Package ‘switchSelection’

August 23, 2023

Type Package

Title Endogenous Switching and Sample Selection Regression Models

Version 1.1.2

Date 2023-08-23


License GPL (>= 2)

Imports Rcpp (>= 1.0.10), hpa (>= 1.3.1), mnorm (>= 1.2.1), gena (>= 1.0.0), methods

LinkingTo Rcpp, RcppArmadillo, hpa, mnorm

RoxygenNote 7.2.3

NeedsCompilation yes

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

Maintainer Bogdan Potanin <bogdanpotanin@gmail.com>

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2023-08-23 15:50:05 UTC

R topics documented:

- boot
- coef.mnprobit
- coef.mvoprobit
- cps
- delta_method

2 3 4 6 7
**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)

---

coef.mnprobit

Coefficients extraction method for mnprobit.

Description

Extract coefficients and other estimates from mnprobit object.

Usage

## S3 method for class 'mnprobit'
coef(object, ..., alt = NULL, regime = NULL, type = "coef")

Arguments

object: object of class "mnprobit"
...: further arguments (currently ignored)
alt: integer representing index of the alternative
regime: integer representing regime of the continuous equation
type character representing the type of the output. Possible options are "coef", "coef2", "cov1", "var", "cov2", coef_lambda. See 'Details' for additional information.

Details

Consider notations from the 'Details' section of \textit{mnprobit}.

Suppose that type = "coef". Then estimates of \( \gamma_j \) coefficients are returned for each \( j \in \{1, ..., J\} \). If \( \text{alt} = j \) then only estimates of \( \gamma_j \) coefficients are returned.

Suppose that type = "coef2". Then estimates of \( \beta_r \) coefficients are returned for each \( r \in \{0, ..., R-1\} \). If \( \text{regime} = r \) then estimates only for the \( r \)-th regime are returned.

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

Suppose that type = "cov12". Then estimates of covariances between \( u_i \) and \( \varepsilon_i \) are returned. If \( \text{alt} = j \) and \( \text{regime} = r \) then the function returns an estimate of \( Cov(u_{ji}, \varepsilon_{ri}) \).

Suppose that type = "var" or type = "cov2". Then estimates of the variances of \( \varepsilon_i \) are returned. If \( \text{regime} = r \) then estimate of \( Var(\varepsilon_{ri}) \) is returned.

Suppose that type = "coef_lambda". Then estimates of the coefficients for \( \hat{\lambda}_{ji} \) are returned i.e. estimates of \( \tau_{jt} \) for each regime. If \( \text{regime} = r \) then estimates are returned for the \( r \)-th regime. If in addition \( \text{alt} = j \) then only estimates for this \( j \)-th alternative and \( r \)-th regime are returned.

Value

See 'Details' section.
coef.mvoprobit

regime  integer representing a regime of the continuous equation.

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

Details

Consider notations from the 'Details' section of mvoprobit.

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

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

Suppose that type = "coef2". Then estimates of $\beta_r$ 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 $\beta_r$ 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.

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

Suppose that type = "cov1". Then 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 a-th row and b-th column.

Suppose that type = "cov12". Then estimates of covariances between $u_i$ and $\varepsilon_i$ 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 estimate of $\text{Cov}(u_{ji}, \varepsilon_{ri})$ for the k-th equation. If eq2 = NULL it is assumed that eq2 = 1.

Suppose that type = "var" or type = "cov2". Then estimates of the variances of $\varepsilon_i$ are returned. If eq2 = k then estimates only for k-th continuous equation are returned. If in addition regime = r then estimate of $\text{Var}(\varepsilon_{ri})$ is returned. Herewith if regime is not NULL and eq2 is NULL it is assumed that eq2 = 1.

Suppose that type = "cov3". Then estimates of the covariances between random errors of different equations in different regimes are returned. If eq2 = c(a, b) and regime = c(c, d) then function returns an estimate of the covariance of random errors of the a-th and b-th continuous equations in regimes c and d correspondingly. If this covariance is not identifiable then NA value is returned.

Suppose that type = "coef_lambda". Then estimates of the coefficients for $\hat{\lambda}_{ji}$ are returned i.e. estimates of $\tau_{jt}$ for each regime. If regime = r then estimates are returned for the r-th regime. If in addition eq = j then only estimates for this j are returned.

Value

See 'Details' section.
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 13 columns. It contains information on wages and some socio-demographic characteristics of middle age (25-54 years) married women:

- age: age of individual measured in years.
- lwage: logarithm of hourly wage.
- slwage: logarithm of hourly wage of a spouse.
- work: binary variable for employment status (0 - unemployed, 1 - employed).
- swork: binary variable for employment status of a spouse (0 - unemployed, 1 - employed).
- nchild: the number of children under age 5.
- health: subjective health status (1 - poor, 2 - fair, 3 - good, 4 - very good, 5 - excellent).
- basic: binary variable which equals 1 for those who have graduated from high school or has at least some college or has associated degree and does not have any higher level of education, 0 - otherwise.
- bachelor: binary variable which equals 1 for those whose highest education level is a bachelor degree.
- master: 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. ...

Source

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

References

**Examples**

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

---

**delta_method**  
*Delta method for mvoprobit and mnprobit functions*

**Description**

This function uses delta method to estimate standard errors of functions of the estimator of the parameters of `mnprobit` and `mvoprobit` functions if maximum-likelihood estimator has been used.

**Usage**

```r
delta_method(
  object,
  fn,
  fn_args = list(),
  eps = max(1e-04, sqrt(.Machine$double.eps) * 10),
  cl = 0.95
)
```

**Arguments**

- `object`: an object of class 'mvoprobit' or 'mnprobit'.
- `fn`: function which returns a numeric vector and should depend on the elements of `object`. This elements should be accessed via `coef.mvoprobit` and `coef.mnprobit` functions. Also it is possible to use `predict.mvoprobit` and `predict.mnprobit` functions. The first argument of `fn` should be `object`. Therefore `coef` and `predict` functions in `fn` should also depend on `object`.
- `fn_args`: list of additional arguments of `fn`.
- `eps`: positive numeric value representing the increment used for numeric differentiation of `fn`.
- `cl`: numeric value between 0 and 1 representing a confidence level of the confidence interval.

**Details**

Numeric differentiation is used to estimate derivatives of `fn` respect to various parameters of `mvoprobit` and `mnprobit` functions.

This function may be used only if `object$estimator = "ml"`. 
Value

This function returns an object of class `delta_method` that is a matrix which columns are as follows:

- **val** - output of the `fn` function.
- **se** - numeric vector such that `se[i]` represents standard error associated with `val[i]`.
- **p_value** - numeric vector such that `p_value[i]` represents p-value of the two-sided significance test associated with `val[i]`.
- **lwr** - realization of the lower (left) bound of the confidence interval.
- **upr** - realization of the upper (right) bound of the confidence interval.

An object of class `delta_method` has implementation of `summary` method `summary.delta_method`.

Examples

```
# 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 ordered equation
## mean
```
# variance

# Linear index of continuous equation
# regime 0
# regime 1

# 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 ordered 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 == 0] <- 2
z2[z1 == 0] <- -1

# Observable continuous outcome such
y <- rep(NA, n)
y[z2 == 0] <- y0_star[z2 == 0]
y[z2 != 0] <- y1_star[z2 != 0]
y[z1 == 0] <- Inf

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

# Assign 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)

# Estimation
model <- mvoprobit(list(z1 ~ s1 + s2 | s2 + s3,
                         z2 ~ s1 + s3),
                   list(y ~ s1 + s3 + s4),
                   groups = groups, groups2 = groups2, data = data)
# Use delta method to estimate standard error for each \( P(z_1 = 0, z_2 = 2) \)

```r
defineFunction(object) = 
  val <- predict(object, group = c(1, 0))

  return(val)
}

prob02 <- delta_method(object = model, fn = defineFunction)
head(prob02)
```

# Use delta method to estimate standard error for each
# \( E(y_1|z_1=0, z_2=2) - E(y_0|z_1=0, z_2=2) \)

```r
defineFunction(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 <- delta_method(object = model, fn = defineFunction)
summary(ATE)
```

# Use delta method to estimate standard error for the difference
# between beta0 and beta1 coefficients

```r
defineFunction(object) = 
  coef1 <- coef(object, regime = 1, type = "coef2")
  coef0 <- coef(object, regime = 0, type = "coef2")

  coef_difference <- coef1 - coef0

  return(coef_difference)
}

coef_val <- delta_method(object = model, fn = defineFunction)
summary(coef_val)
```
**fitted.mvoprobit**

**Arguments**

- `object` : object of class 'mvoprobit'.
- `...` : 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).

**Value**

Returns a data frame. Its first column provides an index of the most probable alternative. Columns which names coincide with the names of the continuous equation provide unconditional expectation of the dependent variable in available regimes.

---

**fitted.mvoprobit**  
**Extract Model Fitted Values**

**Description**

Extracts fitted values from 'mvoprobit' object

**Usage**

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

**Arguments**

- `object` : object of class 'mvoprobit'.
- `...` : 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).

**Value**

Returns a data frame. Its columns which names coincide with the names of the ordered equations provide an index of the most probable category. Columns which names coincide with the names of the continuous equations provide unconditional expectations of the dependent variables in available regimes.
formula.mnprobit  
Formulas of mnprobit model.

Description
Provides formulas associated with the object of class 'mnprobit'.

Usage
## S3 method for class 'mnprobit'
formula(x, ..., type = "formula")

Arguments
x          object of class 'mnprobit'.
...        further arguments (currently ignored).
type       character; if type = "formula" or type = 1 then function returns a formulas of
            multinomial equation. If type = "formula2" or type = 2 then function returns
            a formula of continuous equation.

Value
Returns a formula.

formula.mvoprobit  Formulas of mvoprobit model.

Description
Provides formulas associated with the object of class 'mvoprobit'.

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

Arguments
x          object of class 'mvoprobit'.
...        further arguments (currently ignored).
type       character; if type = "formula" or type = 1 then function returns formulas of
            ordered equations. If type = "formula2" or type = 2 then function returns for-
            mulas of continuous equations.
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.

---

**Description**

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

**Usage**

```r
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 voluntary 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**

```r
# 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**

```r
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**

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

---

**grad_mnprobit**

*Gradient of the Log-likelihood Function of Multinomial Probit Model*

**Description**

Calculates gradient of the log-likelihood function of multinomial probit model.

**Usage**

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

Arguments

par vector of parameters.
control_lnL list with some additional parameters.
out_type string represent 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.

grad_mvoprobit 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_mvoprobit(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 represent 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_mnprobit

Log-likelihood Function of Multinomial Probit Model

Description

Calculates log-likelihood function of multinomial probit model.

Usage

```
lnL_mnprobit(
  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.

lnL_mvoprobit

Log-likelihood Function of Multivariate Ordered Probit Model

Description

Calculates log-likelihood function of multivariate ordered probit model.

Usage

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>par</td>
<td>vector of parameters.</td>
</tr>
<tr>
<td>control_lnL</td>
<td>list with some additional parameters.</td>
</tr>
<tr>
<td>out_type</td>
<td>string represent the output type of the function.</td>
</tr>
<tr>
<td>n_sim</td>
<td>the number of random draws for multivariate normal probabilities.</td>
</tr>
<tr>
<td>n_cores</td>
<td>the number of cores to be used.</td>
</tr>
<tr>
<td>regularization</td>
<td>list of regularization parameters.</td>
</tr>
</tbody>
</table>

Description

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

Usage

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>object of class &quot;mnprobit&quot;</td>
</tr>
<tr>
<td>...</td>
<td>further arguments (currently ignored)</td>
</tr>
</tbody>
</table>

Details

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

Value

Returns an object of class 'logLik'.
logLik.mvoprobit  

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

**Description**

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

**Usage**

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

**Arguments**

- `object`: object of class "mvoprobit"
- `...`: further arguments (currently ignored)

**Details**

If `estimator == "2step"` in `mvoprobit` then function may return `NA` value since two-step estimator of covariance matrix may be not 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**

```r
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

```r
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)
```

Description

This function performs chi-squared test for nested models.

Usage

```r
lrtest(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.

Value

The function returns an object of class `lrtest` 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.
- 1l2 - 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 test statistic value.
- p_value - p-value of the chi-squared test.

Examples

```r
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(model1, model2)
```

**mnprobit**

Multinomial probit model

**Description**

This function estimates parameters of multinomial probit model and sample selection model with continuous outcome and multinomial probit selection mechanism.

**Usage**

```r
mnprobit(
  formula,
  formula2 = NULL,
  data,
  regimes = NULL,
  opt_type = "optim",
  opt_args = NULL,
  start = NULL,
  estimator = "ml",
  cov_type = "sandwich",
  degrees = NULL,
  n_sim = 1000,
  n_cores = 1,
  control = NULL,
  regularization = NULL
)
```
Arguments

formula  an object of class "formula" corresponding to multinomial (selection) equation.

formula2 an object of class "formula" corresponding to continuous (outcome) equation.

data data frame containing the variables in the model.

regimes numeric vector such that regimes[i] is a regime of continuous equation when
1-th alternative is observable. It should start with 0 and special value -1 under-
mines that continuous (outcome) equation is unobservable.

opt_type character representing 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.
genetic algorithm. If opt_type = "pso" then pso will be used i.e. particle
swarm optimization.

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

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

estimator character determining estimation method. If estimator = "ml" then maximum-
likelihood will be used. If estimator = "2step" then two-step estimation pro-
cedure similar to Heckman's method will be applied.

cov_type character determining the type of covariance matrix to be returned and used for
summary. 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" (default) then sandwich covariance
matrix estimator will be applied.

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

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.

n_cores positive integer representing the number of CPU cores used for parallel comput-
ing. If possible it is highly recommend to set it equal to the number of available
physical cores especially when the system of ordered equations has 2 or 3 equa-
tions.

control a list of control parameters. See 'Details'.

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

For identification purposes the following parametrization of the multinomial probit model is used:

\[ z_{ji}^* = w_i \gamma_j + u_{ji}, \quad z_{ji}^* = 0, \]
\[ i \in \{1, 2, ..., n\}, \quad j \in \{1, 2, ..., J - 1\}, \]
\[ z_i = \arg\max_{t \in \{1, ..., J\}} z_{ti}^*, \quad u_i = (u_{1i}, u_{2i}, ..., u_{(J-1)i})^T, \]
\[ u_i \sim N \left( \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, \Sigma \right), \text{i.i.d.}, \]
\[ \Sigma = \begin{bmatrix}
1 & \sigma_{12} & \sigma_{13} & \cdots & \sigma_{1(J-1)} \\
\sigma_{12} & \sigma_{2} & \sigma_{3} & \cdots & \sigma_{2(J-1)} \\
\sigma_{13} & \sigma_{3} & \sigma_{3} & \cdots & \sigma_{3(J-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\sigma_{1(J-1)} & \sigma_{2(J-1)} & \sigma_{3(J-1)} & \cdots & \sigma_{J-1}^2
\end{bmatrix}. \]

Where:

- \( J \) - the number of alternatives.
- \( z_{ji}^* \) - unobservable (latent) value of the \( j \)-th alternative. Usually \( z_{ji}^* \) is interpreted as a utility of the \( j \)-th alternative.
- \( z_i \) - selected alternative.
- \( w_i \) - regressors that should be described in formula. Regressors are assumed to be the same for all alternatives.
- \( \gamma_j \) - regression coefficients of the \( j \)-th alternative’s equation.
- \( w_i \gamma_j \) - linear index of the \( j \)-th alternative’s equation.
- \( u_i \) - multivariate normal random vector which elements are normal random variables.
- \( \Sigma \) - covariance matrix of \( u_i \).

Note that alternatives \( z_i \) should be represented in data as integers starting from 1 (not 0).

It is also possible to account for multinomial sample selection and endogenous switching. Consider a simple example. Suppose that there is a sample containing information on wages of individuals. Let’s denote wages of people who are working in information technologies (IT) sector and of those who are not by \( y_{1i} \) and \( y_{0i} \) correspondingly. The effect of various characteristics \( x_i \) on \( y_{0i} \) and \( y_{1i} \) may differ. For example programming skills probably have a greater impact on \( y_{1i} \) than on \( y_{0i} \). So there is different equations (regimes) for these wages:

\[ y_{0i} = x_i \beta_0 + \varepsilon_{0i}, y_{1i} = x_i \beta_1 + \varepsilon_{1i}, \quad (\varepsilon_{0i}, \varepsilon_{1i}), \text{i.i.d.} \]

where \( \beta_0 \) is a vector of regression coefficients representing the effect of individual characteristics \( x_i \) on wage \( y_{0i} \). Similarly for \( \beta_1 \).

Herewith there is non-random selection into IT sector. Suppose that \( z_i = 1 \) if individual is working in IT sector, \( z_i = 2 \) if individual is employed in non-IT sector, and \( z_i = 3 \) if individual is unemployed. So observable wage is:
It is assumed that joint distribution of $u_{regime[k]}$ for endogenously omitted observations. Dependent variable $y_i$ by default the model is estimated via maximum likelihood method. However if estimator = "2step" then two-step procedure proposed by E. Kossova and B. Potanin (2022) will be used. The idea is similar to Heckman’s method i.e. to estimate the following regression equation:

$$y_i = x_i \beta + \sum_{t=1}^{J-1} \rho_t \sigma_t \hat{z}_{\lambda_t}^{(zi)}$$

where:

$$\hat{\lambda}_t^{(j)} = A_t^{(j)} \lambda_t^{(i)} A_t^{(j)} = \begin{cases} 1, & \text{if } t_1 = j \\ -1, & \text{if } t_1 < j \text{ and } t_1 = t_2 \\ -1, & \text{if } t_1 > j \text{ and } t_1 = t_2 + 1 \\ 0, & \text{otherwise} \end{cases}, t_1, t_2 \leq J - 1$$

Note that $\hat{\lambda}$ are estimated on the first step using estimates from multinomial probit model. On the second step these estimates are used as the regressors (covariates) where $\beta$ and $\rho_t \sigma_t \sigma_z$ are estimated via least squares method. Standard errors are estimated by approach proposed by Newey (2009).

Argument degrees is similar to the same argument of mvoprobit.

Optimization always starts with optim. If opt_type = "gena" or opt_type = "pso" then gena or pso is used to proceed optimization starting from initial point provided by optim. Manual arguments to optim should be provided in a 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 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)).


Function pmnorm is 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.

Value

This function returns an object of class 'mnprobit' which is a list containing the following elements:

- **par** - vector of parameters' estimates.
- **coef** - matrix which j-th column coef[,] is a vector of regression coefficients estimates of the j-th alternative equation i.e. \( \hat{\gamma}_j \).
- **coef2** - matrix which j-th column coef2[,] is a vector of regression coefficients estimates of the continuous equation in (j+1)-th regime.
- **sigma** - estimate of the covariance matrix of random errors of alternatives equations i.e. \( \hat{\Sigma} \).
- **cov2** - matrix which element cov2[i, j] is an estimate of the covariance between random error of i-th alternative equation and random error of continuous equation in (j+1)-th regime.
- **var2** - a vector such that var2[i] is the estimate of the variance of the random error of continuous equation in (i+1)-the regime.
- **logLik** - log-likelihood value.
- **W** - numeric matrix of regressors of the system of multinomial equations.
- **X** - numeric matrix of regressors of continuous equation.
- **z** - numeric vector of multinomial dependent variable values.
- **y** - numeric vector of continuous variable values.
- **control$lnL** - some additional variables to be passed to likelihood function (not intended for users).
- **formula** - the same as formula input argument but all elements are coerced to formula type.
- **formula2** - the same as formula input argument but all elements are coerced to formula type.
- **lambda** - matrix such that lambda[i, j] corresponds to \( \hat{\lambda}_{ji} \).
- **data** - the same as data input argument but without missing values.
- **cov** - estimate of the covariance matrix of parameters' estimator.
- **cov_type** - type of the asymptotic covariance matrix estimator.
- **cov_2step** - estimate of the covariance matrix of parameters' estimator associated with the second step parameters i.e. when estimator = "2step".
- **tbl** - special table used to create a summary (not intended for users).
- **regimes** - the same as regimes input argument or automatically generated matrix representing the structure of the system of equations. Please, see 'Details' section above for more information.
- **n_regimes** - the number of regimes.
- **degrees** - the same as degrees input argument.
- **model1** - first step estimation results when estimator = "2step".
- **coef_lambda** - estimates of coefficients of lambdas.
- **n_alt** - the number of alternatives.
mnprobit

- n_obs - the number of observations.
- J - the Jacobian of the likelihood function.
- p_value - p-values of the tests on significance of the parameters where null hypothesis is that corresponding parameter equals zero.
- other - list of additional variables that is not intended for the user.

It is highly recommended to get estimates via coef.mnprobit function.

References


Examples

```r
# CPS data example
# -------------------------------
set.seed(123)
data(cps)
f_educ <- educ ~ age + I(age^2) + sbachelor + smaster
model1 <- mnprobit(f_educ, data = cps)
summary(model1)

# Endogenous education treatment model
f_lwage <- lwage ~ age + I(age^2) + bachelor + master + health
model2 <- mnprobit(f_educ, f_lwage, data = cps, cov_type = "gop")
summary(model2)
```

f_lwage2 <- lwage ~ age + I(age ^ 2) + health
model3 <- mnprobit(f_educ, f_lwage2, data = cps,
                      regimes = c(0, 1, 2), cov_type = "gop")
summary(model3)

# Simulated data example 1
# Multinomial probit model

# Load required package
library("mnorm")

# ---
# Step 1
# Simulation of 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)

# Random errors
sigma.1 <- 1
sigma.2 <- 0.9
rho <- 0.7
sigma <- matrix(c(sigma.1 ^ 2, sigma.1 * sigma.2 * rho,
                   sigma.1 * sigma.2 * rho, sigma.2 ^ 2),
                ncol = 2, byrow = TRUE)
u <- rmnorm(n = n, mean = c(0, 0), sigma = sigma)

# Coefficients
gamma.1 <- c(0.1, 2, 3)
gamma.2 <- c(-0.1, 3, -2)

# Linear index

# Latent variable
z1.star <- z1.li + u[, 1]
z2.star <- z2.li + u[, 2]
z3.star <- rep(0, n)

# Observable ordered outcome
z <- rep(3, n)
\[
\begin{align*}
& z[(z_1 \star > z_2 \star) \& (z_1 \star > z_3 \star)] \leftarrow 1 \\
& z[(z_2 \star > z_1 \star) \& (z_2 \star > z_3 \star)] \leftarrow 2 \\
& \text{table}(z)
\end{align*}
\]

\# Data
\begin{verbatim}
data <- data.frame(w1 = w1, w2 = w2, z = z)
\end{verbatim}

\# Step 2
\# Estimation of parameters
\begin{verbatim}
model <- mnprobit(z ~ w1 + w2, 
data = data)
summary(model)
\end{verbatim}

\# Compare estimates and true values of parameters
\begin{verbatim}
# regression coefficients
cbind(true = gamma.1, estimate = model$coef[, 1])
cbind(true = gamma.2, estimate = model$coef[, 2])
# covariances
ccbind(true = c(sigma[1, 2], sigma[2, 2]),
estimate = c(model$sigma[1, 2], model$sigma[2, 2]))
\end{verbatim}

\# Step 3
\# Estimation of probabilities and marginal effects
\begin{verbatim}
# For every observation in a sample predict
# probability of 2-nd alternative i.e. P(z = 2)
prob <- predict(model, alt = 2, type = "prob")
head(prob)
# probability of each alternative
prob <- predict(model, alt = NULL, type = "prob")
head(prob)

# Calculate mean marginal effect of w2 on P(z = 1)
mean(predict(model, alt = 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)
predict(model, alt = 2, type = "prob", newdata = newdata)
# linear index
predict(model, type = "li", newdata = newdata)
# marginal effect of w1 on P(z = 2)
predict(model, alt = 2, type = "prob", newdata = newdata, me = "w1")
\end{verbatim}
# marginal effect of w1 and w2 on P(z = 3)
predict(model, alt = 3, type = "prob", newdata = newdata, me = c("w1", "w2"))
# marginal effect of w2 on the linear index
predict(model, alt = 2, type = "li", newdata = newdata, me = "w2")
# discrete marginal effect i.e. P(z = 2 | w1 = 0.5) - P(z = 2 | w1 = 0.2)
predict(model, alt = 2, type = "prob", newdata = newdata, me = "w2", eps = c(0.2, 0.5))
# adjusted conditional expectation for endogenous switching and sample selection models with continuous outcome with random error 'e'
# E(e | z = 2) / cov(e, u)
# where joint distribution of 'e' and 'u' determined by Gaussian copula and 'e' is normally distributed
predict(model, alt = 2, type = "lambda", newdata = newdata)

# Simulated data example 2
# Multinomial selection model
#
# Load required package
library("mnorm")
#
# Step 1
# Simulation of data
#
# Set seed for reproducibility
set.seed(123)

# The number of observations
n <- 1000

# Random errors
sd.z2 <- sqrt(0.9)
cor.z <- 0.3
sd.y0 <- sqrt(2)
cor.z1y0 <- 0.3
cor.z2y0 <- 0.7
sd.y1 <- sqrt(1.8)
cor.z1y1 <- 0.7
cor.z2y1 <- 0.6
cor.y <- 0.8
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 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
gamma <- cbind(c(0.1, 1, 1),
               c(0.2, -1.2, 0.8))
beta <- cbind(c(1, -1, 2),
              c(1, -2, 1))

# Linear indexes
z1.li <- W %*% gamma[, 1]
z2.li <- W %*% gamma[, 2]
y0.li <- X %*% beta[, 1]
y1.li <- X %*% beta[, 2]

# 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]

# Obvservable 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)

# Aggregate observable variable
z <- rep(0, n)
z[z1] <- 1
z[z2] <- 2
z[z3] <- 3
table(z)

# Make unobservable values of continuous variable
y <- rep(Inf, 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 parameters
# ---

# Maximum likelihood method
model.ml <- mnprobit(z ~ x1 + x2,
y ~ x1 + x3, regimes = c(0, 1, -1),
data = data, cov_type = "gop")
summary(model.ml)

# Two-step method
model.2step <- mnprobit(z ~ x1 + x2,
y ~ x1 + x3, regimes = c(0, 1, -1),
data = data, estimator = "2step")
summary(model.2step)

# Semiparametric estimator using 2-nd and 3-rd level polynomials
model.snp <- mnprobit(z ~ x1 + x2,
y ~ x1 + x3, regimes = c(0, 1, -1),
data = data, estimator = "2step",
degrees = c(2, 3))
summary(model.snp)

# Simple least squares as a benchmark
model.lm0 <- lm(y ~ x1 + x3, data = data[z == 1, ])
model.lm1 <- lm(y ~ x1 + x3, data = data[z == 2, ])

# Compare coefficients of continuous equations
# y0
cbind(true = beta[, 1],
ml = model.ml$coef2[, 1],
twostep = model.2step$coef2[, 1],
semiparametric = model.snp$coef2[, 1],
ls = coef(model.lm0))

# y1
cbind(true = beta[, 2],
ml = model.ml$coef2[, 2],
twostep = model.2step$coef2[, 2],
semiparametric = model.snp$coef2[, 2],
ls = coef(model.lm1))

# Compare coefficients of multinomial equations
# 1-nd alternative
cbind(true = gamma[, 1],
ml = model.ml$coef[, 1],
twostep = model.2step$coef[, 1])

# 2-nd alternative
cbind(true = gamma[, 2],
ml = model.ml$coef[, 2],
twostep = model.2step$coef[, 2])

# Compare variances of random errors associated with
# z2
cbind(true = sigma[2, 2], ml = model.ml$sigma[2, 2])
# y0
cbind(true = sd.y0 ^ 2, ml = model.ml$var2[1])
# y1
cbind(true = sd.y1 ^ 2, ml = model.ml$var2[2])

# compare covariances between
# z1 and z2
cbind(true = cor.z * sd.z2,
    ml = model.ml$sigma[1, 2],
    twostep = model.2step$sigma[1, 2])
# z1 and y0
cbind(true = cor.z1y0 * sd.y0,
    ml = model.ml$cov2[1, 1],
    twostep = model.2step$cov2[1, 1])
# z2 and y0
cbind(true = cor.z2y0 * sd.y0, ml = model.ml$cov2[2, 1])
# z1 and y1
cbind(true = cor.z1y1 * sd.y1, ml = model.ml$cov2[1, 2])
# z2 and y1
cbind(true = cor.z2y1 * sd.y1, ml = model.ml$cov2[2, 2])

# ---
# Step 3
# Predictions and marginal effects
# ---

# Unconditional expectation E(y1) for every observation in a sample
predict(model.ml, type = "val", regime = 1, alt = NULL)

# Marginal effect of x1 on conditional expectation E(y0|z = 2)
# for every observation in a sample
predict(model.ml, type = "val", regime = 0, alt = 2, me = "x1")

# Calculate predictions and marginal effects
# for manually provided observations
# using abovementioned 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)
predict(model.ml, type = "val", regime = 0, alt = NULL, newdata = newdata)
predict(model.2step, type = "val", regime = 0, alt = NULL, newdata = newdata)
predict(model.snp, type = "val", regime = 0, alt = NULL, newdata = newdata)

# Conditional expectation E(y1|z=3)
predict(model.ml, type = "val", regime = 1, alt = 3, newdata = newdata)
predict(model.2step, type = "val", regime = 1, alt = 3, newdata = newdata)
predict(model.snp, type = "val", regime = 1, alt = 3, newdata = newdata)
mvoprobit

Multivariate ordered probit model with heteroscedasticity and (non-random) sample selection.

Description

This function allows to estimate parameters of multivariate ordered probit model and its extensions. It is possible to account for heteroscedastic variances, non-normal marginal distributions of random errors (under Gaussian copula) and (non-random) sample selection i.e. when some categories of particular dependent variables are observable only under some specific values of other dependent variables. Also it is possible to include continuous equations to get multivariate generalization of endogenous switching model. In this case both maximum-likelihood and two-step (similar to Heckman’s method) estimation procedures are implemented.

Usage

mvoprobit(
  formula,
  formula2 = NULL,
  data = NULL,
  groups = NULL,
  groups2 = NULL,
  marginal = list(),
  opt_type = "optim",
  opt_args = NULL,
  start = NULL,
  estimator = "ml",
  cov_type = ifelse(estimator == "ml", "sandwich", "parametric"),
  degrees = NULL,
  n_sim = 1000,
  n_cores = 1,
  control = NULL,
  regularization = NULL
)
### Arguments

- **formula**: list which i-th element is an object of class "formula" describing the form of the linear index for the i-th ordered equation. Mean and variance equations should be separated by '|' symbol.

- **formula2**: list which i-th element is an object of class "formula" describing the form of the linear index for the i-th continuous equation.

- **data**: data frame containing the variables in the model.

- **groups**: matrix which (i, j)-th element is j-th ordered category (value starting from 0) of i-th dependent ordered variable. Each row of this matrix describes observable (in data) combination of categories i.e. values of dependent variables. Special category '-1' means that variable in 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 groups argument but for the continuous dependent variables from formula2. See 'Details' for additional information.

- **marginal**: list such that marginal[[i]] represents parameters of 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**: character representing 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. genetic algorithm. If opt_type = "pso" then pso will be used i.e. particle swarm optimization.

- **opt_args**: a list of input arguments for the optimization function selected via opt_type argument. See 'Details' for information.

- **start**: numeric vector of initial parameters’ values. It will be used as a starting point for optimization purposes. It is also possible to provide an object of class 'mvoprobit' then its 'par' element will be used as a starting point.

- **estimator**: 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.

- **cov_type**: character determining the type of covariance matrix to be returned and used for summary. 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" (default) then sandwich covariance matrix estimator will be applied. Second, suppose that estimator = "2step" then by default sandwich estimator will be used for the first step parameters and the following estimators are available for the second step parameters. If cov_type = "parametric" then parametric estimator will be used on the second step which assumes joint normality of random errors. If cov_type = "nonparametric" then nonparametric estimator will be used. Also cov_type may be a character vector such that cov_type[i] determines the covariance matrix estimator of the i-th step parameters.

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

n_cores  
positive integer representing the number of CPU cores used for parallel computing. If possible it is highly recommend to set it equal to the number of available physical cores especially when the system of ordered equations has 2 or 3 equations.

control  
a list of control parameters. See 'Details'.

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

Details
Multivariate ordered probit model with heteroscedastic random errors has the following form:

\[ z_{ji}^* = w_{ji} \gamma_j + \sigma_{ji} u_{ji}, \]
\[ \sigma_{ji} = \exp(w_{ji}^* \gamma_j^*), \quad u_i \sim N \left( \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, \Sigma \right), \text{i.i.d.}, \]

\[ \Sigma = \begin{bmatrix}
  1 & \rho_{12} & \rho_{13} & \cdots & \rho_{1J} \\
  \rho_{12} & 1 & \rho_{23} & \cdots & \rho_{2J} \\
  \rho_{13} & \rho_{23} & 1 & \cdots & \rho_{3J} \\
  \vdots & \vdots & \vdots & \ddots & \vdots \\
  \rho_{1J} & \rho_{2J} & \rho_{3J} & \cdots & 1
\end{bmatrix}, \]

\[ z_{ji} = \begin{cases}
  0, & \text{if } z_{ji}^* \leq c_{j1} \\
  1, & \text{if } c_{j1} < z_{ji}^* \leq c_{j2} \\
  2, & \text{if } c_{j2} < z_{ji}^* \leq c_{j3} \\
  \vdots & \vdots \\
  m_j, & \text{if } z_{ji}^* > c_{jm_j}
\end{cases}, \]

\[ z_i = (z_{1i}, ..., z_{Ji})^T, \quad u_i = (u_{i1}, u_{i2}, ..., u_{ij})^T, \]

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

Where:
- \( n \) - the number of observations. If there are no omitted observations then \( n \) equals to nrow(data).
- \( J \) - the number of equations i.e. length(formula).
• $z_{ji}^*$ - unobservable (latent) value of the $j$-th dependent variable.
• $z_{ji}$ - observable (ordered) value of the $j$-th dependent variable.
• $(m_j + 1)$ - the number of categories of $z_{ji} \in \{0, 1, ..., m_j\}$.
• $c_{jk}$ - $k$-th cut of the $j$-th dependent variable.
• $w_{ji}$ - regressors of the $j$-th mean equation which should be described in formula[[j]].
• $\gamma_j$ - regression coefficients of the $j$-th mean equation.
• $w_{ji}\gamma_j$ - linear index of the $j$-th mean equation.
• $\sigma_{ji}$ - heteroscedastic standard deviation.
• $\sigma_{ji}u_{ji}$ - heteroscedastic random errors.
• $w_{ji}^*$ - regressors of the $j$-th variance equation which should be described in formula[[j]] after ’|’ symbol.
• $\gamma^*_j$ - regression coefficients of the $j$-th variance equation.
• $w_{ji}\gamma^*_j$ - linear index of the $j$-th variance equation.

Parameters of this model are estimated via maximum-likelihood method using numeric optimization approach provided through opt_type argument. The type of covariance matrix estimator may be provided through cov_type argument.

To account for (non-random) sample selection unobservable values of dependent variables should be coded as -1. For example if $z_1$ is a binary variable for employment status (0 - unemployed, 1 - employed) and $z_2$ is ordered variable (ranging from 0 to 2) for job satisfaction (0 - unsatisfied, 1 - satisfied, 2 - highly satisfied) then $z_2$ is observable only when $z_1$ equals 1 since job satisfaction observable only for working individuals. Consequently $z_2$ should be equal to -1 (minus one) whenever $z_1$ equals to 0. If variables are coded in this way then groups matrix will be created automatically. Otherwise user may provide manual structure of selection mechanism by mentioning all possible combinations of $z_1$ and $z_2$ values as a rows of groups matrix. In this particular example matrix groups will have the following form (no need to provide it manually):

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

Again, please, insure that all $z_2$ equal to -1 for all $z_1$ which equal to 0 in your data. Then matrix groups will automatically have the aforementioned structure (accounting for non-random sample selection).

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

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 equation. If all $z_{ji}$ are binary variables then the model becomes multivariate probit.
It is possible to estimate sample-selection and endogenous switching models with continuous dependent variables by providing the form of corresponding equations via `formula2` argument. Selection (switching) mechanism will be determined by the aforementioned multivariate ordered probit model.

First, consider sample selection model with one continuous equation \( y - \text{wage} \) and two ordered equations from the previous example \( z_1 - \text{employment}, z_2 - \text{job satisfaction} \):

\[
y^*_i = x_i \beta + \varepsilon_i,
\]

\[
y_i = \begin{cases} 
y^*_i, & \text{if } z_{1i} = 1 \text{ and } z_{2i} \geq 1 \\
\text{unobservable, otherwise} & 
\end{cases},
\]

where \( y_i \) and \( x_i \) are continuous dependent variable and the vector of exogenous variables correspondingly. These variables should be described in `formula2`. Random errors \( \varepsilon_i \) and \( u_i \) are multivariate normal. In this example it is assumed that information on wages \( y_i \) available only for employed \( z_{1i} = 1 \) individuals with high enough job satisfaction \( z_{2i} \geq 1 \) (suppose that unsatisfied workers where not asked wage question or surely refuse to answer).

To estimate this model it is also necessarily to set unobservable values of \( y_i \) in data to \( \text{Inf} \) to distinguish them from \( \text{NA} \) values representing observations omitted by random. Finally one needs to manually specify the structure of equations via `groups` and `groups2` arguments by providing all possible combinations of ordered and continuous equations values:

\[
groups = \begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad groups2 = \begin{bmatrix} 0 \\ 0 \\ -1 \\ -1 \end{bmatrix}.
\]

Where 0 category in `group2` indicates that continuous dependent variable \( y_i \) is observable. For example `groups2[2] = 0` indicates that \( y_i \) is observable when `groups[2,] = c(1, 1)` i.e. \( z_{1i} = 1 \) and \( z_{2i} = 1 \).

Further suppose that we assume that wage equations are different for satisfied \( (z_{2i} = 1) \) and highly satisfied \( (z_{2i} = 2) \) workers:

\[
y^*_{0i} = x_i \beta_0 + \varepsilon_{0i},
\]

\[
y^*_{1i} = x_i \beta_1 + \varepsilon_{1i},
\]

\[
y_i = \begin{cases} 
y^*_0, & \text{if } z_{1i} = 1 \text{ and } z_{2i} = 1 \\
y^*_1, & \text{if } z_{1i} = 1 \text{ and } z_{2i} = 2 \\
\text{unobservable, otherwise} & 
\end{cases}.
\]

To estimate this endogenous switching model arguments `groups` and `groups2` should be specified as follows:

\[
groups = \begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad groups2 = \begin{bmatrix} 1 \\ 0 \\ -1 \\ -1 \end{bmatrix}.
\]
For example, 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. 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 regime 0 corresponding to the wage of satisfied workers.

Therefore by specifying groups and groups2 arguments in the aforementioned way it is possible to estimate various sample selection and endogenous switching models. Furthermore one may specify several continuous equations. Indeed, consider additional continuous equation (\( y^h \) - working hours):

\[
y^h_i = x_i^h \beta^h + \varepsilon^h_i, \quad y^*_h = \begin{cases} y^h_i, & \text{if } z_{1i} = 1 \\ \text{unobservable, otherwise} \end{cases}
\]

Then to estimate the system accounting for this additional continuous equation simply substitute all \( y^*_h \) (such that \( z_{1i} = 0 \)) in data with \( \text{Inf} \) and specify:

\[
\text{groups} = \begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad \text{groups2} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ -1 & 0 \\ -1 & -1 \end{bmatrix},
\]

where \( \text{groups2}[1, ] \) describes regimes of the wage equation \( y_i \) while \( \text{groups}[2, ] \) contains regimes of the hours equation \( y^h_i \). Note that formula of the first equation (wage) should be specified in formula2[[1]] and 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.

By default all the models are estimated via maximum likelihood method. However if estimator = "2step" then models with one continuous equation (but voluntary number of regimes of this equation) will be estimated via two-step procedure proposed by E. Kossova and B. Potanin (2018). The idea is similar to classical Heckman’s method i.e. to substitute conditional expectation of random error into continuous equation with it’s consistent estimator. For simplicity suppose that there is only one regime. Then regression equation may be represented in the following form:

\[
y_i = x_i \beta + \sum_{j=1}^{J} \rho_j \sigma \lambda_{ji} + \varepsilon_i^*, \quad \varepsilon_i^* = \varepsilon_i - E(\varepsilon_i | z_{1i}, ..., z_{ji}) = \varepsilon_i - \sum_{j=1}^{J} \rho_j \sigma \lambda_{ji},
\]

where:

\[
\lambda_{ji} = \lambda_{ji}^{(1)} + \lambda_{ji}^{(2)},
\]

\[
\lambda_{ji}^{(1)} = \begin{cases} 0, & \text{if } z_{ji} = 0 \\ -\partial \ln P_i^* / \partial a_{ji}, & \text{otherwise} \end{cases}, \quad \lambda_{ji}^{(2)} = \begin{cases} 0, & \text{if } z_{ji} = m_j \\ -\partial \ln P_i^* / \partial b_{ji}, & \text{otherwise} \end{cases},
\]

\[
P_i^*(a_{1i}, ..., a_{ji}; b_{1i}, ..., b_{ji}) = P(a_{1i} \leq u_{1i} \leq b_{1i}, ..., a_{ji} \leq u_{ji} \leq b_{ji}),
\]
On the first step $\hat{\lambda}_{ji}$ are calculated using estimates of multivariate ordered probit model. On the second step $\hat{\lambda}_{ji}$ are used as the regressors instead of $\lambda_{ji}$ in a least squares estimation of $y_i$ equation. If `cov_type = "parametric"` then asymptotic covariance matrix estimator proposed by E. Kossova and B. Potanin (2018) is used. If `cov_type = "nonparametric"` then robust covariance matrix estimator of Newey (2009) is applied. To relax normality assumption of $\varepsilon_i$ it is possible to use multivariate generalization of Newey (2009) method described in E. Kossova and B. Potanin (2020). The idea is to use polynomials of $\lambda_{ji}$ on the second step:

$$y_i = x_i \beta + \sum_{j=1}^{J} \sum_{t=1}^{d_j} \tau_{jt} \lambda_{ji}^t + \varepsilon_i,$$

where $\tau_{jt}$ are polynomial coefficients. Polynomial order $d_j$ is determined by `degrees[j]` value. If there are more than one regime then degrees should be a matrix such that `degrees[r, j]` is $d_j$ corresponding to the $r$-th regime. However if there are more than one regime and `degrees` is a vector it will be transformed into a matrix which rows are the same as degrees.

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

Function `pmnorm` is used internally for calculation of multivariate normal probabilities, densities and their derivatives. Marginal distribution of $u_i$ may be determined with `marginal` argument that is similar to the same argument in `pmnorm`. Note that joint distribution of $u$ (random errors of ordered equations) and $\varepsilon$ (random errors of continuous equations) will be determined by Gaussian copula.

Optimization always starts with `optim`. If `opt_type = "gena"` or `opt_type = "pso"` then `gena` or `pso` is used to proceed optimization starting from initial point provided by `optim`. Manual arguments to `optim` should be provided in a 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 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))`.


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

**Value**

This function returns an object of class 'mvoprobit' which is a list containing the following elements:

- `par` - vector of parameters’ estimates.
• coef - list which j-the element coef[[j]] is a vector of regression coefficients estimates of the j-th ordered equation i.e. \( \hat{\gamma}_j \).
• coef_var - list which j-the element coef_var[[j]] is a vector of regression coefficients estimates of the variance part of the j-th ordered equation i.e. \( \hat{\gamma}^*_j \).
• coef2 - list which j-the element coef2[[j]] is a matrix of regression coefficients estimates of the j-th continuous equation. Wherein i-th row of this matrix contains estimates of regression coefficients corresponding to the i-th regime of j-th continuous variable.
• sigma - estimate of the covariance matrix of random errors of ordered equations i.e. \( \hat{\Sigma} \).
• var2 - estimates of the variances of random errors of continuous equations.
• sigma2 - estimates of covariances between random errors of continuous equations.
• cov2 - list which j-the element cov_y[[j]] contains estimates of covariances between random errors of j-th continuous equation in different regimes.
• cuts - list which j-the element cuts[[j]] is a vector of cuts estimates of the j-th equation i.e. \( \hat{c}_j \).
• ind - list containing some indexes partition of the model (not intended for users).
• logLik - log-likelihood value.
• regressors - numeric matrix which j-the element regressors[[j]] is a regressors matrix of the j-th equation i.e. \( w_j \).
• regressors2 - list which j-the element regressors2[[j]] is a regressors matrix of the j-th variance equation i.e. \( w^*_j \).
• dependent - numeric matrix which j-the column dependent[, j] is a vector of dependent variable \( z_j \) values.
• control_lnL - some additional variables to be passed to likelihood function (not intended for users).
• formula - the same as formula input argument but all elements are coerced to formula type.
• lambda - matrix such that lambda[i, j] corresponds to \( \hat{\lambda}_{ij} \). predict.mvoprobit for more information.
• data_list - list which j-the element data_list[[j]] is a dataframe containing regressors and dependent variable of the j-th equation.
• data - the same as data input argument but without missing values.
• cov - estimate of the covariance matrix of parameters’ estimator.
• cov_type - type of the asymptotic covariance matrix estimator.
• cov_2step - estimate of the covariance matrix of parameters’ estimator associated with the second step parameters i.e. when estimator = "2step".
• sd - standard errors of the estimates.
• p_value - p-values of the tests on significance of the parameters where null hypothesis is that corresponding parameter equals zero.
• tbl - special table used to create a summary (not intended for users).
• 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.
• marginal - the same as marginal input argument.
• degrees - the same as degrees input argument.
• model1 - first step estimation results when estimator = "2step".
• coef_lambda - estimates of coefficients of lambdas.
• other - list of additional variables not intended for the user.

It is highly recommended to get estimates via `coef.mvoprobit` function.

References


Examples

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

# Set seed for reproducibility
set.seed(123)

# Upload data
data(cps)

# Prepare ordered 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 <- mvoprobit(f_work, data = cps)
summary(model1)

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

# Labor supply with endogenous education
# treatment (recursive or hierarchical ordered probit) model
model3 <- mvoprobit(list(f_work, f_educ), data = cps)
summary(model3)

# Sample selection (on employment) Heckman's model
f_lwage <- lwage ~ age + I(age ^ 2) + bachelor + master + health
cps$lwage[cps$work == 0] <- Inf
model4 <- mvoprobit(f_work, f_lwage, data = cps)
summary(model4)

# Endogenous education treatment with non-random sample selection
model5 <- mvoprobit(list(f_work, f_educ), f_lwage, data = cps)
summary(model5)

# Endogenous switching model with non-random sample selection
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
model6 <- mvoprobit(list(f_work, f_educ), f_lwage2,
groups = groups, groups2 = groups2,
data = cps)
summary(model6)

# -------------------------------
# Simulated data example 1
# Ordered probit model
# -------------------------------
# ---
# Step 1
# Simulation of 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
```r
u <- rnorm(n = n, mean = 0, sd = 1)

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

# Linear index

# 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.frame(w1 = w1, w2 = w2, z = z)

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

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

# Compare estimates and true values of parameters
# regression coefficients
cbind(true = gamma, estimate = model$coef[[1]])

# cuts
cbind(true = cuts, estimate = model$cuts[[1]])

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

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

# Calculate mean marginal effect of w2 on P(z = 1)
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 \( P(z = 2) \)
predict(model, group = 2, type = "prob", newdata = newdata)

# linear index
predict(model, type = "li", newdata = newdata)

# marginal effect of \( w_1 \) on \( P(z = 2) \)
predict(model, group = 2, type = "prob", newdata = newdata, me = "w1")

# marginal effect of \( w_1 \) and \( w_2 \) on \( P(z = 3) \)
predict(model, group = 3, type = "prob",  
         newdata = newdata, me = c("w1", "w2"))

# marginal effect of \( w_2 \) on the linear index
predict(model, group = 2, type = "li", newdata = newdata, me = "w2")

# discrete marginal effect i.e. \( P(z = 2 \mid w_1 = 0.5) - P(z = 2 \mid w_1 = 0.2) \)
predict(model, group = 2, type = "prob", newdata = newdata,  
         me = "w2", eps = c(0.2, 0.5))

# adjusted conditional expectation for endogenous switching and
# sample selection models with continuous outcome with random error 'e'
# \( E(e \mid z = 2) / \text{cov}(e, u) \)
# where joint distribution of 'e' and 'u' determined by
# Gaussian copula and 'e' is normally distributed
predict(model, group = 2, type = "lambda", newdata = newdata)

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

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

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

# Calculation of probabilities and marginal effects is identical
# to the previous example
# probability \( P(z = 1) \)
predict(logit, group = 1, type = "prob", newdata = newdata)
# marginal effect of w2 on P(z = 1)
predict(logit, group = 1, type = "prob", newdata = newdata, me = "w2")
# E(e | z == 1) / cov(e, u)
predict(logit, group = 1, type = "lambda", newdata = newdata)

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

g <- mvoprobit(z ~ w1 + w2,
data = data,
marginal = list("PGN" = 3))
summary(pgn)

# Calculation of probabilities and marginal effects is identical
# to the previous example
# probability P(z = 3)
predict(pgn, group = 3, type = "prob", newdata = newdata)
# marginal effect of w2 on P(z = 3)
predict(pgn, group = 3, type = "prob", newdata = newdata, me = "w2")
# E(e | z == 3) / cov(e, u)
predict(pgn, group = 3, type = "lambda", newdata = newdata)

# Test normality assumption via likelihood ratio test
lrtest(model, pgn)

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

# Load required package
library("mnorm")

# ---
# Step 1
# Simulation of 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 mean equation
gamma <- c(-1, 2)

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

# Linear index of mean equation

# Linear index of variance equation

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

# Latent variable
z_star <- li + u * sd_het

# 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

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

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

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

summary(model)

# Compare estimates and true values of parameters
# regression coefficients of mean equation
cbind(true = gamma, estimate = model$coef[[1]])
# regression coefficients of variance equation

cbind(true = gamma_het, estimate = model$coef_var[[1]])
# cuts
cbind(true = cuts, estimate = model$cuts[[1]])
# Test for homoscedasticity
model0 <- mvoprobit(z ~ w1 + w2, data = data)
lrtest(model, model0)

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

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

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

# Calculate corresponding probability, linear index 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)
predict(model, group = 2, type = "prob", newdata = newdata)
# linear index
predict(model, type = "li", newdata = newdata)
# standard deviation
predict(model, type = "sd", newdata = newdata)
# marginal effect of w3 on P(Z = 3)
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 i.e. P(Z = 2 | w1 = 0.5) - P(Z = 2 | w1 = 0.2)
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 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 variance equation
gamma2_het <- c(0.5, -1)

# Linear indexes

# Linear index of variance equation

# Heteroscedastic standard deviation
# i.e. value of 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 ordered 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.frame(w1 = w1, w2 = w2,
                   w3 = w3, w4 = w4,
                   z1 = z1, z2 = z2)

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

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

# Compare estimates and true values of parameters
# regression coefficients of the first equation
cbind(true = gamma1, estimate = model$coef[[1]])
# regression coefficients of the second equation
cbind(true = gamma2, estimate = model$coef[[2]])
# cuts of the first equation
cbind(true = cuts1, estimate = model$cuts[[1]])
# cuts of the second equation
cbind(true = cuts2, estimate = model$cuts[[2]])
# correlation coefficients
cbind(true = rho, estimate = model$sigma[1, 2])
# regression coefficients of variance equation
cbind(true = gamma2_het, estimate = model$coef_var[[2]])

# ---
# Step 3
# Estimation of probabilities and linear indexes
# ---

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

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

# Calculate corresponding probability and linear
# index for 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)
predict(model, group = c(2, 0), type = "prob", newdata = newdata)
# linear index
predict(model, type = "li", newdata = newdata)
# marginal probability P(z2 = 1)
predict(model, group = c(-1, 1), type = "prob", newdata = newdata)
# marginal probability P(z1 = 3)
predict(model, group = c(3, -1), type = "prob", newdata = newdata)
# conditional probability P(z1 = 2 | z2 = 0)
predict(model, group = c(2, 0), given_ind = 2,
    type = "prob", newdata = newdata)
# conditional probability P(z2 = 1 | z1 = 3)
predict(model, group = c(3, 1), given_ind = 1,
    type = "prob", newdata = newdata)
# marginal effect of w4 on P(Z2 = 2)
predict(model, group = c(-1, 2),
    type = "prob", newdata = newdata, me = "w4")
# marginal effect of w4 on P(z1 = 3, Z2 = 2)
predict(model, group = c(3, 2),
    type = "prob", newdata = newdata, me = "w4")
# marginal effect of w4 on P(z1 = 3 | z2 = 2)
predict(model, group = c(3, 2), given_ind = 2,
    type = "prob", newdata = newdata, me = "w4")

# ---
# Step 4
# Replication under 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 estimation procedure
model <- mvoprobit(list(z1 ~ w1 + w2,
    z2 ~ w2 + w3 | w2 + w4),
    cov_type = "GOP",
    data = data)
summary(model)
# Compare estimates and true values of parameters
# regression coefficients of the first equation
cbind(true = gamma1, estimate = model$coef[[1]])
# regression coefficients of the second equation
cbind(true = gamma2, estimate = model$coef[[2]])
# cuts of the first equation
cbind(true = cuts1, estimate = model$cuts[[1]])
# cuts of the second equation
cbind(true = cuts2, estimate = model$cuts[[2]])
# correlation coefficients
cbind(true = rho, estimate = model$sigma[1, 2])
# regression coefficients of variance equation
cbind(true = gamma2_het, estimate = model$coef_var[[2]])

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

# Estimate the model
model <- mvoprobit(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
# ordered selection mechanism
# -------------------------------

# Load required package
library("mnorm")

# ---
# Step 1
# Simulation of 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 index

# 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 'Inf'
y <- y_star
y[z <= 1] <- Inf

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

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

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

# Compare estimates and true values of parameters
# regression coefficients of ordered equation
cbind(true = gamma, estimate = model$coef[[1]])
# cuts
cbind(true = cuts, estimate = model$cuts[[1]])
# regression coefficients of continuous equation
cbind(true = beta, estimate = as.numeric(model$coef2[[1]]))
# variance
cbind(true = var.y, estimate = as.numeric(model$var2[[1]]))
# covariance
cbind(true = rho * sd.y, estimate = as.numeric(model$cov2[[1]]))

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

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

# Predict unconditional expectation of the dependent variable
predict(model, group2 = 0, newdata = newdata)

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

# ---
# Step 4
# Classical Heckman's two-step estimation procedure
# ---

# Predict adjusted conditional expectations
lambda2 <- predict(model, group = 2, type = "lambda")
lambda3 <- predict(model, group = 3, type = "lambda")

# Construct variable responsible for adjusted conditional expectation in linear regression equation
data$lambda <- NA
data$lambda[model$data$z == 2] <- lambda2[model$data$z == 2]
data$lambda[model$data$z == 3] <- lambda3[model$data$z == 3]

# Alternatively simply get this variable from the estimation output
# of a selection part of the model
model_probit <- mvoprobit(z ~ w1 + w2, data = data)
data$lambda <- model_probit$lambda

# Estimate model via classical least squares
model_lm <- lm(y ~ w1 + w3, data = data[!is.infinite(data$y), ])
summary(model_lm)

# Estimate model via two-step procedure
model_ts <- lm(y ~ w1 + w3 + lambda, data = data[!is.infinite(data$y), ])
summary(model_ts)

# Automatic estimation of two-step model with robust standard errors
model_ts <- mvoprobit(z ~ w1 + w2, y ~ w1 + w3, data = data, estimator = "2step")
summary(model_ts)

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

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

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

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

# To additionally relax normality assumption of
# random error of continuous equation it is possible
# to use Newey’s two-step procedure.
model_Newey <- mvoprobit(z ~ w1 + w2,
```r
y ~ w1 + w3,
data = data,
marginal = list(PGN = 3),
estimator = "2step",
degrees = 2,
cov_type = "nonparametric")

summary(model_Newey)
```

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

# Load required package
library("mnorm")

# ---
# Step 1
# Simulation of 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 index of ordered equation
# mean
# variance

# Linear index of continuous equation
# regime 0
# regime 1

# 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 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
# in regime 1 when 'z == 1',
# in regime 0 when 'z <= 1',
# unobservable when 'z == 0'
y <- rep(NA, n)
y[z == 0] <- Inf
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 parameters
# Assign groups
groups <- matrix(0:3, ncol = 1)
groups2 <- matrix(c(-1, 0, 1, 1), ncol = 1)

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

summary(model)

# Compare estimates and true values of parameters
# regression coefficients of ordered equation
cbind(true = gamma, estimate = model$coef[[1]])

# cuts
cbind(true = cuts, estimate = model$cuts[[1]])

# regression coefficients of continuous equation
cbind(true = beta_0, estimate = model$coef2[[1]][1, ])

# variances
cbind(true = c(var2_0, var2_1), estimate = model$var2[[1]])

# covariances
cbind(true = c(cov2_z_0, cov2_z_1), estimate = model$cov2[[1]])

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

# Load required package
library("mnorm")

# ---
# Step 1
# Simulation of 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 of ordered equation
# mean

# variance
li1_het <- gamma1_het[1] * w2 + gamma1_het[2] * w3

# Linear index of continuous equation
# regime 0

# regime 1

# 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(-1, 1)

# 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] <- Inf

data <- data.frame(w1 = w1, w2 = w2, w3 = w3, w4 = w4,
                   z1 = z1, z2 = z2, y = y)
# Step 2
# Estimation of parameters

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

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

# Compare estimates and true values of parameters
  # regression coefficients of the first ordered equation
  cbind(true = gamma1, estimate = model$coef[[1]])
  cbind(true = gamma1_het, estimate = model$coef_var[[1]])
  # regression coefficients of the second ordered equation
  cbind(true = gamma2, estimate = model$coef[[2]])
  # cuts
  cbind(true = cuts1, estimate = model$cuts[[1]])
  cbind(true = cuts2, estimate = model$cuts[[2]])
  # regression coefficients of continuous equation
  cbind(true = beta0, estimate = model$coef2[[1]][1, ])
  cbind(true = beta1, estimate = model$coef2[[1]][2, ])
  # variances
  cbind(true = c(var20, var21), estimate = model$var2[[1]])
  # covariances
  cbind(true = c(cov20_z1, cov20_z2), estimate = model$cov2[[1]][1, ])
  cbind(true = c(cov21_z1, cov21_z2), estimate = model$cov2[[1]][2, ])

# Step 3
# Estimation of expectations and marginal effects

# New data
newdata <- data.frame(z1 = 1,
                      z2 = 1,
                      y = 1,
                      w1 = 0.1,
                      w2 = 0.2,
                      w3 = 0.3,
# Predict unconditional expectation of the dependent variable
# regime 0
predict(model, group2 = 0, newdata = newdata)
# regime 1
predict(model, group2 = 1, newdata = newdata)

# Predict conditional expectations of the dependent variable
# \( E(y_1 \mid z_1 = 2, z_2 = 1) \)
predict(model, group = c(2, 1), group2 = 1, newdata = newdata)

# Marginal effect of \( w_3 \) on \( E(y_1 \mid z_1 = 2, z_2 = 1) \)
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 least squares
# Estimate model via classical least squares for a benchmark
model.ls.0 <- lm(y ~ w1 + w3 + w4,
      data = data[!is.infinite(data$y) & (data$z1 == 1), ])
model.ls.1 <- lm(y ~ w1 + w3 + w4,
      data = data[!is.infinite(data$y) & (data$z1 != 1), ])

# Apply two-step procedure
model_ts <- mvoprobit(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 two-step procedure with logistic marginal distributions
# that is multivariate generalization of Lee's method
model_Lee <- mvoprobit(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 multivariate generalization of Newey's method
model_Newey <- mvoprobit(list(z1 ~ w1 + w2 | w2 + w3,
      z2 ~ w1 + w3),
      y ~ w1 + w3 + w4,
      marginal = list(logistic = NULL, logistic = NULL),
      data = data)
```

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

# beta1
tbl <- cbind(true = beta1,
    ls = coef(model.ls.1),
    ml = model$coef2[[1]][2, ],
    twostep = model_ts$coef2[[1]][2, ],
    Lee = model_Lee$coef2[[1]][2, ],
    Newey = model_Newey$coef2[[1]][2, ])
print(tbl)

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

# Load required package
library("mnorm")

# ---
# Step 1
# Simulation of 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
var20 <- 0.9
var21 <- 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, var20, var21, 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 index of ordered equation
# mean

# variance
li1_het <- gamma1_het[1] * w2 + gamma1_het[2] * w3

# Linear index of the first continuous equation
# regime 0
# regime 1

# Linear 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

# 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] <- Inf

# 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 parameters
# ---

# Assign groups
groups <- matrix(c(0, 0,
mvoprobit

\[
\begin{align*}
0, & 1, \\
1, & 0, \\
1, & 1, \\
2, & 0, \\
2, & 1), \\
\text{byrow} = \text{TRUE}, \text{ncol} = 2)
\end{align*}
\]

\[
groups2 \leftarrow \text{matrix}(\text{NA}, \text{nrow} = \text{nrow(groups)}, \text{ncol} = 2)
\]

\[
groups2[\text{groups}[1] == 1, 1] \leftarrow 0
\]

\[
groups2[\text{groups}[1] == 0 | (\text{groups}[1] == 2), 1] \leftarrow 1
\]

\[
groups2[\text{groups}[1] == \text{groups}[2], 2] \leftarrow 0
\]

\[
groups2[\text{groups}[1] > \text{groups}[2], 2] \leftarrow 1
\]

\[
groups2[\text{groups}[1] < \text{groups}[2], 2] \leftarrow 2
\]

\[
\text{cbind(groups, groups2)}
\]

# Estimation

\[
\text{model} \leftarrow \text{mvoprobit(list}(z1 \sim w1 + w2 \mid w2 + w3,
   z2 \sim w1 + w3),
   \text{list}(y \sim w1 + w3 + w4,
   g \sim w2 + w3 + w5),
   \text{groups} = \text{groups, groups2} = \text{groups2,}
   \text{data} = \text{data})
\]

\[
\text{summary(model)}
\]

# Compare estimates and true values of parameters

# regression coefficients of the first ordered equation
\[
\text{cbind(true} = \text{gamma1, estimate} = \text{model$coef[[1]])}
\]

# regression coefficients of the second ordered equation
\[
\text{cbind(true} = \text{gamma2, estimate} = \text{model$coef[[2]])}
\]

# cuts
\[
\text{cbind(true} = \text{cuts1, estimate} = \text{model$cuts[[1]])}
\]

# regression coefficients of the first continuous equation
\[
\text{cbind(true} = \text{beta0_y, estimate} = \text{model$coef2[[1]]}[1, ])
\]

# regression coefficients of the second continuous equation
\[
\text{cbind(true} = \text{beta0_g, estimate} = \text{model$coef2[[2]]}[1, ])
\]

# variances
\[
\text{cbind(true} = \text{c(var20, var21), estimate} = \text{model$var2[[1]]})
\]

# correlation between ordered equations
\[
\text{cbind(true} = \text{c(sigma[1, 2], estimate} = \text{model$sigma[1, 2])}
\]

# covariances between continuous and ordered equations
\[
\text{cbind(true} = \text{c(sigma[1:2, 3], estimate} = \text{model$cov2[[1]]}[1, ])
\]

# covariances between continuous equations
\[
\text{cbind(true} = \text{c(sigma[4, 7], sigma[3, 5], sigma[4, 6]),}
\]
estimate = model$sigma2[[1]])

# ---
# Step 3
# Estimation of 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), E(g1)
predict(model, group2 = c(0, 1), newdata = newdata)

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

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

---

nobs.mnprobit

Extract the Number of Observations from a Fit of the mnprobit Function.

Description

Extract the number of observations from a model fit of the mnprobit function.

Usage

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

Arguments

object object of class "mnprobit"

... further arguments (currently ignored)
**Details**

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

**Value**

A single positive integer number.

---

**nobs.mvoprobit**

*Extract the Number of Observations from a Fit of the mvoprobit Function.*

**Description**

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

**Usage**

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

**Arguments**

- `object` : object of class "mvoprobit"
- `...` : further arguments (currently ignored)

**Details**

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

**Value**

A single positive integer number.

---

**predict.mnprobit**

*Predict method for mnprobit function*

**Description**

Predict method for mnprobit function
### S3 method for class 'mnprobit'

```r
predict(
  object,
  ..., 
  newdata = NULL,
  alt = 1,
  regime = -1,
  type = ifelse(is.null(regime) | (regime == -1), "prob", "val"),
  alt_obs = "all",
  me = NULL,
  eps = NULL,
  control = list(),
  se = FALSE
)
```

## Arguments

- `object`: object of class "mnprobit"
- `...`: 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).
- `alt`: index of the alternative. See 'Details' for more information.
- `regime`: regime of the continuous equation. See 'Details' for more information.
- `type`: string representing a type of prediction. See 'Details' for more information.
- `alt_obs`: if `alt_obs = "all"` then all observations will be used for prediction. If `alt_obs` equals to the index of the alternative then only observations associated with this alternative will be used.
- `me`: string representing the name of the variable for which marginal effect should be estimated. See 'Details' for more information.
- `eps`: numeric vector of length 1 or 2 used for calculation of marginal effects. See 'Details'.
- `control`: list of additional arguments. Currently is not intended for the users.
- `se`: logical; if TRUE then the function also returns standard errors and p-values of the two-sided significance test associated with the function output. Works only if `predict.mvoprobit` returns numeric vector or a single column matrix. See `delta_method` for more information.

## Details

See 'Examples' section of `mnprobit` for examples of this function application.

If `type = "prob"` then function returns a probability that alternative `alt` will be chosen. For example if `alt = 2` then probabilities $P(z_i = 2 | w_i)$ will be estimated. If `n_alt` is null then the function returns a matrix such that i-th column contains probability of selecting i-th alternative.
If type = "li" then function returns a matrix which columns are linear indexes of corresponding equations.

If type = "val" then function returns predictions of conditional (on group) expectation of dependent variable in continuous equation with regimes determined by regime argument. To predict unconditional expectations just set alt = NULL.

If type = "lambda" then function returns conditional (on alt) expectation of random error of continuous equation in regime regime.

If me is provided then the function returns marginal effect of variable me respect to the statistic determined by type argument. For example if me = "x1" and type = "prob" then 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 numeric differentiation procedure. If eps is NULL then this increment will be selected automatically taking into account scaling of variables. If length(eps) = 2 then marginal effects will be estimated as the difference between predicted value when variable me equals eps[2] and when it equals eps[1] correspondingly.

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).

predict.mvoprobit

S3 method for mvoprobit function

Description

Predicted values based on object of class 'mvoprobit'.

Usage

## S3 method for class 'mvoprobit'
predict(
  object,
  ..., 
  newdata = NULL,
  given_ind = numeric(),
  group = NULL,
  group2 = NULL,
  type = ifelse(is.null(group2), "prob", "val"),
  me = NULL,
  eps = NULL,
  control = list(),
  se = FALSE
)
predict.mvoprobit

Arguments

  object  object of class "mvoprobit"
  ...    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 numeric vector of indexes of conditioned components.
  group     numeric vector which i-th element represents a value of the i-th dependent vari-
            able. If this value equals -1 then this component will be ignored (useful for
            estimation of marginal probabilities).
  group2    numeric vector which i-th element represents a value of the i-th dependent vari-
            able of the continuous equation. If this value equals -1 then this component will
            be ignored.
  type      string representing a type of prediction. See 'Details' for more information.
  me        string representing the name of the variable for which marginal effect should be
            estimated. See 'Details' for more information.
  eps       numeric vector of length 1 or 2 used for calculation of marginal effects. See
            'Details'.
  control   list of additional arguments. Currently is not intended for the users.
  se        logical; if TRUE then the function also returns standard errors and p-values of
            the two-sided significance test associated with the function output. Works only
            if predict.mvoprobit returns numeric vector or a single column matrix. See
            delta_method for more information.

Details

See 'Examples' section of mvoprobit for examples of this function application.

If type = "prob" then function returns a joint probability that dependent variables will have values
assigned in group. To calculate marginal probabilities set unnecessary group values to -1. To
estimate conditional probabilities provide indexes of conditioned variables through given_ind. For
example if \( z_1, z_2 \) and \( z_3 \) are dependent variables then to calculate \( P(z_1 = 2 | z_3 = 0) \) set given_ind
= 3 and groups = c(2, -1, 0). Note that conditioning on covariates (regressors) is omitted for
notations brevity and this conditioning depends on the values in newdata.

If type = "li" then function returns a matrix which columns are linear indexes of corresponding
equations.

If type = "sd" then function returns a matrix which columns are standard deviations of random
errors for corresponding equations.

If type = "li" or type = "sd" and some groups are equal to -1 then corresponding components
will be omitted from the output matrix.

If type = "val" then function returns predictions of conditional (on group) expectation of depend-
ent variable in continuous equations with regimes determined by group2 argument. To predict
unconditional expectations just set group = NULL.
If \( \text{type} = \text{"lambda"} \) then function returns conditional (on \( \text{group} \)) expectations of random error of continuous equation in regime \( \text{group2} \).

If \( \text{type} = \text{"val"} \) or \( \text{type} = \text{"lambda"} \) then output is a matrix which \( i \)-th column corresponds to prediction associated with \( i \)-th continuous equation.

If \( \text{me} \) is provided then the function returns marginal effect of variable \( \text{me} \) respect to the statistic determined by \( \text{type} \) argument. For example if \( \text{me} = \text{"x1"} \) and \( \text{type} = \text{"prob"} \) then function returns marginal effect of \( x1 \) on the corresponding probability i.e. one that would be estimated if \( \text{me} \) is \( \text{NULL} \).

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

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

\[
\frac{\partial P(z_1 = 2 | z_3 = 0)}{\partial x_1}.
\]

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

\[
P(z_1 = 2 | z_3 = 0, x_1 = 3) - P(z_1 = 2 | z_3 = 0, x_1 = 1).
\]

Value

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

---

**print.lrtest**

*Print Method for Likelihood Ratio Test*

**Description**

Prints summary for an object of class 'lrtest'.

**Usage**

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

**Arguments**

- \( x \) object of class "lrtest"
- \( ... \) further arguments (currently ignored).

**Value**

The function returns input argument \( x \).
print.mnprobit  
Print for an Object of Class mnprobit

Description

Prints information on the object of class ‘mnprobit’.

Usage

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

Arguments

x       object of class ‘mnprobit’
...     further arguments (currently ignored)

Value

The function returns input argument NULL.

print.mvoprobit  
Print for an Object of Class mvoprobit

Description

Prints information on the object of class ‘mvoprobit’.

Usage

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

Arguments

x       object of class ‘mvoprobit’
...     further arguments (currently ignored)

Value

The function returns NULL.
print.summary.delta_method

Print summary for an Object of Class delta_method

Description

Prints summary for an object of class 'delta_method'.

Usage

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

Arguments

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

Value

The function returns input argument x.

print.summary.lrtest  Print Summary Method for Likelihood Ratio Test

Description

Prints summary for an object of class 'lrtest'.

Usage

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

Arguments

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

Value

The function returns input argument x changing it's class to lrtest.
**print.summary.mnprobit**

*Print summary for an Object of Class mnprobit*

**Description**

Prints summary for an object of class 'mnprobit'.

**Usage**

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

**Arguments**

- `x`: object of class 'mnprobit'
- `...`: further arguments (currently ignored)

**Value**

The function returns `x`.

---

**print.summary.mvoprobit**

*Print summary for an Object of Class mvoprobit*

**Description**

Prints summary for an object of class 'mvoprobit'.

**Usage**

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

**Arguments**

- `x`: object of class 'mvoprobit'
- `...`: further arguments (currently ignored)

**Value**

The function returns `x`. 
Description

Extract standard deviations of random errors of continuous equation of \texttt{mnprobit} function.

Usage

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

Arguments

- \texttt{object} : object of class \texttt{"mnprobit"}.
- \texttt{use.fallback} : logical, passed to \texttt{noobs} (currently ignored).
- \texttt{...} : further arguments (currently ignored).
- \texttt{regime} : regime of continuous equation

Details

Available only if \texttt{estimator = "ml"}.

Value

Returns the estimates of standard deviations of \( \varepsilon_i \). If \texttt{regime = r} then estimate of \( \sqrt{Var(\varepsilon_{ri})} \) is returned.

Description

Extract standard deviations of random errors of continuous equations of \texttt{mvoprobit} function.

Usage

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

Arguments

- **object**: object of class "mvoprobit".
- **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 $\varepsilon_i$. If $eq2 = k$ then estimates only for $k$-th continuous equation are returned. If in addition `regime = r` then estimate of $\sqrt{\text{Var}(\varepsilon_{ri})}$ 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

```r
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

```r
p_value <- c(0.002, 0.2, 0.03, 0.08)
starsVector(p_value)
```
**summary.delta_method**  
*Summary for an Object of Class delta_method*

**Description**

Provides summary for an object of class 'delta_method'.

**Usage**

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

**Arguments**

- `object` object of class 'delta_method'
- `...` further arguments (currently ignored)

**Value**

Returns an object of class 'summary.delta_method'.

---

**summary.lrtest**  
*Summary Method for Likelihood Ratio Test*

**Description**

Provides summary for an object of class 'lrtest'.

**Usage**

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

**Arguments**

- `object` object of class 'lrtest'
- `...` further arguments (currently ignored)

**Details**

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

**Value**

Returns an object of class 'summary.lrtest'.
Summary for an Object of Class `mnprobit`

Description

Provides summary for an object of class `mnprobit`.

Usage

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

Arguments

- `object`: object of class `mnprobit`
- `...`: 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 `vcov.mnprobit` function will be used which 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 `mvoprobit`.

Details

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

Value

Returns an object of class `summary.mnprobit`.

Summary for an Object of Class `mvoprobit`

Description

Provides summary for an object of class `mvoprobit`.

Usage

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

object          object of class 'mvoprobit'
...             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 vcov.mvoprobit function will be used which 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 mvoprobit.

Details

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

Value

Returns an object of class 'summary.mvoprobit'.

Description

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

Usage

## S3 method for class 'mnprobit'
vcov(
  object,
  ...
  type = object$cov_type,
  regime = NULL,
  n_cores = object$other$n_cores,
  n_sim = object$other$n_sim
)
Arguments

object: an object of class `mnprobit`.
...
 further arguments (currently ignored).

type: character representing the type of the asymptotic covariance matrix estimator. If non-negative integer representing the regime of the two-step procedure for which covariance matrix should be returned. If estimator = "2step" and regime = NULL then covariance matrix of the first step parameters’ estimator will be returned. Otherwise the function estimates covariance matrix for the second step parameters associated with corresponding regime.

regime: positive integer representing the number of CPU cores used for parallel computing. If possible it is highly recommend to set it equal to the number of available physical cores especially when the system of ordered equations has 2 or 3 equations. estimator argument of `mnprobit` is "ml" then type may be changed to any value available for input argument cov_type of `mnprobit`. Otherwise type will coincide with cov_type output value of `mnprobit`.

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

Details

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

Value

Returns numeric matrix which represents estimate of the asymptotic covariance matrix of model’s parameters.

vcov.mvoprobit

`v cov.mvoprobit` Calculate Variance-Covariance Matrix for a mvoprobit Object.

Description

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

Usage

```r
## S3 method for class 'mvoprobit'
v cov( object, 
 ...,
 type = object$ cov_type, 
 regime = NULL, 
 n_ cores = object$other$n_ cores, 
 n_ sim = object$other$n_ sim 
 )
```
Arguments

- **object**: an object of class `mvoprobit`.
- **...**: further arguments (currently ignored).
- **type**: character representing the type of the asymptotic covariance matrix estimator. If `estimator` argument of `mvoprobit` is "ml" then `type` may be changed to any value available for input argument `cov_type` of `mvoprobit`. Otherwise `type` will coincide with `cov_type` output value of `mvoprobit`.
- **regime**: non-negative integer representing the regime of the two-step procedure for which covariance matrix should be returned. If `estimator = "2step"` and `regime = NULL` or `is.na(regime)` then covariance matrix of the first step parameters’ estimator will be returned. Otherwise the function estimates covariance matrix for the second step parameters associated with corresponding regime.
- **n_cores**: positive integer representing the number of CPU cores used for parallel computing. If possible it is highly recommend 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.

Details

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

Value

Returns numeric matrix which represents estimate of the asymptotic covariance matrix of model’s parameters.
Index

* datasets
   - cps, 6

boot, 2
   - coef.mnprobit, 3, 7, 25
   - coef.mvoprobit, 4, 7, 40
   - cps, 6
   - delta_method, 7, 67, 69

fitted.mnprobit, 10
   - fitted.mvoprobit, 11
   - formula.mnprobit, 12
   - formula.mvoprobit, 12
   - formula_merge, 13
   - formula_split, 14
   - gena, 21, 23, 33, 38
   - grad_mnprobit, 14
   - grad_mvoprobit, 15
   - lnl_mnprobit, 16
   - lnl_mvoprobit, 16
   - logLik, 19
   - logLik.mnprobit, 17
   - logLik.mvoprobit, 18
   - loocv, 18
   - lrtest, 19
   - mnprobit, 4, 7, 17, 20, 65, 67, 74, 79
   - mvoprobit, 5, 7, 18, 23, 32, 38, 66, 69, 74, 77, 78, 80
   - nobs, 19
   - nobs.mnprobit, 65
   - nobs.mvoprobit, 66
   - optim, 21, 23, 33, 38
   - pmnorm, 23, 33, 38

   - predict.mnprobit, 7, 66
   - predict.mvoprobit, 7, 39, 67, 68, 69
   - print.lrtest, 70
   - print.mnprobit, 71
   - print.mvoprobit, 71
   - print.summary.delta_method, 72
   - print.summary.lrtest, 72
   - print.summary.mnprobit, 73
   - print.summary.mvoprobit, 73
   - pso, 21, 23, 33, 38
   - sigma.mnprobit, 74
   - sigma.mvoprobit, 74
   - starsVector, 75
   - summary.delta_method, 8, 76
   - summary.lrtest, 76
   - summary.mnprobit, 21, 77
   - summary.mvoprobit, 34, 77
   - vcov.mnprobit, 77, 78
   - vcov.mvoprobit, 78, 79