the conditioned outcomes through the `given_ind` argument.

For example, if z_{1i}, z_{2i} and z_{3i} are the ordinal outcomes then to estimate $P(z_{1i} = 2|z_{3i} = 0, w_{1i}, w_{2i}, w_{3i})$ set `given_ind = 3` and `groups = c(2, -1, 0)`.

Linear predictors (indexes) of the multivariate ordinal equations

If `type = "li"` or `type = "lp"` then the function returns a matrix whose columns are linear predictors (indexes) of the corresponding equations. If `group[j] = -1` then linear predictors (indexes) associated with the `j`-th ordinal equation will be omitted from the output.

For example, if `group = c(0, -1, 1)` then the function returns a matrix whose first column is $w_{1i}\hat{\gamma}_1$ and the second column is $w_{3i}\hat{\gamma}_3$.

Standard deviations of the multivariate ordinal equations

If `type = "sd"` then the function returns a matrix whose columns are the estimates of the standard deviations of the random errors for the corresponding equations. If `group[j] = -1` then the standard deviations associated with the `j`-th ordinal equation will be omitted from the output.

For example, if `group = c(0, -1, 1)` then the function returns a matrix whose first column is $\hat{\sigma}^*_{1i}$ and the second column is $\hat{\sigma}^*_{3i}$.

Predictions of the continuous outcomes

If `type = "val"` then the function returns the predictions of the conditional (on `group`) expectation of the continuous outcomes in the regimes determined by the `group2` argument. To predict unconditional expectations set `group` to a vector of `-1` values.

For example, suppose that there is a single continuous equation y_i and two ordinal equations z_{1i} and z_{2i}. To estimate $E(y_{2i}|x_i)$ set `group = c(-1, -1)` and `group2 = 2`. To estimate $E(y_{1i}|x_i, z_{1i} = 2, z_{2i} = 0)$ set `group = c(2, 0)` and `group2 = 1`. To estimate $E(y_{0i}|x_i, z_{2i} = 1)$ set `group = c(-1, 1)` and `group2 = 0`.

Suppose that there are two continuous $y_i^{(1)}$, $y_i^{(2)}$ and two ordinal z_{1i}, z_{2i} equations. If `group2 = c(1, 3)` and `group = c(3, 0)` then the function returns a matrix whose first column contains the estimates of $E(y_{1i}^{(1)}|z_{1i} = 3, z_{2i} = 0, x_i^{(1)})$ and the second column contains the estimates of $E(y_{3i}^{(2)}|z_{1i} = 3, z_{2i} = 0, x_i^{(2)})$.

Selectivity terms

If `type = "lambda"` then the function returns a matrix whose `j`-th column is a numeric vector of estimates of the selectivity terms λ_{ji} associated with the ordinal equations. Similarly if `type = "lambda_mn"` then the function returns a numeric matrix with the selectivity terms of the multinomial equations.

Probabilities of the multinomial equation

If `type = "prob_mn"` and `group3 = j` then the function returns a vector of the estimates of the probabilities $P(\tilde{z}_i = j|\tilde{w}_i)$.

Linear indexes (predictors) of the multinomial equation

If `type = "li_mn"` or `type = "lp_mn"` then the function returns a numeric matrix whose `j`-th column is a numeric vector of estimates of the linear predictor (index) associated with the `(j-1)`-th alternative $\tilde{w}_i\tilde{\gamma}_{(j-1)}$.

Estimation of the marginal effects

If me is provided then the function returns the marginal effect of variable me with respect to the statistic determined by the type argument.

For example, if me = "x1" and type = "prob" then the function returns a marginal effect of x1 on the corresponding probability i.e., one that would be estimated if me is NULL.

If length(eps) = 1 then eps is an increment in the numeric differentiation procedure. If eps is NULL then this increment will be selected automatically taking into account the scaling of the variables. If length(eps) = 2 then marginal effects will be estimated as the difference between predicted values when variable me equals eps[2] and eps[1], respectively.

For example, suppose that type = "prob", me = "x1", given_ind = 3 and groups = c(2, -1, 0). Then if eps is NULL or a small number (something like eps = 0.0001) then the following marginal effect will be estimated (via numeric differentiation):

$$\frac{\partial P(z_{1i} = 2 | z_{3i} = 0)}{\partial x_{1i}}.$$

If eps = c(1, 3) then the function estimates the following difference (useful for estimation of marginal effects of ordered covariates):

$$P(z_{1i} = 2 | z_{3i} = 0, x_{1i} = 3) - P(z_{1i} = 2 | z_{3i} = 0, x_{1i} = 1).$$

Notice that the conditioning on w_{ji} has been omitted for brevity.

Causal inference

Argument exogenous is useful for the causal inference. For example, suppose that there are two binary outcomes z_{1i} and z_{2i}. Also z_{1i} is the endogenous regressor for z_{2i}. That is z_{1i} appears both on the left hand side of formula[[1]] and on the right hand side of formula[[2]]. Consider the estimation of the average treatment effect:

$$ATE = P(z_{2i} = 1 | do(z_{1i}) = 1) - P(z_{2i} = 1 | do(z_{1i}) = 0),$$

where do is a do-calculus operator. The estimate of the average treatment effect is as follows:

$$\widehat{ATE} = \frac{1}{n} \sum_{i=1}^{n} p_{1i} - p_{0i},$$

where:

$$p_{1i} = \hat{P}(z_{2i} = 1 | do(z_{1i}) = 1, w_{1i}, w_{2i}^{(*)}),$$
$$p_{0i} = \hat{P}(z_{2i} = 1 | do(z_{1i}) = 0, w_{1i}, w_{2i}^{(*)}).$$

Vector $w_{2i}^{(*)}$ denotes all the regressors w_{2i} except the endogenous one z_{1i}.

To get $\widehat{ATE}$ it is sufficient to make the following steps. First, calculate p_{1i} by setting type = "prob", group = c(-1, 1) and providing the value 1 to z_{1i} through the exogenous argument. Second, calculate p_{0i} by setting type = "prob", group = c(-1, 0) and providing the value 0 to z_{1i} through the exogenous argument. Third, take the average value of $p_{1i} - p_{0i}$.

Value

This function returns predictions for each row of newdata or for each observation in the model if newdata is NULL. Structure of the output depends on the type argument (see 'Details' section).

print.lrtest_msel *Print Method for Likelihood Ratio Test*

Description

Prints summary for an object of class 'lrtest_msel'.

Usage

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

Arguments

x	object of class "summary.lrtest_msel".
...	further arguments (currently ignored).

Value

The function returns the input argument x.

print.msel *Print for an Object of Class msel*

Description

Prints information on the object of class 'msel'.

Usage

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

Arguments

x	object of class 'msel'
...	further arguments (currently ignored)

Value

The function returns NULL.

print.struct_msel *Print for an Object of Class struct_msel*

Description

Prints information on the object of class 'struct_msel'.

Usage

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

Arguments

x	object of class 'struct_msel'
...	further arguments (currently ignored)

Value

The function returns NULL.

print.summary.lrtest_msel *Print Summary Method for Likelihood Ratio Test*

Description

Prints summary for an object of class 'lrtest_msel'.

Usage

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

Arguments

x	object of class "lrtest_msel"
...	further arguments (currently ignored)

Value

The function returns input argument x changing its class to lrtest_msel.

print.summary.msel *Print summary for an Object of Class msel*

Description

Prints summary for an object of class 'msel'.

Usage

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

Arguments

x	object of class 'msel'
...	further arguments (currently ignored)

Value

The function returns x.

print.summary.test_msel *Print summary for an Object of Class test_msel*

Description

Prints summary for an object of class 'test_msel'.

Usage

```
## S3 method for class 'summary.test_msel'
print(x, ..., is_legend = TRUE)
```

Arguments

x	object of class 'test_msel'
...	further arguments (currently ignored)
is_legend	a logical; if TRUE then additional information is shown.

Value

The function returns input argument x.

sigma.msel *Extract Residual Standard Deviation 'Sigma'*

Description

Extract standard deviations of random errors of continuous equations of `msel` function.

Usage

```
## S3 method for class 'msel'
sigma(object, use.fallback = TRUE, ..., regime = NULL, eq2 = NULL)
```

Arguments

`object`	object of class "msel".
`use.fallback`	logical, passed to `nobs` (currently ignored).
`...`	further arguments (currently ignored).
`regime`	regime of continuous equation
`eq2`	index of continuous equation

Details

Available only if `estimator = "ml"` or all `degrees` values are equal to `1`.

Value

Returns estimates of the standard deviations of ε_i. If `eq2 = k` then estimates only for k-th continuous equation are returned. If in addition `regime = r` then estimate of $\sqrt{Var(\varepsilon_{ri})}$ is returned. Herewith if `regime` is not `NULL` and `eq2` is `NULL` it is assumed that `eq2 = 1`.

starsVector *Stars for p-values*

Description

This function assigns stars (associated with different significance levels) to p-values.

Usage

```
starsVector(p_value)
```

Arguments

`p_value`	vector of values between 0 and 1 representing p-values.

Details

Three stars are assigned to p-values not greater than 0.01. Two stars are assigned to p-values greater than 0.01 and not greater than 0.05. One star is assigned to p-values greater than 0.05 and not greater than 0.1.

Value

The function returns a string vector of stars assigned according to the rules described in 'Details' section.

Examples

```
p_value <- c(0.002, 0.2, 0.03, 0.08)
starsVector(p_value)
```

---

struct_msel *Structure of the Object of Class msel*

Description

Prints information on the structure of the model.

Usage

```
struct_msel(x)
```

Arguments

x	object of class 'msel'

Value

The function returns a numeric matrix whose columns are `groups`, `groups2`, and `groups3`, respectively. It also has additional (last) column with the number of observations associated with the corresponding combinations of the groups.

summary.lrtest_msel *Summary Method for Likelihood Ratio Test*

Description

Provides summary for an object of class 'lrtest_msel'.

Usage

```
## S3 method for class 'lrtest_msel'
summary(object, ...)
```

Arguments

object	object of class "lrtest_msel"
...	further arguments (currently ignored)

Details

This function just changes the class of the 'lrtest_msel' object to 'summary.lrtest_msel'.

Value

Returns an object of class 'summary.lrtest_msel'.

summary.msel *Summary for an Object of Class msel*

Description

Provides summary for an object of class 'msel'.

Usage

```
## S3 method for class 'msel'
summary(object, ..., vcov = NULL, show_ind = FALSE)
```

Arguments

object	object of class 'msel'
...	further arguments (currently ignored)

vcov	positively defined numeric matrix representing asymptotic variance-covariance matrix of the estimator to be used for calculation of standard errors and p-values. It may also be a character. Then the vcov.msel function will be used, whose input argument type will be set to vcov. If estimator = "2step" then vcov should be an estimate of the asymptotic covariance matrix of the first step estimator.
show_ind	logical; if TRUE then indexes of parameters will be shown. Particularly, these indexes may be used in ind element of regularization parameter of msel.

Details

If vcov is NULL then this function just changes the class of the 'msel' object to 'summary.msel'. Otherwise it additionally changes object$cov to vcov and uses it to recalculate object$se, object$p_value and object$tbl values. It also adds the value of show_ind argument to the object.

Value

Returns an object of class 'summary.msel'.

---

summary.test_msel *Summary for an Object of Class delta_method*

Description

Provides summary for an object of class 'delta_method'.

Usage

```
## S3 method for class 'test_msel'
summary(object, ..., is_legend = TRUE)
```

Arguments

object	object of class 'delta_method'
...	further arguments (currently ignored)
is_legend	a logical; if TRUE then additional information is shown.

Value

Returns an object of class 'summary.delta_method'.

test_msel *Tests and confidence intervals for the parameters estimated by the msel function*

Description

This function conducts various statistical tests and calculates confidence intervals for the parameters of the model estimated via the msel function.

Usage

```
test_msel(
  object,
  fn,
  fn_args = list(),
  test = "t",
  method = "classic",
  ci = "classic",
  cl = 0.95,
  se_type = "dm",
  trim = 0,
  vcov = NULL,
  iter = 100,
  generator = rnorm,
  bootstrap = NULL,
  par_ind = NULL,
  eps = max(1e-04, sqrt(.Machine$double.eps) * 10),
  n_sim = 1000,
  n_cores = 1
)
```

Arguments

object	an object of class 'msel'. It also may be a list of two objects. Then object[[1]] and object[[2]] are supplied to the arguments model1 and model2 of the lrtest_msel function.
fn	a function which returns a numeric vector and should depend on the elements of object. These elements should be accessed via coef.msel or predict.msel functions. The first argument of fn should be an object. Therefore coef and predict functions in fn should also depend on object.
fn_args	a list of additional arguments of fn.
test	a character representing the test to be used. If test = "t" then t-test is used. If test = "wald" then Wald test is applied.
method	a character representing a method used to conduct a test. If test = "t" or test = "wald" and method = "classic" then p-values are calculated by using the quantiles of the standard normal distribution. If test = "t" or test = "wald"

and method = "bootstrap" then p-values are calculated by using the bootstrap as described in Hansen (2022). If test = "wald" and method = "score" then score bootstrap Wald test of P. Kline and A. Santos (2012) is used.

ci — a character representing the type of the confidence interval used. Available only if test = "t". If ci = "classic" then quantiles of the standard normal distribution are used to build an asymptotic confidence interval. If ci = "percentile" then percentile bootstrap interval is applied. If ci = "bc" then the function constructs a bias-corrected percentile bootstrap confidence interval of Efron (1982) as described in Hansen (2022).

cl — a numeric value between 0 and 1 representing a confidence level of the confidence interval.

se_type — a character representing a method used to estimate the standard errors of the outputs of fn. If se_type = "dm" then delta method is used. If se_type = "bootstrap" then bootstrap is applied.

trim — a numeric value between 0 and 1 representing the share of bootstrap estimates to be nullified when standard errors are estimated for se_type = "bootstrap".

vcov — an estimate of the asymptotic covariance matrix of the parameters of the model.

iter — the number of iterations used by the score bootstrap Wald test.

generator — a function which is used by the score bootstrap to generate random weights. It should have an argument n representing the number of random weights to generate. Other arguments are ignored.

bootstrap — an object of class 'bootstrap_msel' which is an output of the bootstrap_msel function. This object is used to retrieve the estimates of the bootstrap samples.

par_ind — a vector of indexes of the model parameters used in the calculation of fn. If only necessary indexes are included then in some cases estimation time may greatly decrease. Set ind = TRUE in summary.msel to see the indexes of the model parameters. If eps is a vector then eps[i] determines the increment used to differentiate fn with respect to the parameter with par_ind[i]-th index.

eps — a positive numeric value representing the increment used for the numeric differentiation of fn. It may also be a numeric vector such that eps[i] is an increment used to differentiate the fn with respect to the par_ind[i]-th parameter of the model. Set ind = TRUE in summary.msel, to see the indexes of the model parameters. If eps[i] = 0 then derivative of fn with respect to par_ind[i]-th parameter is assumed to be zero.

n_sim — the value passed to the n_sim argument of the msel function.

n_cores — the value passed to the n_cores argument of the msel function.

Details

Suppose that θ is a vector of parameters of the model estimated via the msel function and $g(\theta)$ is a differentiable function representing fn which returns a m-dimensional vector of real values:

$$g(\theta) = (g_1(\theta), ..., g_m(\theta))^T.$$

Classic and bootstrap t-test

If `test = "t"` then for each $j \in \{1, ..., m\}$ the following hypothesis is tested:

$$H_0 : g_j(\theta) = 0, \qquad H_1 : g_j(\theta) \neq 0.$$

The test statistic is:

$$T = g_j(\hat{\theta})/\hat{\sigma}_j,$$

where $\hat{\sigma}$ is a standard error of $g_j(\hat{\theta})$.

If `se_type = "dm"` then delta method is used to estimate this standard error:

$$\hat{\sigma}_j = \sqrt{\nabla g_j(\hat{\theta})^T \widehat{As.Cov}(\hat{\theta}) \nabla g_j(\hat{\theta})},$$

where $\nabla g_j(\hat{\theta})$ is a gradient as a column vector and the estimate of the asymptotic covariance matrix of the estimates $\widehat{As.Cov}(\hat{\theta})$ is provided via the `vcov` argument. Numeric differentiation is used to calculate $\nabla g_j(\hat{\theta})$.

If `se_type = "bootstrap"` then bootstrap is applied to estimate the standard error:

$$\hat{\sigma}_j = \sqrt{\frac{1}{B-1} \sum_{b=1}^{B} (g_j(\hat{\theta}^{(b)}) - \overline{g_j(\hat{\theta}^{(b)})})^2},$$

where B is the number of the bootstrap iterations `bootstrap$iter`, $\hat{\theta}^{(b)}$ is the estimate associated with the b-th of these iterations `bootstrap$par[b, ]`, and $\overline{g_j(\hat{\theta}^{(b)})}$ is a sample mean of the bootstrap estimates estimates:

$$\overline{g_j(\hat{\theta}^{(b)})} = \frac{1}{B} \sum_{b=1}^{B} g_j(\hat{\theta}^{(b)}).$$

If `method = "classic"` it is assumed that if the null hypothesis is true then the asymptotic distribution of the test statistic is standard normal. This distribution is used for the calculation of the p-value:

$$p - value = 2 \min(\Phi(T), 1 - \Phi(T)),$$

where $\Phi()$ is a cumulative distribution function of the standard normal distribution.

If `method = "bootstrap"` then p-value is calculated via the bootstrap as suggested by Hansen (2022):

$$p - value = \frac{1}{B} \sum_{b=1}^{B} I(|T_b - T| > |T|),$$

where $T_b = g_j(\hat{\theta}^{(b)})/\hat{\sigma}_j$ is the value of the test statistic estimated on the b-th bootstrap sample and $I(q)$ is an indicator function which equals 1 when q is a true statement and 0 - otherwise.

Classic and bootstrap Wald test

Suppose that `method = "classic"` or `method = "bootstrap"`. If `test = "wald"` then the null hypothesis is:

$$H_0 : \begin{cases} g_1(\theta) = 0 \\ g_2(\theta) = 0 \\ \vdots \\ g_m(\theta) = 0 \end{cases}.$$

The alternative hypothesis is that there is such $j \in \{1, ..., m\}$ that:

$$H_1 : g_j(\theta) \neq 0.$$

The test statistic is:

$$T = g(\hat{\theta})^T \widehat{As.Cov}(g(\hat{\theta}))^{-1} g(\hat{\theta}),$$

where $\widehat{As.Cov}(g(\hat{\theta}))$ is the estimate of the asymptotic covariance matrix of $g(\hat{\theta})$.

If `se_type = "dm"` then delta method is used to estimate this matrix:

$$\widehat{As.Cov}(g(\hat{\theta})) = g'(\hat{\theta}) \widehat{As.Cov}(\hat{\theta}) g'(\hat{\theta})^T,$$

where $g'(\hat{\theta})$ is a Jacobian matrix. Numeric differentiation is used to calculate its elements:

$$g'(\hat{\theta})_{ij} = \frac{\partial g_i(\theta)}{\partial \theta_j}|_{\theta=\hat{\theta}}.$$

If `se_type = "bootstrap"` then bootstrap is used to estimate this matrix:

$$\widehat{As.Cov}(g(\hat{\theta})) = \frac{1}{B-1} \sum_{i=1}^{B} q_b q_b^T,$$

where:

$$q_b = (g(\hat{\theta}^{(b)}) - \overline{g(\hat{\theta}^{(b)})}),$$

$$\overline{g(\hat{\theta}^{(b)})} = \frac{1}{B} \sum_{i=1}^{B} g(\hat{\theta}^{(b)}).$$

If `method = "classic"` then it is assumed that if the null hypothesis is true then the asymptotic distribution of the test statistic is chi-squared with m degrees of freedom. This distribution is used for the calculation of the p-value:

$$p - value = 1 - F_m(T),$$

where F_m is a cumulative distribution function of the chi-squared distribution with m degrees of freedom.

If `method = "bootstrap"` then p-value is calculated via the bootstrap as suggested by Hansen (2022):

$$p - value = \frac{1}{B} \sum_{b=1}^{B} I(T_b > T),$$

where:

$$T_b = s_b^T \widehat{As.Cov}(g(\hat{\theta}))^{-1} s_b,$$
$$s_b = g(\hat{\theta}^{(b)}) - g(\hat{\theta}).$$

Score bootstrap Wald test

If `method = "score"` and `test = "Wald"` then score bootstrap Wald test of Kline and Santos (2012) is used.

Consider B independent samples of n independent identically distributed random weights with zero mean and unit variance. Let w_{ib} denote the i-th weight of the b-th sample. Argument `generator` is used to supply a function which generates these weights w_{ib} and `iter` argument represents B. Also n is the number of observations in the model `object$other$n_obs`.

Let J denote a matrix of sample scores `object$J`. Further, denote by J_b a matrix such that its b-th row is a product of the w_{ib} and the b-th row of J. Also, denote by H a matrix of mean values of the derivatives of sample scores with respect to the estimated parameters `object$H`.

In addition consider the following notations:

$$A = g'(\theta)H^{-1}, \qquad S_b = AJ_b^{(c)},$$

where $J_b^{(c)}$ is a vector of the column sums of J_b.

The test statistic is as follows:

$$T = g(\hat{\theta})^T (A\widehat{Cov}(J)A^T)^{-1} g(\hat{\theta})/n,$$

where $\widehat{Cov}(J)$ is a sample covariance matrix of the sample scores of the model `cov(object$J)`.

The test statistic on the b-th bootstrap sample is similar:

$$T_b = S^T (A\widehat{Cov}(J_b)A^T)^{-1} S/n.$$

The p-value is estimated as follows:

$$p - value = \frac{1}{B}\sum_{b=1}^{B} I(T_b > T).$$

Confidence intervals

If `test = "t"` then the function also returns the realizations of the lower and upper bounds of the $100\times$`cl` percent symmetric asymptotic confidence interval of $g_j(\theta)$.

If `ci = "classic"` then classic confidence interval is used which assumes asymptotic normality of $g_j(\hat{\theta})$:

$$(g_j(\hat{\theta}) + z_{(1-cl)/2}\hat{\sigma}_j, g_j(\hat{\theta}) + z_{1-(1-cl)/2}\hat{\sigma}_j),$$

where z_q is the q-th quantile of the standard normal distribution and cl is a confidence level `cl`. The method used to estimate $\hat{\sigma}_j$ depends on the `se_type` argument as described above.

If `ci = "percentile"` then percentile bootstrap confidence interval is used. Therefore the sample quantiles of $g_j(\hat{\theta}^{(b)})$ are used as the realizations of the lower and upper bounds of the confidence interval.

If ci = "bc" then bias-corrected percentile bootstrap confidence interval of Efron (1982) is used as described in Hansen (2022). The default percentile bootstrap confidence interval uses sample quantiles of levels $(1-cl)/2$ and $1-(1-cl)/2$. Bias-corrected version uses the sample quantiles of the following levels:

$$(1-cl)/2+\Phi(\Phi^{-1}((1-cl)/2)+s),$$

$$1-(1-cl)/2+\Phi(\Phi^{-1}(1-(1-cl)/2)+s),$$

where:

$$s=2\Phi^{-1}(\frac{1}{B}\sum_{b=1}^{B}I(g_j(\hat{\theta}^{(b)})\leq g_j(\hat{\theta}))).$$

Trimming

If se_type = "bootstrap" and trim > 0 then trimming is used as described in Hansen (2022) to estimate $\hat{\sigma}_j$ and $\widehat{As.Cov}(g(\hat{\theta}))$. The algorithm is as follows. First, nullify 100trim percent of $g(\hat{\theta}^{(b)})$ with the greatest values of the L2-norm of q_b (defined above). Then use this 'trimmed' sample to estimate the standard error and the asymptotic covariance matrix.

Cross-model tests

It is also possible to test the hypothesis on the parameters estimated via several different models. To do so it is necessary to provide a list of models (objects of class msel) via the object argument. Particularly, cross-model tests are useful for comparing marginal and treatment effects among the different models. In addition, cross-model tests may be used for testing over-identifying restrictions.

If se_type = "dm" then the joint covariance matrix of the estimators is obtained via the stacking method similar to one implemented by suest postestimation command in STATA. Note that there are the following requirements:

- Covariance matrix type cov_type of each object[[i]] should be either "mm" or "sandwich".
- Argument data of each object[[i]] should contain a special variable crossind. Suppose that each object[[i]]$data is a subsample of some data0. Then crossind argument should take different values in range between 1 and nrow(data0). In addition, observations with the same value of crossind in object[[i]]$data should correspond to the same observation in data0. A simple way to create crossind variable is just to set it equal to 1:nrow(data0) in data0 before splitting it into subsamples used by the models.
- Argument cluster of each object[[i]] is ignored. However, clustering may be provided via crosscluster variable in object[[i]]$data. Then observations with the same value of crosscluster are considered to be from the same cluster.

Arguments eps and par_ind may be lists whose elements are associated with the corresponding models. If eps is a scalar, then the same increment will be used for the numeric differentiation with respect to parameters of all models. Also, by default indexes of all parameters of all the models will be included in par_ind.

Value

This function returns an object of class 'test_msel' which is a list. It may have the following elements:

- tbl - a list with the elements described below.

- `is_bootstrap` - a logical value which equals `TRUE` if bootstrap has been used.
- `is_ci` - a logical value which equals `TRUE` if confidence intervals were used.
- `test` - the same as the input argument `test`.
- `method` - the same as the input argument `method`.
- `se_type` - the same as the input argument `se_type`.
- `ci` - the same as the input argument `ci`.
- `cl` - the same as the input argument `cl`.
- `iter` - the same as the input argument `iter`.
- `n_bootstrap` - an integer representing the number of bootstrap iterations used.
- `n_val` - the length of the vector returned by `fn`.

A list `tbl` may have the following elements:

- `val` - an output of the `fn` function.
- `se` - a numeric vector such that `se[i]` represents a standard error associated with `val[i]`.
- `p_value` - a numeric vector of p-values.
- `lwr` - a numeric vector such that `lwr[i]` is the realization of the lower (left) bound of the confidence interval for the true value of `val[i]`.
- `upr` - a numeric vector such that `upr[i]` is the realization of the upper (right) bound of the confidence interval for the true value of `val[i]`.
- `stat` - a numeric vector of values of the test statistics.

An object of class `'test_msel'` has an implementation of the `summary` method `summary.test_msel`.

In a special case when `test = "lr"` and `object` is a list of length 2 the function returns an object of class `'lrtest_msel'` since the function `lrtest_msel` is called internally.

References

B. Efron (1982). The Jackknife, the Bootstrap, and Other Resampling Plans. Society for Industrial and Applied Mathematics.

B. Hansen (2022). Econometrics. Princeton University Press.

P. Kline, A. Santos (2012). A Score Based Approach to Wild Bootstrap Inference. Journal of Econometric Methods, vol. 67, no. 1, pages 23-41.

Examples

```
# ------------------------------
# CPS data example
# ------------------------------

# Set seed for reproducibility
set.seed(123)

# Load the data
data(cps)
```

```
# Estimate the employment model
model <- msel(work ~ age + I(age ^ 2) + bachelor + master, data = cps)
summary(model)

# Use Wald test to test the hypothesis that age has no
# effect on the conditional probability of employment:
# H0: coef age     = 0
#     coef age ^ 2 = 0
age_fn <- function(object)
{
  lwage_coef <- coef(object, type = "coef")[[1]]
  val        <- c(lwage_coef["age"], lwage_coef["I(age^2)"])
  return(val)
}
age_test <- test_msel(object = model, fn = age_fn, test = "wald")
summary(age_test)

# Use t-test to test for each individual the hypothesis:
# P(work = 1 | x) = 0.8
prob_fn <- function(object)
{
  prob <- predict(object, group = 1, type = "prob")
  val  <- prob - 0.8
  return(val)
}
prob_test <- test_msel(object = model, fn = prob_fn, test = "t")
summary(prob_test)

# --------------------------------
# Simulated data example
# Model with continuous outcome
# and ordinal selection
# --------------------------------

# ---
# Step 1
# Simulation of the data
# ---

# Set seed for reproducibility
set.seed(123)

# Load required package
library("mnorm")

# The number of observations
n <- 10000

# Regressors (covariates)
s1 <- runif(n = n, min = -1, max = 1)
s2 <- runif(n = n, min = -1, max = 1)
s3 <- runif(n = n, min = -1, max = 1)
```

```
s4 <- runif(n = n, min = -1, max = 1)

# Random errors
sigma <- matrix(c(1,    0.4,  0.45, 0.7,
                  0.4,  1,    0.54, 0.8,
                  0.45, 0.54, 0.81, 0.81,
                  0.7,  0.8,  0.81, 1), nrow = 4)
errors <- mnorm::rmnorm(n = n, mean = c(0, 0, 0, 0), sigma = sigma)
u1   <- errors[, 1]
u2   <- errors[, 2]
eps0 <- errors[, 3]
eps1 <- errors[, 4]

# Coefficients
gamma1     <- c(-1, 2)
gamma2     <- c(1, 1)
gamma1_het <- c(0.5, -1)
beta0      <- c(1, -1, 1, -1.2)
beta1      <- c(2, -1.5, 0.5, 1.2)
# Linear index of the ordinal equations
# mean part
li1 <- gamma1[1] * s1 + gamma1[2] * s2
li2 <- gamma2[1] * s1 + gamma2[2] * s3
# variance part
li1_het <- gamma1_het[1] * s2 + gamma1_het[2] * s3

# Linear index of the continuous equation
# regime 0
li_y0 <- beta0[1] + beta0[2] * s1 + beta0[3] * s3 + beta0[4] * s4
# regime 1
li_y1 <- beta1[1] + beta1[2] * s1 + beta1[3] * s3 + beta1[4] * s4

# Latent variables
z1_star <- li1 + u1 * exp(li1_het)
z2_star <- li2 + u2
y0_star <- li_y0 + eps0
y1_star <- li_y1 + eps1

# Cuts
cuts1 <- c(-1)
cuts2 <- c(0, 1)

# Observable ordinal outcome
# first
z1                     <- rep(0, n)
z1[z1_star > cuts1[1]] <- 1
# second
z2                                                <- rep(0, n)
z2[(z2_star > cuts2[1]) & (z2_star <= cuts2[2])] <- 1
z2[z2_star > cuts2[2]]                            <- 2
z2[z1 == 0]                                       <- NA

# Observable continuous outcome
```

```
y                <- rep(NA, n)
y[which(z2 == 0)] <- y0_star[which(z2 == 0)]
y[which(z2 != 0)] <- y1_star[which(z2 != 0)]
y[which(z1 == 0)] <- NA

# Data
data <- data.frame(s1 = s1, s2 = s2, s3 = s3, s4 = s4,
                   z1 = z1, z2 = z2, y = y,   crossind = 1:n)

# ---
# Step 2
# Estimation of the parameters
# ---

# Assign the groups
groups  <- matrix(c(1, 2,
                    1, 1,
                    1, 0,
                    0, -1),
                  byrow = TRUE, ncol = 2)
groups2 <- matrix(c(1, 1, 0, -1), ncol = 1)

# Estimate the model
model <- msel(list(z1 ~ s1 + s2 | s2 + s3,
                   z2 ~ s1 + s3),
              list(y  ~ s1 + s3 + s4),
              groups = groups, groups2 = groups2,
              data   = data)

# ---
# Step 3
# Hypotheses testing
# ---

# Use t-test to test for each observation the hypothesis
# H0: P(z1 = 1, z2 = 0 | Xi) = 0
prob02_fn <- function(object)
{
   val <- predict(object, group = c(1, 0))

   return(val)
}
prob02_test <- test_msel(object = model, fn = prob02_fn, test = "t")
summary(prob02_test)

# Use t-test to test the hypothesis
# H0: E(y1|z1=0, z2=2) - E(y0|z1=0, z2=2)
ATE_fn <- function(object)
{
   val1 <- predict(object, group = c(0, 2), group2 = 1)
   val0 <- predict(object, group = c(0, 2), group2 = 0)
   val <- mean(val1 - val0)
```

```
   return(val)
}
ATE_test <- test_msel(object = model, fn = ATE_fn)
summary(ATE_test)

# Use Wald test to test the hypothesis
# H0: beta1 = beta0
coef_fn <- function(object)
{
   coef1 <- coef(object, regime = 1, type = "coef2")
   coef0 <- coef(object, regime = 0, type = "coef2")
   coef_difference <- coef1 - coef0

   return(coef_difference)
}
coef_test <- test_msel(object = model, fn = coef_fn, test = "wald")
summary(coef_test)

# Use t-test to test for each 'k' the hypothesis
# H0: beta1k = beta0k
coef_test2 <- test_msel(object = model, fn = coef_fn, test = "t")
summary(coef_test2)

# Use Wald test to test the hypothesis
# H0: beta11 + beta12 - 0.5 = 0
#     beta11 * beta13 - beta03 = 0
test_fn <- function(object)
{
  coef1 <- coef(object, regime = 1, type = "coef2")
  coef0 <- coef(object, regime = 0, type = "coef2")
  val   <- c(coef1[1] + coef1[2] - 0.5,
  coef1[1] * coef1[3] - coef0[3])

  return(val)
}
# classic Wald test
wald1 <- test_msel(object = model,  fn     = test_fn,
                   test   = "wald", method = "classic")
summary(wald1)
# score bootstrap Wald test
wald2 <- test_msel(object = model,  fn     = test_fn,
                   test   = "wald", method = "score")
summary(wald2)

# Replicate the latter test with the 2-step estimator
model2 <- msel(list(z1 ~ s1 + s2 | s2 + s3,
                    z2 ~ s1 + s3),
               list(y ~ s1 + s3 + s4),
               groups    = groups, groups2   = groups2,
               data      = data,   estimator = "2step")
# classic Wald test
wald1_2step <- test_msel(object = model2, fn     = test_fn,
                         test   = "wald", method = "classic")
```

```
summary(wald1_2step)
# score bootstrap Wald test
wald2_2step <- test_msel(object = model2, fn     = test_fn,
                         test   = "wald", method = "score")
summary(wald2_2step)

# ---
# Step 4
# Cross-model tests
# ---

# Use Wald test to test the hypothesis that coefficients
# in regime 1 are the same in both models
cmtest_fn <- function(object)
{
  coef1 <- coef(object[[1]], regime = 1, type = "coef2")
  coef2 <- coef(object[[2]], regime = 1, type = "coef2")
  val   <- coef1 - coef2

  return(val)
}

# Classic Wald test
test_wald <- test_msel(object = list(model, model2), fn     = cmtest_fn,
                       test   = "wald",              method = "classic")
summary(test_wald)


# Score bootstrap Wald test
test_score <- test_msel(object = list(model, model2), fn     = cmtest_fn,
                        test   = "wald",              method = "score")
summary(test_score)
```

`update_msel` *Update msel object with the new estimates*

Description

This function updates parameters of the model estimated via `msel` function.

Usage

```
update_msel(object, par)
```

Arguments

object	an object of class 'msel'.
par	a vector of parameters which substitutes object$par and is used to update the estimates i.e., object$coef, object$cuts and others.

Details

It may be useful to apply this function to the bootstrap estimates of bootstrap_msel.

Value

This function returns an object of class 'msel' in which object$par is substituted with par. Also, par is used to update the estimates i.e., object$coef, object$cuts and others.

---

vcov.msel *Calculate Variance-Covariance Matrix for a msel Object.*

Description

Return the variance-covariance matrix of the parameters of msel model.

Usage

```
## S3 method for class 'msel'
vcov(
  object,
  ...,
  type = object$cov_type,
  n_cores = object$other$n_cores,
  n_sim = object$other$n_sim,
  recalculate = FALSE,
  cluster = NULL
)
```

Arguments

object	an object of class msel.
...	further arguments (currently ignored).
type	character representing the type of the asymptotic covariance matrix estimator. It takes the same values as cov_type parameter of the msel function.
n_cores	positive integer representing the number of CPU cores used for parallel computing. If possible it is highly recommended to set it equal to the number of available physical cores especially when the system of ordered equations has 2 or 3 equations.

n_sim	integer representing the number of GHK draws when there are more than 3 ordered equations. Otherwise alternative (much more efficient) algorithms will be used to calculate multivariate normal probabilities.
recalculate	logical; if TRUE then covariance matrix will be recalculated even if 'type' is the same as 'cov_type' input argument of the model.
cluster	an object which takes the same values as the cluster argument of the msel function. The only exception is cluster = NULL which implies cluster = object$cluster.

Details

Argument type is closely related to the argument cov_type of msel function. See 'Details' and 'Usage' sections of msel for more information on cov_type argument.

The user may also estimate asymptotic covariance matrix of the parameters of several models. For more information, see paragraph 'Cross-model tests' in 'Details' section of test_msel.

Value

Returns numeric matrix which represents estimate of the asymptotic covariance matrix of model's parameters. If object is a list of models then rows and columns of this matrix will have names "mipj" indicating that corresponding element is associated with j-th parameter of the i-th model.

---

vcov_combine *Combine the asymptotic covariance matrices*

Description

Combine the asymptotic covariance matrices

Usage

```
vcov_combine(object, type = "cov")
```

Arguments

object	a list of objects of class msel.
type	the same as the type argument of the vcov.msel function.

Index

∗ **datasets**
    cps, 9

boot, 2
bootstrap, 3
bootstrap_combine_msel, *3, 4*
bootstrap_combine_msel *(*bootstrap*)*, 3
bootstrap_msel, *3, 4*, *75*, *86*
bootstrap_msel *(*bootstrap*)*, 3

coef.msel, 7, *30*, *74*
cps, 9

dmnorm, *28*

exogenous_fn, 11

fitted.msel, 11
formula.msel, 12
formula_merge, 13
formula_split, 14

gena, *19*, *28*
grad_msel, 14

lnL_msel, 15
logLik, *17*
logLik.msel, 16
loocv, 16
lrtest_msel, 17, *74*, *80*

msel, *3, 4*, *8, 9*, *16*, 18, *29*, *62*, *64*, *70*, *73–75*, *85–87*

nobs, *17*
nobs.msel, 62

optim, *19*, *28*

pmnorm, *19*, *22*, *28*
predict.msel, 63, *74*
print.lrtest_msel, 67
print.msel, 67
print.struct_msel, 68
print.summary.lrtest_msel, 68
print.summary.msel, 69
print.summary.test_msel, 69
pso, *19*, *28*

sigma.msel, 70
starsVector, 70
struct_msel, 71
summary.lrtest_msel, 72
summary.msel, *20*, 72, *75*
summary.test_msel, 73, *80*

test_msel, *4*, *64*, 74, *87*

update_msel, *4*, 85

vcov.msel, *73*, 86, *87*
vcov_combine, 87