Package 'hpa'

July 1, 2020

Type Package

Title Distributions Hermite Polynomial Approximation

Version 1.1.1

Date 2020-07-01

Author Potanin Bogdan

Maintainer Potanin Bogdan <bogdanpotanin@gmail.com>

Description Multivariate conditional and marginal densities, moments, cumulative distribution functions as well as binary choice and sample selection models based on hermite polynomial approximation which was proposed and described by A. Gallant and D. W. Nychka (1987) <doi:10.2307/1913241>.

License GPL-3

Imports Rcpp (>= 1.0.4), RcppParallel (>= 5.0.0)

LinkingTo Rcpp, RcppArmadillo, RcppParallel

RoxygenNote 7.1.0

Encoding UTF-8

Suggests ggplot2, mvtnorm, titanic, sampleSelection, GA (>= 3.2)

NeedsCompilation yes

SystemRequirements GNU make

Repository CRAN

Date/Publication 2020-07-01 12:00:15 UTC

R topics documented:

AIC.hpaBinary *Calculates AIC for "hpaBinary" object*

Description

This function calculates AIC for "hpaBinary" object

Usage

```
## S3 method for class 'hpaBinary'
AIC(object, ..., k = 2)
```

Arguments

`object`	Object of class "hpaBinary"
`...`	further arguments (currently ignored)
`k`	numeric, the penalty per parameter to be used; the default k = 2 is the classical AIC.

AIC.hpaML *Calculates AIC for "hpaML" object*

Description

This function calculates AIC for "hpaML" object

Usage

```
## S3 method for class 'hpaML'
AIC(object, ..., k = 2)
```

Arguments

`object`	Object of class "hpaML"
`...`	further arguments (currently ignored)
`k`	numeric, the penalty per parameter to be used; the default k = 2 is the classical AIC.

AIC.hpaSelection *Calculates AIC for "hpaSelection" object*

Description

This function calculates AIC for "hpaSelection" object

Usage

```
## S3 method for class 'hpaSelection'
AIC(object, ..., k = 2)
```

Arguments

`object`	Object of class "hpaSelection"
`...`	further arguments (currently ignored)
`k`	numeric, the penalty per parameter to be used; the default k = 2 is the classical AIC.

AIC_hpaBinary *Calculates AIC for "hpaBinary" object*

Description

This function calculates AIC for "hpaBinary" object

Usage

```
AIC_hpaBinary(object, k = 2)
```

Arguments

`object`	Object of class "hpaBinary"
`k`	numeric, the penalty per parameter to be used; the default k = 2 is the classical AIC.

AIC_hpaML — *Calculates AIC for "hpaML" object*

Description

This function calculates AIC for "hpaML" object

Usage

```
AIC_hpaML(object, k = 2)
```

Arguments

`object`	Object of class "hpaML"
`k`	numeric, the penalty per parameter to be used; the default k = 2 is the classical AIC.

AIC_hpaSelection — *Calculates AIC for "hpaSelection" object*

Description

This function calculates AIC for "hpaSelection" object

Usage

```
AIC_hpaSelection(object, k = 2)
```

Arguments

`object`	Object of class "hpaSelection"
`k`	numeric, the penalty per parameter to be used; the default k = 2 is the classical AIC.

dhpa *Density function hermite polynomial approximation*

Description

This function calculates density function hermite polynomial approximation.

Usage

```
dhpa(
  x = matrix(1, 1),
  pol_coefficients = numeric(0),
  pol_degrees = numeric(0),
  given_ind = logical(0),
  omit_ind = logical(0),
  mean = numeric(0),
  sd = numeric(0),
  is_parallel = FALSE
)
```

Arguments

`x`	numeric matrix of density function arguments. Note that `x` rows are observations while variables are columns.
`pol_coefficients`	numeric vector of polynomial coefficients.
`pol_degrees`	non-negative integer vector of polynomial degrees.
`given_ind`	logical vector indicating wheather corresponding component is conditioned. By default it is a logical vector of FALSE values.
`omit_ind`	logical vector indicating wheather corresponding component is omitted. By default it is a logical vector of FALSE values.
`mean`	numeric vector of expected values.
`sd`	positive numeric vector of standard deviations.
`is_parallel`	if TRUE then multiple cores will be used for some calculations. It usually provides speed advantage for large enough samples (about more than 1000 observations).

Details

Densities hermite polynomial approximation approach has been proposed by A. Gallant and D. W. Nychka in 1987. The main idea is to approximate unknown distribution density with hermite polynomial of degree `pol_degree`. In this framework hermite polynomial represents adjusted (to insure integration to 1) product of squared polynomial and normal distribution densities. Parameters `mean` and `sd` determine means and standard deviations of normal distribution density functions which are parts of this polynomial. For more information please refer to the literature listed below.

Parameters `mean`, `sd`, `given_ind`, `omit_ind` should have the same length as `pol_degrees` parameter.

Value

This function returns density function hermite polynomial approximation at point `x`.

References

A. Gallant and D. W. Nychka (1987) <doi:10.2307/1913241>

Examples

```
## Let's approximate some three random variables joint density function
## at point (0,1, 0.2, 0.3) with hermite polynomial of (1,2,3) degrees which polynomial
## coefficients equals 1 except coefficient related to x1*(x^3) polynomial element
## which equals 2. Also suppose that normal density related mean vector
## equals (1.1, 1.2, 1.3) while standard deviations vector is (2.1, 2.2, 2.3).

# Prepare initial values
x <- matrix(c(0.1, 0.2, 0.3), nrow=1)
mean <- c(1.1, 1.2, 1.3)
sd <- c(2.1, 2.2, 2.3)
pol_degrees <- c(1, 2, 3)

# Create polynomial powers and indexes correspondence matrix
pol_ind <- polynomialIndex(pol_degrees)
# Set all polynomial coefficients to 1
pol_coefficients <- rep(1, ncol(pol_ind))
pol_degrees_n <- length(pol_degrees)

# Assign coefficient 2 to the polynomial element(x1 ^ 1)*(x2 ^ 0)*(x3 ^ 2)
pol_coefficients[which(colSums(pol_ind == c(1, 0, 2)) == pol_degrees_n)] <- 2

# Visualize correspondence between polynomial elements and their coefficients
as.data.frame(rbind(pol_ind, pol_coefficients),
row.names = c("x1 power", "x2 power", "x3 power", "coefficients"),
optional = TRUE)
printPolynomial(pol_degrees, pol_coefficients)

# Calculate density approximation at point x
dhpa(x = x,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd)

# Condition second component to be 0.5
# Substitute x second component with conditional value 0.5
x <- matrix(c(0.1, 0.5, 0.3), nrow = 1)
#Set TRUE to the second component indicating that it is conditioned
given_ind <- c(FALSE, TRUE, FALSE)

# Calculate conditional (on x2=0.5) density approximation at point x
dhpa(x = x,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd,
given_ind = given_ind)
```

```
# Consider third component marginal distribution
# conditioned on the second component 0.5 value
# Set TRUE to the first component indicating that it is omitted
omit_ind <- c(TRUE, FALSE, FALSE)

# Calculate conditional (on x2=0.5) marginal (for x3) density approximation
# at point x
dhpa(x = x,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd,
given_ind = given_ind, omit_ind = omit_ind)
```

---

dhpaDiff *Calculate gradient of density function hermite polynomial approximation*

Description

This function calculates gradient of density function hermite polynomial approximation.

Usage

```
dhpaDiff(
  x = matrix(1, 1),
  pol_coefficients = numeric(0),
  pol_degrees = numeric(0),
  given_ind = logical(0),
  omit_ind = logical(0),
  mean = numeric(0),
  sd = numeric(0),
  type = "pol_coefficients",
  is_parallel = FALSE
)
```

Arguments

x	numeric matrix of density function arguments. Note that x rows are observations while variables are columns.
pol_coefficients	numeric vector of polynomial coefficients.
pol_degrees	non-negative integer vector of polynomial degrees.
given_ind	logical vector indicating wheather corresponding component is conditioned. By default it is a logical vector of FALSE values.
omit_ind	logical vector indicating wheather corresponding component is omitted. By default it is a logical vector of FALSE values.
mean	numeric vector of expected values.

`sd`	positive numeric vector of standard deviations.
`type`	determines the partial derivatives to be included into gradient. Currently `type="pol_coefficients"` is the only available option (default) meaning that the gradient will contain partial derivatives respect to polynomial coefficients listed in the same order as `pol_coefficients`.
`is_parallel`	if TRUE then multiple cores will be used for some calculations. It usually provides speed advantage for large enough samples (about more than 1000 observations).

Details

Densities hermite polynomial approximation approach has been proposed by A. Gallant and D. W. Nychka in 1987. The main idea is to approximate unknown distribution density with hermite polynomial of degree `pol_degree`. In this framework hermite polynomial represents adjusted (to insure integration to 1) product of squared polynomial and normal distribution densities. Parameters `mean` and `sd` determine means and standard deviations of normal distribution density functions which are parts of this polynomial. For more information please refer to the literature listed below.

Parameters `mean`, `sd`, `given_ind`, `omit_ind` should have the same length as `pol_degrees` parameter.

If `x` has more then one row then the output will be jacobian matrix where rows are gradients.

Value

This function returns gradient of density function hermite polynomial approximation at point `x`. Gradient elements are determined by the `type` argument.

References

A. Gallant and D. W. Nychka (1987) <doi:10.2307/1913241>

Examples

```
## Let's approximate some three random variables joint density function
## at point (0,1, 0.2, 0.3) with hermite polynomial of (1,2,3) degrees which polynomial
## coefficients equals 1 except coefficient related to x1*(x^3) polynomial element
## which equals 2. Also suppose that normal density related mean vector
## equals (1.1, 1.2, 1.3) while standard deviations vector is (2.1, 2.2, 2.3).
## In this example let's calculate density approximating function gradient respect to
## polynomial coefficients.

# Prepare initial values
x <- matrix(c(0.1, 0.2, 0.3), nrow=1)
mean <- c(1.1, 1.2, 1.3)
sd <- c(2.1, 2.2, 2.3)
pol_degrees <- c(1, 2, 3)

# Create polynomial powers and indexes correspondence matrix
pol_ind <- polynomialIndex(pol_degrees)
# Set all polynomial coefficients to 1
 pol_coefficients <- rep(1, ncol(pol_ind))
 pol_degrees_n <- length(pol_degrees)

# Assign coefficient 2 to the polynomial element(x1 ^ 1)*(x2 ^ 0)*(x3 ^ 2)
pol_coefficients[which(colSums(pol_ind == c(1, 0, 2)) == pol_degrees_n)] <- 2

# Visualize correspondence between polynomial elements and their coefficients
as.data.frame(rbind(pol_ind, pol_coefficients),
 row.names = c("x1 power", "x2 power", "x3 power", "coefficients"),
 optional = TRUE)
printPolynomial(pol_degrees, pol_coefficients)

# Calculate density approximation gradient
# respect to polynomial coefficients at point x
dhpaDiff(x = x,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd)

# Condition second component to be 0.5
# Substitute x second component with conditional value 0.5
x <- matrix(c(0.1, 0.5, 0.3), nrow = 1)
# Set TRUE to the second component indicating that it is conditioned
given_ind <- c(FALSE, TRUE, FALSE)

# Calculate conditional (on x2=0.5) density approximation
# gradient respect to polynomial coefficients at point x
dhpaDiff(x = x,
 pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
 mean = mean, sd = sd,
 given_ind = given_ind)

# Consider third component marginal distribution
# conditioned on the second component 0.5 value
# Set TRUE to the first component indicating that it is omitted
omit_ind <- c(TRUE, FALSE, FALSE)

# Calculate conditional (on x2=0.5) marginal (for x3) density approximation
# gradient respect to polynomial coefficients at point x
dhpaDiff(x = x,
 pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
 mean = mean, sd = sd,
 given_ind = given_ind, omit_ind = omit_ind)
```

`dnorm_parallel` *Calculate normal pdf in parallel*

Description

Calculate in parallel for each value from vector `x` density function of normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

Usage

```
dnorm_parallel(x, mean = 0, sd = 1, is_parallel = FALSE)
```

Arguments

`x`	vector of quantiles: should be numeric vector, not just double value.
`mean`	double value.
`sd`	double positive value.
`is_parallel`	if TRUE then multiple cores will be used for some calculations. It usually provides speed advantage for large enough samples (about more than 1000 observations).

`dtrhpa` *Truncated density function hermite polynomial approximation*

Description

This function calculates truncated density function hermite polynomial approximation.

Usage

```
dtrhpa(
  x = matrix(1, 1),
  tr_left = matrix(),
  tr_right = matrix(),
  pol_coefficients = numeric(0),
  pol_degrees = numeric(0),
  given_ind = logical(0),
  omit_ind = logical(0),
  mean = numeric(0),
  sd = numeric(0),
  is_parallel = FALSE
)
```

Arguments

`x`	numeric matrix of density function arguments. Note that `x` rows are observations while variables are columns.
`tr_left`	numeric matrix of left (lower) truncation limits. Note that `tr_right` rows are observations while variables are columns. If `tr_left` or `tr_right` is single row matrix then the same truncation limits would be applied to all observations that are determined by the first rows of these matrices.
`tr_right`	numeric matrix of right (upper) truncation limits. Note that `tr_right` rows are observations while variables are columns. If `tr_left` or `tr_right` is single row matrix then the same truncation limits would be applied to all observations that are determined by the first rows of these matrices.

`pol_coefficients`	numeric vector of polynomial coefficients.
`pol_degrees`	non-negative integer vector of polynomial degrees.
`given_ind`	logical vector indicating wheather corresponding component is conditioned. By default it is a logical vector of FALSE values.
`omit_ind`	logical vector indicating wheather corresponding component is omitted. By default it is a logical vector of FALSE values.
`mean`	numeric vector of expected values.
`sd`	positive numeric vector of standard deviations.
`is_parallel`	if TRUE then multiple cores will be used for some calculations. It usually provides speed advantage for large enough samples (about more than 1000 observations).

Details

Densities hermite polynomial approximation approach has been proposed by A. Gallant and D. W. Nychka in 1987. The main idea is to approximate unknown distribution density with hermite polynomial of degree `pol_degree`. In this framework hermite polynomial represents adjusted (to insure integration to 1) product of squared polynomial and normal distribution densities. Parameters `mean` and `sd` determine means and standard deviations of normal distribution density functions which are parts of this polynomial. For more information please refer to the literature listed below.

Parameters `mean`, `sd`, `given_ind`, `omit_ind` should have the same length as `pol_degrees` parameter.

Value

This function returns density function hermite polynomial approximation at point `x` for truncated distribution.

References

A. Gallant and D. W. Nychka (1987) <doi:10.2307/1913241>

Examples

```
##Let's approximate some three truncated random variables joint density function
##at point (0,1, 0.2, 0.3) with hermite polynomial of (1,2,3) degrees which polynomial
##coefficients equals 1 except coefficient related to x1*(x^3) polynomial
## element which equals 2. Also suppose that normal density related
## mean vector equals (1.1, 1.2, 1.3) while standard deviations vector
## is (2.1, 2.2, 2.3). Suppose that lower and upper truncation points
## are (-1.1,-1.2,-1.3) and (1.1,1.2,1.3) correspondingly.

#Prepare initial values
x <- matrix(c(0.1, 0.2, 0.3), nrow=1)
tr_left = matrix(c(-1.1,-1.2,-1.3), nrow = 1)
tr_right = matrix(c(1.1,1.2,1.3), nrow = 1)
mean <- c(1.1, 1.2, 1.3)
sd <- c(2.1, 2.2, 2.3)
```

```
pol_degrees <- c(1, 2, 3)

#Create polynomial powers and indexes correspondence matrix
pol_ind <- polynomialIndex(pol_degrees)

#Set all polynomial coefficients to 1
pol_coefficients <- rep(1, ncol(pol_ind))
pol_degrees_n <- length(pol_degrees)

#Assign coefficient 2 to the polynomial element(x1 ^ 1)*(x2 ^ 0)*(x3 ^ 2)
pol_coefficients[which(colSums(pol_ind == c(1, 0, 2)) == pol_degrees_n)] <- 2

#Visualize correspondence between polynomial elements and their coefficients
as.data.frame(rbind(pol_ind, pol_coefficients),
row.names = c("x1 power", "x2 power", "x3 power", "coefficients"),
optional = TRUE)
printPolynomial(pol_degrees, pol_coefficients)

#Calculate density approximation at point x
dtrhpa(x = x,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd,
tr_left = tr_left, tr_right = tr_right)

#Condition second component to be 0.5
#Substitute x second component with conditional value 0.5
x <- matrix(c(0.1, 0.5, 0.3), nrow = 1)
#Set TRUE to the second component indicating that it is conditioned
given_ind <- c(FALSE, TRUE, FALSE)
#Calculate conditional (on x2=0.5) density approximation at point x
dtrhpa(x = x,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd,
given_ind = given_ind,
tr_left = tr_left, tr_right = tr_right)

#Consider third component marginal distribution
#conditioned on the second component 0.5 value
#Set TRUE to the first component indicating that it is omitted
omit_ind <- c(TRUE, FALSE, FALSE)

#Calculate conditional (on x2=0.5) marginal (for x3) density approximation at point x
dtrhpa(x = x,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd,
given_ind = given_ind, omit_ind = omit_ind,
tr_left = tr_left, tr_right = tr_right)
```

ehpa *Expected powered product hermite polynomial approximation*

Description

This function calculates expected powered product hermite polynomial approximation.

Usage

```
ehpa(
  x = matrix(1, 1),
  pol_coefficients = numeric(0),
  pol_degrees = numeric(0),
  given_ind = logical(0),
  omit_ind = logical(0),
  mean = numeric(0),
  sd = numeric(0),
  expectation_powers = numeric(0),
  is_parallel = FALSE
)
```

Arguments

`x`	non-negative numeric matrix of quantiles. Note that `x` rows are observations while variables are columns.
`pol_coefficients`	numeric vector of polynomial coefficients.
`pol_degrees`	non-negative integer vector of polynomial degrees.
`given_ind`	logical vector indicating wheather corresponding component is conditioned. By default it is a logical vector of FALSE values.
`omit_ind`	logical vector indicating wheather corresponding component is omitted. By default it is a logical vector of FALSE values.
`mean`	numeric vector of expected values.
`sd`	positive numeric vector of standard deviations.
`expectation_powers`	integer vector of random vector components powers.
`is_parallel`	if TRUE then multiple cores will be used for some calculations. It usually provides speed advantage for large enough samples (about more than 1000 observations).

Details

Expected powered product of random variables is expectation of their product given powers `expectation_powers`. Therefore in order to approximate expected value of i-th random vector component just set all `expectation_powers` to zero except it's i-th component which should be assigned 1.

Densities hermite polynomial approximation approach has been proposed by A. Gallant and D. W. Nychka in 1987. The main idea is to approximate unknown distribution density with hermite polynomial of degree `pol_degree`. In this framework hermite polynomial represents adjusted (to insure integration to 1) product of squared polynomial and normal distribution densities. Parameters mean and sd determine means and standard deviations of normal distribution density functions which are parts of this polynomial. For more information please refer to the literature listed below.

Parameters mean, sd, given_ind, omit_ind should have the same length as pol_degrees parameter.

Value

This function returns numeric vector of expected powered product hermite polynomial approximations.

References

A. Gallant and D. W. Nychka (1987) <doi:10.2307/1913241>

Examples

```
##Let's approximate some three random variables powered product expectation for
##powers (3,2,1) with hermite polynomial of (1,2,3) degrees which polynomial coefficients
##equals 1 except coefficient related to x1*(x^3) polynomial element which equals 2.
## Also suppose that normal density related mean vector equals (1.1, 1.2, 1.3) while
## standard deviations vector is (2.1, 2.2, 2.3).

#Prepare initial values
expectation_powers = c(3,2,1)
mean <- c(1.1, 1.2, 1.3)
sd <- c(2.1, 2.2, 2.3)
pol_degrees <- c(1, 2, 3)

#Create polynomial powers and indexes correspondence matrix
pol_ind <- polynomialIndex(pol_degrees)

#Set all polynomial coefficients to 1
pol_coefficients <- rep(1, ncol(pol_ind))
pol_degrees_n <- length(pol_degrees)

#Assign coefficient 2 to the polynomial element (x1 ^ 1)*(x2 ^ 0)*(x3 ^ 2)
pol_coefficients[which(colSums(pol_ind == c(1, 0, 2)) == pol_degrees_n)] <- 2

#Visualize correspondence between polynomial elements and their coefficients
as.data.frame(rbind(pol_ind, pol_coefficients),
row.names = c("x1 power", "x2 power", "x3 power", "coefficients"),
optional = TRUE)
printPolynomial(pol_degrees, pol_coefficients)

#Calculate expected powered product approximation
ehpa(pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd, expectation_powers = expectation_powers)

#Condition second component to be 0.5
#Substitute x second component with conditional value 0.5
x <- matrix(c(NA, 0.5, NA), nrow = 1)
#Set TRUE to the second component indicating that it is conditioned
```

```
given_ind <- c(FALSE, TRUE, FALSE)

#Calculate conditional(on x2 = 0.5) expected powered product approximation
ehpa(x = x,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd, expectation_powers = expectation_powers,
given_ind = given_ind)

#Consider third component marginal distribution
#conditioned on the second component 0.5 value
#Set TRUE to the first component indicating that it is omitted
omit_ind <- c(TRUE, FALSE, FALSE)

#Calculate conditional (on x2=0.5) marginal (for x3) expected powered
#product approximation at points x_lower and x_upper
ehpa(x = x,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd, expectation_powers = expectation_powers,
given_ind = given_ind, omit_ind = omit_ind)
```

etrhpa — *Expected powered product hermite polynomial approximation for truncated distribution*

Description

This function calculates expected powered product hermite polynomial approximation for truncated distribution.

Usage

```
etrhpa(
  tr_left = matrix(1, 1),
  tr_right = matrix(1, 1),
  pol_coefficients = numeric(0),
  pol_degrees = numeric(0),
  mean = numeric(0),
  sd = numeric(0),
  expectation_powers = numeric(0),
  is_parallel = FALSE
)
```

Arguments

`tr_left`	numeric matrix of left (lower) truncation limits. Note that `tr_right` rows are observations while variables are columns. If `tr_left` or `tr_right` is single row matrix then the same truncation limits would be applied to all observations that are determined by the first rows of these matrices.

`tr_right`	numeric matrix of right (upper) truncation limits. Note that `tr_right` rows are observations while variables are columns. If `tr_left` or `tr_right` is single row matrix then the same truncation limits would be applied to all observations that are determined by the first rows of these matrices.
`pol_coefficients`	numeric vector of polynomial coefficients.
`pol_degrees`	non-negative integer vector of polynomial degrees.
`mean`	numeric vector of expected values.
`sd`	positive numeric vector of standard deviations.
`expectation_powers`	integer vector of random vector components powers.
`is_parallel`	if TRUE then multiple cores will be used for some calculations. It usually provides speed advantage for large enough samples (about more than 1000 observations).

Details

Expected powered product of random variables is expectation of their product given powers `expectation_powers`. Therefore in order to approximate expected value of i-th random vector component just set all `expectation_powers` to zero except it's i-th component which should be assigned 1.

Densities hermite polynomial approximation approach has been proposed by A. Gallant and D. W. Nychka in 1987. The main idea is to approximate unknown distribution density with hermite polynomial of degree `pol_degree`. In this framework hermite polynomial represents adjusted (to insure integration to 1) product of squared polynomial and normal distribution densities. Parameters `mean` and `sd` determine means and standard deviations of normal distribution density functions which are parts of this polynomial. For more information please refer to the literature listed below.

Parameters `mean`, `sd`, `given_ind`, `omit_ind` should have the same length as `pol_degrees` parameter.

Value

This function returns numeric vector of expected powered product hermite polynomial approximations for truncated distribution.

References

A. Gallant and D. W. Nychka (1987) <doi:10.2307/1913241>

Examples

```
##Let's approximate some three truncated random variables powered product expectation
##for powers (3,2,1) with hermite polynomial of (1,2,3) degrees which polynomial
##coefficients equals 1 except coefficient related to x1*(x^3) polynomial element which
##equals 2. Also suppose that normal density related mean vector equals (1.1, 1.2, 1.3)
##while standard deviations vector is (2.1, 2.2, 2.3). Suppose that lower and upper
##truncation points are (-1.1,-1.2,-1.3) and (1.1,1.2,1.3) correspondingly.

#Prepare initial values
```

```
expectation_powers = c(3,2,1)
tr_left = matrix(c(-1.1,-1.2,-1.3), nrow = 1)
tr_right = matrix(c(1.1,1.2,1.3), nrow = 1)
mean <- c(1.1, 1.2, 1.3)
sd <- c(2.1, 2.2, 2.3)
pol_degrees <- c(1, 2, 3)

#Create polynomial powers and indexes correspondence matrix
pol_ind <- polynomialIndex(pol_degrees)
#Set all polynomial coefficients to 1
pol_coefficients <- rep(1, ncol(pol_ind))
pol_degrees_n <- length(pol_degrees)

#Assign coefficient 2 to the polynomial element(x1 ^ 1)*(x2 ^ 0)*(x3 ^ 2)
pol_coefficients[which(colSums(pol_ind == c(1, 0, 2)) == pol_degrees_n)] <- 2

#Visualize correspondence between polynomial elements and their coefficients
as.data.frame(rbind(pol_ind, pol_coefficients),
row.names = c("x1 power", "x2 power", "x3 power", "coefficients"),
optional = TRUE)
printPolynomial(pol_degrees, pol_coefficients)

#Calculate expected powered product approximation for truncated distribution
etrhpa(pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd, expectation_powers = expectation_powers,
tr_left = tr_left, tr_right = tr_right)
```

hpaBinary *Perform semi-nonparametric binary choice model estimation*

Description

This function performs semi-nonparametric single index binary choice model estimation via hermite polynomial densities approximation.

Usage

```
hpaBinary(
  formula,
  data,
  K = 1L,
  z_mean_fixed = NA_real_,
  z_sd_fixed = NA_real_,
  z_constant_fixed = 0,
  is_z_coef_first_fixed = TRUE,
  is_x0_probit = TRUE,
  is_sequence = FALSE,
  x0 = numeric(0),
  cov_type = "sandwich",
  boot_iter = 100L,
  is_parallel = FALSE,
  opt_type = "optim",
  opt_control = NULL
)
```

Arguments

`formula`	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted.
`data`	data frame containing the variables in the model.
`K`	non-negative integer representing polynomial degree.
`z_mean_fixed`	numeric value for binary choice equation random error density mean parameter. Set it to NA (default) if this parameter should be estimated rather then fixed.
`z_sd_fixed`	numeric value for binary choice equation random error density `sd` parameter. Set it to NA (default) if this parameter should be estimated rather then fixed.
`z_constant_fixed`	numeric value for binary choice equation constant parameter. Set it to NA (default) if this parameter should be estimated rather then fixed.
`is_z_coef_first_fixed`	bool value indicating weather binary equation first independend variable coefficient should be fixed (TRUE) or estimated (FALSE).
`is_x0_probit`	logical; if `TRUE` (default) then initial points for optimization routine will be obtained by probit model estimated via glm function.
`is_sequence`	if TRUE then function calculates models with polynomial degrees from 0 to K each time using initial values obtained from the previous step. In this case function will return the list of models where i-th list element correspond to model calculated under K=(i-1).
`x0`	numeric vector of optimization routine initial values. Note that `x0=c(pol_coefficients[-1],mean,sd,`
`cov_type`	string value determinign 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 applied. If `cov_type = "sandwich"` (default) then sandwich covariance matrix estimator will be applied. If `cov_type = "bootstrap"` then bootstrap with `boot_iter` iterations will be applied. If `cov_type = "hessianFD"` or `cov_type = "sandwichFD"` then accurate but computationally demanding finite difference hessian approximation will be calculated for the inverse hessian and sandwich estimators correspondingly.
`boot_iter`	the number of bootstrap iterations for `cov_type = "bootstrap"` covariance matrix estimator type.
`is_parallel`	if TRUE then multiple cores will be used for some calculations. It usually provides speed advantage for large enough samples (about more than 1000 observations).
`opt_type`	string value determining the type of the optimization routine to be applied. The default is "optim" meaning that BFGS method from the `optim` function will be applied. If `opt_type = "GA"` then `ga` function will be additionally applied.

opt_control	a list containing arguments to be passed to the optimization routine depending on opt_type argument value. Please see details to get additional information.

Details

Semi-nonparametric (SNP) approach has been implemented via densities hermite polynomial approximation

Densities hermite polynomial approximation approach has been proposed by A. Gallant and D. W. Nychka in 1987. The main idea is to approximate unknown distribution density with hermite polynomial of degree pol_degree. In this framework hermite polynomial represents adjusted (to insure integration to 1) product of squared polynomial and normal distribution densities. Parameters mean and sd determine means and standard deviations of normal distribution density functions which are parts of this polynomial. For more information please refer to the literature listed below.

Parameters mean, sd, given_ind, omit_ind should have the same length as pol_degrees parameter.

The first polynomial coefficient (zero powers) set to identity constant for identification reasons.

Note that if is_z_coef_first_fixed value is TRUE then the coefficient for the first independent variable in formula will be fixed to 1.

Parameter sd will be scale adjusted in order to provide better initial point for optimization routine. Please, extract sd adjusted value from this function's output list.

All variables mentioned in formula should be numeric vectors.

The function calculates standard errors via sandwich estimator and significance levels are reported taking into account quasi maximum likelihood estimator (QMLE) asymptotic normality. If ones wants to switch from QMLE to semi-nonparametric estimator (SNPE) during hypothesis testing then covariance matrix should be reestimated using bootstrap.

This function maximizes (quasi) log-likelihood function via optim function setting it's method argument to "BFGS". If opt_type = "GA" then genetic algorithm from ga function will be additionally (after optim putting it's solution (par) to suggestions matrix) applied in order to perform global optimization. Note that global optimization takes much more time (usually minutes but sometimes hours or even days). The number of iterations and population size of the genetic algorithm will grow linearly along with the number of estimated parameters. If it is seems that global maximum has not been found than it is possible to continue the search restarting the function setting it's input argument x0 to x1 output value. Note that if cov_type = "bootstrap" then ga function will not be used for bootstrap iterations since it may be extremely time consuming.

If opt_type = "GA" then opt_control should be the list containing the values to be passed to ga function. It is possible to pass arguments lower, upper, popSize, pcrossover, pmutation, elitism, maxiter, suggestions, optim, optimArgs and seed. Note that it is possible to set population, selection, crossover and mutation arguments changing ga default parameters via gaControl function. These arguments information reported in ga. In order to provide manual values for lower and upper bounds please follow parameters ordering mentioned above for the x0 argument. If these bonds are not provided manually then they (except those related to the polynomial coefficients) will depend on the estimates obtained by local optimization via optim function (this estimates will be in the middle between lower and upper). Specifically for each sd parameter lower (upper) bound is 5 times lower (higher) then this parameter optim estimate. For each mean and regression coefficient parameter it's lower and upper bounds deviate from corresponding optim estimate by two absolute value of this estimate. Finally, lower and upper bounds for each polynomial coefficient are -10 and 10 correspondingly (do not depend on their optim estimates).

The following arguments are differ from their defaults in ga:

- pmutation = 0.2,
- optim = TRUE,
- optimArgs = list("method" = "Nelder-Mead","poptim" = 0.2,"pressel" = 0.5),
- seed = 8,
- elitism = 2 + round(popSize * 0.1).

Let's denote by n_reg the number of regressors included to the formula. The arguments popSize and maxiter of ga function have been set proportional to the number of estimated polynomial coefficients and independent variables:

- popSize = 10 + 5 * (K + 1) + 2 * n_reg
- maxiter = 50 * (1 + K) + 10 * n_reg

Value

This function returns an object of class "hpaBinary".

An object of class "hpaBinary" is a list containing the following components:

- optim - optim function output. If opt_type = "GA" then it is the list containing optim and ga functions outputs.
- x1 - numeric vector of distribution parameters estimates.
- mean - mean (mu) parameter of density function estimate.
- sd - sd (sigma) parameter of density function estimate.
- pol_coefficients - polynomial coefficients estimates.
- pol_degrees - the same as K input parameter.
- coefficients - regression (single index) coefficients estimates.
- cov_matrix - estimated parameters covariance matrix estimate.
- marginal_effects - marginal effects matrix where columns are variables and rows are observations.
- results - numeric matrix representing estimation results.
- log-likelihood - value of Log-Likelihood function.
- AIC - AIC value.
- errors_exp - random error expectation estimate.
- errors_var - random error variance estimate.
- dataframe - dataframe containing variables mentioned in formula without NA values.
- model_Lists - lists containing information about fixed parameters and parameters indexes in x1.
- n_obs - number of observations.
- z_latent - latent variable (signle index) estimates.
- z_prob - probabilities of positive outcome (i.e. 1) estimates.

References

A. Gallant and D. W. Nychka (1987) <doi:10.2307/1913241>

See Also

summary.hpaBinary, predict.hpaBinary, plot.hpaBinary, AIC.hpaBinary, logLik.hpaBinary

Examples

```
## Estimate survival probability on Titanic

library("titanic")

#Prepare data set converting
#all variables to numeric vectors
h <- data.frame("male" = as.numeric(titanic_train$Sex == "male"))
h$class_1 <- as.numeric(titanic_train$Pclass == 1)
h$class_2 <- as.numeric(titanic_train$Pclass == 2)
h$class_3 <- as.numeric(titanic_train$Pclass == 3)
h$sibl <- titanic_train$SibSp
h$survived <- titanic_train$Survived
h$age <- titanic_train$Age
h$parch <- titanic_train$Parch
h$fare <- titanic_train$Fare

#Estimate model parameters
model_hpa_1 <- hpaBinary(survived ~class_1 + class_2 +
male + age + sibl + parch + fare,
K = 3, data = h)
#get summary
summary(model_hpa_1)

#Get predicted probabilities
pred_hpa_1 <- predict(model_hpa_1)

#Calculate number of correct predictions
hpa_1_correct_0 <- sum((pred_hpa_1 < 0.5) & (model_hpa_1$dataframe$survived == 0))
hpa_1_correct_1 <- sum((pred_hpa_1 >= 0.5) & (model_hpa_1$dataframe$survived == 1))
hpa_1_correct <- hpa_1_correct_1 + hpa_1_correct_0

#Plot random errors density approximation
plot(model_hpa_1)



##Estimate parameters on data simulated from student distribution

library("mvtnorm")
set.seed(123)

#Simulate independent variables from normal distribution
```

```
n <- 5000
X <- rmvnorm(n=n, mean = c(0,0),
sigma = matrix(c(1,0.5,0.5,1), ncol=2))

#Simulate random errors from student distribution
epsilon <- rt(n, 5) * (3 / sqrt(5))

#Calculate latent and observable variables values
z_star <- 1 + X[, 1] + X[, 2] + epsilon
z <- as.numeric((z_star > 0))

#Store the results into dataframe
h <- as.data.frame(cbind(z,X))
names(h) <- c("z", "x1", "x2")

#Estimate model parameters
model <- hpaBinary(formula = z ~ x1 + x2, data=h, K = 4)
summary(model)

#Get predicted probabilities of 1 values
predict(model)

#Plot density function approximation
plot(model)
```

hpaML *Semi-nonparametric maximum likelihood estimation*

Description

This function performs semi-nonparametric maximum likelihood estimation via hermite polynomial densities approximation.

Usage

```
hpaML(
  x,
  pol_degrees = numeric(0),
  tr_left = numeric(0),
  tr_right = numeric(0),
  given_ind = logical(0),
  omit_ind = logical(0),
  x0 = numeric(0),
  cov_type = "sandwich",
  boot_iter = 100L,
  is_parallel = FALSE,
  opt_type = "optim",
  opt_control = NULL
)
```

Arguments

`x`	numeric matrix which rows are realizations of independend identically distributed random vectors while columns correspond to variables.
`pol_degrees`	non-negative integer vector of polynomial degrees.
`tr_left`	numeric matrix of left (lower) truncation limits. Note that `tr_right` rows are observations while variables are columns. If `tr_left` or `tr_right` is single row matrix then the same truncation limits would be applied to all observations that are determined by the first rows of these matrices.
`tr_right`	numeric matrix of right (upper) truncation limits. Note that `tr_right` rows are observations while variables are columns. If `tr_left` or `tr_right` is single row matrix then the same truncation limits would be applied to all observations that are determined by the first rows of these matrices.
`given_ind`	logical vector indicating wheather corresponding component is conditioned. By default it is a logical vector of FALSE values.
`omit_ind`	logical vector indicating wheather corresponding component is omitted. By default it is a logical vector of FALSE values.
`x0`	numeric vector of optimization routine initial values. Note that `x0=c(pol_coefficients[-1],mean,sd)` For `pol_coefficients`, `mean` and `sd` documentation see `dhpa` function.
`cov_type`	string value determinign 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 applied. If `cov_type = "sandwich"` (default) then sandwich covariance matrix estimator will be applied. If `cov_type = "bootstrap"` then bootstrap with `boot_iter` iterations will be applied. If `cov_type = "hessianFD"` or `cov_type = "sandwichFD"` then accurate but computationally demanding finite difference hessian approximation will be calculated for the inverse hessian and sandwich estimators correspondingly.
`boot_iter`	the number of bootstrap iterations for `cov_type = "bootstrap"` covariance matrix estimator type.
`is_parallel`	if TRUE then multiple cores will be used for some calculations. It usually provides speed advantage for large enough samples (about more than 1000 observations).
`opt_type`	string value determining the type of the optimization routine to be applied. The default is "optim" meaning that BFGS method from the `optim` function will be applied. If `opt_type = "GA"` then `ga` function will be additionally applied.
`opt_control`	a list containing arguments to be passed to the optimization routine depending on `opt_type` argument value. Please see details to get additional information.

Details

Semi-nonparametric (SNP) approach has been implemented via densities hermite polynomial approximation

Densities hermite polynomial approximation approach has been proposed by A. Gallant and D. W. Nychka in 1987. The main idea is to approximate unknown distribution density with hermite polynomial of degree pol_degree. In this framework hermite polynomial represents adjusted (to insure integration to 1) product of squared polynomial and normal distribution densities. Parameters mean and sd determine means and standard deviations of normal distribution density functions which are parts of this polynomial. For more information please refer to the literature listed below.

Parameters mean, sd, given_ind, omit_ind should have the same length as pol_degrees parameter.

The first polynomial coefficient (zero powers) set to identity constant for identification reasons.

The function calculates standard errors via sandwich estimator and significance levels are reported taking into account quasi maximum likelihood estimator (QMLE) asymptotic normality. If ones wants to switch from QMLE to semi-nonparametric estimator (SNPE) during hypothesis testing then covariance matrix should be reestimated using bootstrap.

This function maximizes (quasi) log-likelihood function via optim function setting it's method argument to "BFGS". If opt_type = "GA" then genetic algorithm from ga function will be additionally (after optim putting it's solution (par) to suggestions matrix) applied in order to perform global optimization. Note that global optimization takes much more time (usually minutes but sometimes hours or even days). The number of iterations and population size of the genetic algorithm will grow linearly along with the number of estimated parameters. If it is seems that global maximum has not been found than it is possible to continue the search restarting the function setting it's input argument x0 to x1 output value. Note that if cov_type = "bootstrap" then ga function will not be used for bootstrap iterations since it may be extremely time consuming.

If opt_type = "GA" then opt_control should be the list containing the values to be passed to ga function. It is possible to pass arguments lower, upper, popSize, pcrossover, pmutation, elitism, maxiter, suggestions, optim, optimArgs and seed. Note that it is possible to set population, selection, crossover and mutation arguments changing ga default parameters via gaControl function. These arguments information reported in ga. In order to provide manual values for lower and upper bounds please follow parameters ordering mentioned above for the x0 argument. If these bonds are not provided manually then they (except those related to the polynomial coefficients) will depend on the estimates obtained by local optimization via optim function (this estimates will be in the middle between lower and upper). Specifically for each sd parameter lower (upper) bound is 5 times lower (higher) then this parameter optim estimate. For each mean and regression coefficient parameter it's lower and upper bounds deviate from corresponding optim estimate by two absolute value of this estimate. Finally, lower and upper bounds for each polynomial coefficient are -10 and 10 correspondingly (do not depend on their optim estimates).

The following arguments are differ from their defaults in ga:

- pmutation = 0.2,
- optim = TRUE,
- optimArgs = list("method" = "Nelder-Mead","poptim" = 0.2,"pressel" = 0.5),
- seed = 8,
- elitism = 2 + round(popSize * 0.1).

The arguments popSize and maxiter of ga function have been set proportional to the number of estimated polynomial coefficients

- popSize = 10 + (prod(pol_degrees + 1) -1) * 2.
- maxiter = 50 * (prod(pol_degrees + 1))

Value

This function returns an object of class "hpaML".

An object of class "hpaML" is a list containing the following components:

- optim - optim function output. If opt_type = "GA" then it is the list containing optim and ga functions outputs.
- x1 - numeric vector of distribution parameters estimates.
- mean - density function mean vector estimate.
- sd - density function sd vector estimate.
- pol_coefficients - polynomial coefficients estimates.
- tr_left - the same as tr_left input parameter.
- tr_right - the same as tr_right input parameter.
- omit_ind - the same as omit_ind input parameter.
- given_ind - the same as given_ind input parameter.
- cov_matrix - estimated parameters covariance matrix estimate.
- results - numeric matrix representing estimation results.
- log-likelihood - value of Log-Likelihood function.
- AIC - AIC value.
- data - the same as x input parameter but without NA observations.
- n_obs - number of observations.
- bootstrap - list where bootstrap estimation results are stored.

References

A. Gallant and D. W. Nychka (1987) <doi:10.2307/1913241>

See Also

summary.hpaML, predict.hpaML, AIC.hpaML, logLik.hpaML

Examples

```
## Approximate student (t) distribution

# Simulate 5000 realizations of student distribution with 5 degrees of freedom
n <- 5000
df <- 5
```

```
x <- matrix(rt(n, df), ncol = 1)
pol_degrees <- c(4)

# Apply pseudo maximum likelihood routine
ml_result <- hpa::hpaML(x = x, pol_degrees = pol_degrees)
summary(ml_result)

# Get predicted probabilites (density values) approximations
predict(ml_result)

## Approximate chi-squared distribution

# Simulate 5000 realizations of chi-squared distribution with 5 degrees of freedom
# Let's set lower truncation point at sample minimum realization
n <- 5000
df <- 5
x <- matrix(rchisq(n, df), ncol = 1)
pol_degrees <- c(1)
tr_left <- as.vector(min(x))
tr_right <- as.vector(max(x))

# Apply pseudo maximum likelihood routine
ml_result <- hpa::hpaML(x = x, pol_degrees = as.vector(pol_degrees),
tr_left = as.vector(tr_left),
tr_right = as.vector(tr_right))
summary(ml_result)

# Get predicted probabilites (density values) approximations
predict(ml_result)

## Approximate multivariate student (t) distribution
## Note that calculations may take up to a minute

# Simulate 5000 realizations of three dimensional student distribution with 5 degrees of freedom
library("mvtnorm")
cov_mat <- matrix(c(1, 0.25, -0.25, 0.25, 1, 0.25, -0.25, 0.25, 1), ncol = 3)
x <- rmvt(n = 5000, sigma = cov_mat, df = 5)

# Estimate approximating joint distribution parameters
model <- hpaML(x = x, pol_degrees = c(1,1,1))

# Get summary
summary(model)

# Get predicted values for joint density function
predict(model)

## Approximate student (t) distribution and plot densities approximated
## under different hermite polynomial degrees against
## true density (of student distribution)

# Simulate 5000 realizations of t-distribution with 5 degrees of freedom
n <- 5000
```

```
df <- 5
x <- matrix(rt(n, df), ncol=1)

# Apply pseudo maximum likelihood routine
# Create matrix of lists where i-th element contains hpaML results for K=i
ml_result <- matrix(list(), 4, 1)
for(i in 1:4)
{
 ml_result[[i]] <- hpa::hpaML(x = x, pol_degrees = i)
}

# Generate test values
test_values <- seq(qt(0.001, df), qt(0.999, df), 0.001)
n0 <- length(test_values)

# t-distribution density function at test values points
true_pred <- dt(test_values, df)

# Create matrix of lists where i-th element contains densities predictions for K=i
PGN_pred <- matrix(list(), 4, 1)
for(i in 1:4)
{
  PGN_pred[[i]] <- predict(object = ml_result[[i]],
                           newdata = matrix(test_values, ncol=1))
}
# Plot the result
library("ggplot2")

# prepare the data
h <- data.frame("values" = rep(test_values,5),
                "predictions" = c(PGN_pred[[1]],PGN_pred[[2]],
                                  PGN_pred[[3]],PGN_pred[[4]],
                                  true_pred),
                "Density" = c(
                  rep("K=1",n0), rep("K=2",n0),
                  rep("K=3",n0), rep("K=4",n0),
                  rep("t-distribution",n0))
                  )

# build the plot
ggplot(h, aes(values, predictions)) + geom_point(aes(color = Density)) +
  theme_minimal() + theme(legend.position = "top", text = element_text(size=26),
                legend.title=element_text(size=20), legend.text=element_text(size=28)) +
  guides(colour = guide_legend(override.aes = list(size=10))
  )

# Get informative estimates summary for K=4 (minimal AIC)
summary(ml_result[[4]])
```

hpaSelection *Perform semi-nonparametric selection model estimation*

Description

This function performs semi-nonparametric selection model estimation via hermite polynomial densities approximation.

Usage

```
hpaSelection(
  selection,
  outcome,
  data,
  z_K = 1L,
  y_K = 1L,
  pol_elements = 3L,
  is_Newey = FALSE,
  x0 = numeric(0),
  is_Newey_loocv = FALSE,
  z_sd_fixed = -1,
  cov_type = "sandwich",
  boot_iter = 100L,
  is_parallel = FALSE,
  opt_type = "optim",
  opt_control = NULL
)
```

Arguments

`selection`	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the selection equation form.
`outcome`	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the outcome equation form.
`data`	data frame containing the variables in the model.
`z_K`	non-negative integer representing polynomial degree related to selection equation.
`y_K`	non-negative integer representing polynomial degree related to outcome equation.
`pol_elements`	number of conditional expectation approximating terms for Newey's method. If `is_Newey_loocv` is `TRUE` then determines maximum number of these terms during leave-one-out cross-validation.
`is_Newey`	logical; if TRUE then returns only Newey's method estimation results (default value is FALSE).
`x0`	numeric vector of optimization routine initial values. Note that `x0=c(pol_coefficients[-1],mean,sd,`

`is_Newey_loocv`	logical; if TRUE then number of conditional expectation approximating terms for Newey's method will be selected based on leave-one-out cross-validation criteria iterating througt 0 to pol_elements number of these terms.
`z_sd_fixed`	positive value that is fixed sigma parameter for selection equation.
`cov_type`	string value determinign 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 applied. If `cov_type = "sandwich"` (default) then sandwich covariance matrix estimator will be applied. If `cov_type = "bootstrap"` then bootstrap with `boot_iter` iterations will be applied. If `cov_type = "hessianFD"` or `cov_type = "sandwichFD"` then accurate but computationally demanding finite difference hessian approximation will be calculated for the inverse hessian and sandwich estimators correspondingly.
`boot_iter`	the number of bootstrap iterations for `cov_type = "bootstrap"` covariance matrix estimator type.
`is_parallel`	if TRUE then multiple cores will be used for some calculations. It usually provides speed advantage for large enough samples (about more than 1000 observations).
`opt_type`	string value determining the type of the optimization routine to be applied. The default is "optim" meaning that BFGS method from the `optim` function will be applied. If `opt_type = "GA"` then `ga` function will be additionally applied.
`opt_control`	a list containing arguments to be passed to the optimization routine depending on `opt_type` argument value. Please see details to get additional information.

Details

Semi-nonparametric (SNP) approach has been implemented via densities hermite polynomial approximation

Densities hermite polynomial approximation approach has been proposed by A. Gallant and D. W. Nychka in 1987. The main idea is to approximate unknown distribution density with hermite polynomial of degree `pol_degree`. In this framework hermite polynomial represents adjusted (to insure integration to 1) product of squared polynomial and normal distribution densities. Parameters `mean` and `sd` determine means and standard deviations of normal distribution density functions which are parts of this polynomial. For more information please refer to the literature listed below.

Parameters `mean`, `sd`, `given_ind`, `omit_ind` should have the same length as `pol_degrees` parameter.

The first polynomial coefficient (zero powers) set to identity constant for identification reasons.

Note that coefficient for the first independent variable in `selection` will be fixed to 1.

Parameter `sd` will be scale adjusted in order to provide better initial point for optimization routine. Please, extract `sd` adjusted value from this function's output list.

All variables mentioned in `selection` and `outcome` should be numeric vectors.

The function calculates standard errors via sandwich estimator and significance levels are reported taking into account quasi maximum likelihood estimator (QMLE) asymptotic normality. If ones wants to switch from QMLE to semi-nonparametric estimator (SNPE) during hypothesis testing then covariance matrix should be reestimated using bootstrap.

Initial values for optimization routine are obtained throught Newey method (see the reference below).

Note that selection equation dependent variables should have exactly two levels (0 and 1) where "0" states for the selection results which leads to unobservable values of dependent variable in outcome equation.

This function maximizes (quasi) log-likelihood function via `optim` function setting it's `method` argument to "BFGS". If `opt_type = "GA"` then genetic algorithm from `ga` function will be additionally (after `optim` putting it's solution (`par`) to `suggestions` matrix) applied in order to perform global optimization. Note that global optimization takes much more time (usually minutes but sometimes hours or even days). The number of iterations and population size of the genetic algorithm will grow linearly along with the number of estimated parameters. If it is seems that global maximum has not been found than it is possible to continue the search restarting the function setting it's input argument `x0` to `x1` output value. Note that if `cov_type = "bootstrap"` then `ga` function will not be used for bootstrap iterations since it may be extremely time consuming.

If `opt_type = "GA"` then `opt_control` should be the list containing the values to be passed to `ga` function. It is possible to pass arguments `lower`, `upper`, `popSize`, `pcrossover`, `pmutation`, `elitism`, `maxiter`, `suggestions`, `optim`, `optimArgs` and `seed`. Note that it is possible to set `population`, `selection`, `crossover` and `mutation` arguments changing `ga` default parameters via `gaControl` function. These arguments information reported in `ga`. In order to provide manual values for `lower` and `upper` bounds please follow parameters ordering mentioned above for the `x0` argument. If these bonds are not provided manually then they (except those related to the polynomial coefficients) will depend on the estimates obtained by local optimization via `optim` function (this estimates will be in the middle between `lower` and `upper`). Specifically for each sd parameter `lower` (`upper`) bound is 5 times lower (higher) then this parameter `optim` estimate. For each mean and regression coefficient parameter it's lower and upper bounds deviate from corresponding `optim` estimate by two absolute value of this estimate. Finally, lower and upper bounds for each polynomial coefficient are `-10` and `10` correspondingly (do not depend on their `optim` estimates).

The following arguments are differ from their defaults in `ga`:

- `pmutation = 0.2`,
- `optim = TRUE`,
- `optimArgs = list("method" = "Nelder-Mead","poptim" = 0.2,"pressel" = 0.5)`,
- `seed = 8`,
- `elitism = 2 + round(popSize * 0.1)`.

Let's denote by `n_reg` the number of regressors included to the `selection` and `outcome` formulas. The arguments `popSize` and `maxiter` of `ga` function have been set proportional to the number of estimated polynomial coefficients and independent variables:

- `popSize = 10 + 5 * (z_K + 1) * (y_K + 1) + 2 * n_reg`
- `maxiter = 50 * (z_K + 1) * (y_K + 1) + 10 * n_reg`

Value

This function returns an object of class "hpaSelection".

An object of class "hpaSelection" is a list containing the following components:

- optim - optim function output. If opt_type = "GA" then it is the list containing optim and ga functions outputs.
- x1 - numeric vector of distribution parameters estimates.
- Newey - list containing information concerning Newey's method estimation results.
- z_mean - estimate of the hermite polynomial mean parameter related to selection equation random error marginal distribution.
- y_mean - estimate of the hermite polynomial mean parameter related to outcome equation random error marginal distribution.
- z_sd - adjusted value of sd parameter related to selection equation random error marginal distribution.
- y_sd - estimate of the hermite polynomial sd parameter related to outcome equation random error marginal distribution.
- pol_coefficients - polynomial coefficients estimates.
- pol_degrees - numeric vector which first element is z_K and the second is y_K.
- z_coef - selection equation regression coefficients estimates.
- y_coef - outcome equation regression coefficients estimates.
- cov_matrix - estimated parameters covariance matrix estimate.
- results - numeric matrix representing estimation results.
- log-likelihood - value of Log-Likelihood function.
- AIC - AIC value.
- re_moments - list which contains information about random errors expectations, variances and correlation.
- data_List - list containing model variables and their partiotion according to outcome and selection equations.
- n_obs - number of observations.
- ind_List - list which contains information about parameters indexes in x1.
- selection_formula - the same as selection input parameter.
- outcome_formula - the same as outcome input parameter.

Abovementioned list Newey has class "hpaNewey" and contains the following components:

- y_coef - regression coefficients estimates (except constant term which is part of conditional expectation approximating polynomial).
- z_coef - regression coefficients estimates related to selection equation.
- constant_biased - biased estimate of constant term.
- inv_mills - inverse mills rations estimates and their powers (including constant).
- inv_mills_coef - coefficients related to inv_mills.
- pol_elements - the same as pol_elements input parameter. However if is_Newey_loocv is TRUE then it will equal to the number of conditional expectation approximating terms for Newey's method which minimize leave-one-out cross-validation criteria.
- outcome_exp_cond - dependend variable conditional expectation estimates.

- `selection_exp` - selection equation random error expectation estimate.
- `selection_var` - selection equation random error variance estimate.
- `hpaBinaryModel` - object of class "hpaBinary" which contains selection equation estimation results.

Abovementioned list `re_moments` contains the following components:

- `selection_exp` - selection equation random errors expectation estimate.
- `selection_var` - selection equation random errors variance estimate.
- `outcome_exp` - outcome equation random errors expectation estimate.
- `outcome_var` - outcome equation random errors variance estimate.
- `errors_covariance` - outcome and selection equation random errors covariance estimate.
- `rho` - outcome and selection equation random errors correlation estimate.
- `rho_std` - outcome and selection equation random errors correlation estimator standard error estimate.

References

A. Gallant and D. W. Nychka (1987) <doi:10.2307/1913241>

W. K. Newey (2009) <https://doi.org/10.1111/j.1368-423X.2008.00263.x>

Mroz T. A. (1987) <doi:10.2307/1911029>

See Also

summary.hpaSelection, predict.hpaSelection, plot.hpaSelection, AIC.hpaSelection, logLik.hpaSelection

Examples

```
## Let's estimate wage equation accounting for non-random selection.
## See the reference to Mroz TA (1987) to get additional details about
## the data this examples use

# Prepare data
library("sampleSelection")
data("Mroz87")
h = data.frame("kids" = as.numeric(Mroz87$kids5 + Mroz87$kids618 > 0),
"age" = as.numeric(Mroz87$age),
"faminc" = as.numeric(Mroz87$faminc),
"educ" = as.numeric(Mroz87$educ),
"exper" = as.numeric(Mroz87$exper),
"city" = as.numeric(Mroz87$city),
"wage" = as.numeric(Mroz87$wage),
"lfp" = as.numeric(Mroz87$lfp))

# Estimate model parameters
model <- hpaSelection(selection = lfp ~ educ + age + I(age ^ 2) +
                                  kids + faminc,
                      outcome = log(wage) ~ exper + I(exper ^ 2) +
                                            educ + city,
                                 z_K = 1, y_K = 3, data = h,
                                 pol_elements = 3, is_Newey_loocv = TRUE)
summary(model)

# Plot outcome equation random errorrs density
plot(model, is_outcome = TRUE)
# Plot selection equation random errorrs density
plot(model, is_outcome = FALSE)


## Estimate semi-nonparametric sample selection model
## parameters on simulated data given chi-squared random errors


set.seed(100)
library("mvtnorm")

# Sample size

n <- 1000

# Simulate independent variables
X_rho <- 0.5
X_sigma <- matrix(c(1,X_rho,X_rho,X_rho,1,X_rho,X_rho,X_rho,1), ncol=3)
X <- rmvnorm(n=n, mean = c(0,0,0),
             sigma = X_sigma)

# Simulate random errors
epsilon <- matrix(0, n, 2)
epsilon_z_y <- rchisq(n, 5)
epsilon[, 1] <- (rchisq(n, 5) + epsilon_z_y) * (sqrt(3/20)) - 3.8736
epsilon[, 2] <- (rchisq(n, 5) + epsilon_z_y) * (sqrt(3/20)) - 3.8736
# Simulate selection equation
z_star <- 1 + 1 * X[,1] + 1 * X[,2] + epsilon[,1]
z <- as.numeric((z_star > 0))

# Simulate outcome equation
y_star <- 1 + 1 * X[,1] + 1 * X[,3] + epsilon[,2]
z <- as.numeric((z_star > 0))
y <- y_star
y[z==0] <- NA
h <- as.data.frame(cbind(z, y, X))
names(h) <- c("z", "y", "x1", "x2", "x3")

# Estimate parameters
model <- hpaSelection(selection = z ~ x1 + x2,
                      outcome = y ~ x1 + x3,
                      data = h, z_K = 1, y_K = 3)
summary(model)

# Get conditional predictions for outcome equation
```

```
model_pred_c <- predict(model,is_cond = TRUE)
# Conditional predictions y|z=1
model_pred_c$y_1
# Conditional predictions y|z=0
model_pred_c$y_0

# Get unconditional predictions for outcome equation
model_pred_u <- predict(model,is_cond = FALSE)
model_pred_u$y

# Get conditional predictions for selection equation
# Note that for z=0 these predictions are NA
predict(model, is_cond = TRUE, is_outcome = FALSE)
# Get unconditional predictions for selection equation
predict(model, is_cond = FALSE, is_outcome = FALSE)
```

---

ihpa *Interval distribution function hermite polynomial approximation*

Description

This function calculates interval distribution function hermite polynomial approximation.

Usage

```
ihpa(
  x_lower = matrix(1, 1),
  x_upper = matrix(1, 1),
  pol_coefficients = numeric(0),
  pol_degrees = numeric(0),
  given_ind = logical(0),
  omit_ind = logical(0),
  mean = numeric(0),
  sd = numeric(0),
  is_parallel = FALSE
)
```

Arguments

`x_lower`	numeric matrix of lower integration limits. Note that `x_lower` rows are observations while variables are columns.
`x_upper`	numeric matrix of upper integration limits. Note that `x_upper` rows are observations while variables are columns.
`pol_coefficients`	numeric vector of polynomial coefficients.

pol_degrees	non-negative integer vector of polynomial degrees.
given_ind	logical vector indicating wheather corresponding component is conditioned. By default it is a logical vector of FALSE values.
omit_ind	logical vector indicating wheather corresponding component is omitted. By default it is a logical vector of FALSE values.
mean	numeric vector of expected values.
sd	positive numeric vector of standard deviations.
is_parallel	if TRUE then multiple cores will be used for some calculations. It usually provides speed advantage for large enough samples (about more than 1000 observations).

Details

Interval distribution function represents probability that random vector components will be greater then values given in `x_lower` and lower then values that are in `x_upper`.

Densities hermite polynomial approximation approach has been proposed by A. Gallant and D. W. Nychka in 1987. The main idea is to approximate unknown distribution density with hermite polynomial of degree `pol_degree`. In this framework hermite polynomial represents adjusted (to insure integration to 1) product of squared polynomial and normal distribution densities. Parameters `mean` and `sd` determine means and standard deviations of normal distribution density functions which are parts of this polynomial. For more information please refer to the literature listed below.

Parameters `mean`, `sd`, `given_ind`, `omit_ind` should have the same length as `pol_degrees` parameter.

Value

This function returns interval distribution function hermite polynomial approximation at point `x`.

References

A. Gallant and D. W. Nychka (1987) <doi:10.2307/1913241>

Examples

```
##Let's approximate some three random variables joint interval distribution function (idf)
##at lower and upper points (0,1, 0.2, 0.3) and (0,4, 0.5, 0.6) correspondingly
##with hermite polynomial of (1,2,3) degrees which polynomial coefficients equals 1 except
##coefficient related to x1*(x^3) polynomial element which equals 2.
##Also suppose that normal density related mean vector equals (1.1, 1.2, 1.3) while
##standard deviations vector is (2.1, 2.2, 2.3).

##Prepare initial values
x_lower <- matrix(c(0.1, 0.2, 0.3), nrow=1)
x_upper <- matrix(c(0.4, 0.5, 0.6), nrow=1)
mean <- c(1.1, 1.2, 1.3)
sd <- c(2.1, 2.2, 2.3)
pol_degrees <- c(1, 2, 3)
```

```
#Create polynomial powers and indexes correspondence matrix
pol_ind <- polynomialIndex(pol_degrees)

#Set all polynomial coefficients to 1
pol_coefficients <- rep(1, ncol(pol_ind))
pol_degrees_n <- length(pol_degrees)

#Assign coefficient 2 to the polynomial element(x1 ^ 1)*(x2 ^ 0)*(x3 ^ 2)
pol_coefficients[which(colSums(pol_ind == c(1, 0, 2)) == pol_degrees_n)] <- 2

#Visualize correspondence between polynomial elements and their coefficients
as.data.frame(rbind(pol_ind, pol_coefficients),
row.names = c("x1 power", "x2 power", "x3 power", "coefficients"),
optional = TRUE)
printPolynomial(pol_degrees, pol_coefficients)

#Calculate idf approximation at points x_lower and x_upper
ihpa(x_lower = x_lower, x_upper = x_upper,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd)

#Condition second component to be 0.7
#Substitute x second component with conditional value 0.7
x_upper <- matrix(c(0.4, 0.7, 0.6), nrow = 1)

#Set TRUE to the second component indicating that it is conditioned
given_ind <- c(FALSE, TRUE, FALSE)

#Calculate conditional(on x2 = 0.5) idf approximation at points x_lower and x_upper
ihpa(x_lower = x_lower, x_upper = x_upper,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd,
given_ind = given_ind)

#Consider third component marginal distribution
#conditioned on the second component 0.7 value
#Set TRUE to the first component indicating that it is omitted
omit_ind <- c(TRUE, FALSE, FALSE)

#Calculate conditional (on x2=0.5) marginal (for x3) idf approximation at points x_lower and x_upper
ihpa(x_lower = x_lower, x_upper = x_upper,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd,
given_ind = given_ind, omit_ind = omit_ind)
```

`ihpaDiff` *Calculate gradient of interval distribution function hermite polynomial approximation*

Description

This function calculates interval distribution function hermite polynomial approximation.

Usage

```
ihpaDiff(
  x_lower = matrix(1, 1),
  x_upper = matrix(1, 1),
  pol_coefficients = numeric(0),
  pol_degrees = numeric(0),
  given_ind = logical(0),
  omit_ind = logical(0),
  mean = numeric(0),
  sd = numeric(0),
  type = "pol_coefficients",
  is_parallel = FALSE
)
```

Arguments

`x_lower`	numeric matrix of lower integration limits. Note that `x_lower` rows are observations while variables are columns.
`x_upper`	numeric matrix of upper integration limits. Note that `x_upper` rows are observations while variables are columns.
`pol_coefficients`	numeric vector of polynomial coefficients.
`pol_degrees`	non-negative integer vector of polynomial degrees.
`given_ind`	logical vector indicating wheather corresponding component is conditioned. By default it is a logical vector of FALSE values.
`omit_ind`	logical vector indicating wheather corresponding component is omitted. By default it is a logical vector of FALSE values.
`mean`	numeric vector of expected values.
`sd`	positive numeric vector of standard deviations.
`type`	determines the partial derivatives to be included into gradient. Currently `type="pol_coefficients"` is the only available option (default) meaning that the gradient will contain partial derivatives respect to polynomial coefficients listed in the same order as `pol_coefficients`.
`is_parallel`	if TRUE then multiple cores will be used for some calculations. It usually provides speed advantage for large enough samples (about more than 1000 observations).

Details

Interval distribution function represents probability that random vector components will be greater then values given in `x_lower` and lower then values that are in `x_upper`.

Densities hermite polynomial approximation approach has been proposed by A. Gallant and D. W. Nychka in 1987. The main idea is to approximate unknown distribution density with hermite polynomial of degree `pol_degree`. In this framework hermite polynomial represents adjusted (to insure integration to 1) product of squared polynomial and normal distribution densities. Parameters `mean` and `sd` determine means and standard deviations of normal distribution density functions which are parts of this polynomial. For more information please refer to the literature listed below.

Parameters `mean`, `sd`, `given_ind`, `omit_ind` should have the same length as `pol_degrees` parameter.

If `x` has more then one row then the output will be jacobian matrix where rows are gradients.

Value

This function returns gradient of interval distribution function hermite polynomial approximation at point `x`. Gradient elements are determined by the `type` argument.

References

A. Gallant and D. W. Nychka (1987) <doi:10.2307/1913241>

Examples

```
## Let's approximate some three random variables joint interval distribution function (idf)
## at lower and upper points (0,1, 0.2, 0.3) and (0,4, 0.5, 0.6) correspondingly
## with hermite polynomial of (1,2,3) degrees which polynomial coefficients equals 1 except
## coefficient related to x1*(x^3) polynomial element which equals 2.
## Also suppose that normal density related mean vector equals (1.1, 1.2, 1.3) while
## standard deviations vector is (2.1, 2.2, 2.3).
## In this example let's calculate interval distribution approximating function gradient
## respect to polynomial coefficients.

# Prepare initial values
x_lower <- matrix(c(0.1, 0.2, 0.3), nrow=1)
x_upper <- matrix(c(0.4, 0.5, 0.6), nrow=1)
mean <- c(1.1, 1.2, 1.3)
sd <- c(2.1, 2.2, 2.3)
pol_degrees <- c(1, 2, 3)

# Create polynomial powers and indexes correspondence matrix
pol_ind <- polynomialIndex(pol_degrees)

# Set all polynomial coefficients to 1
pol_coefficients <- rep(1, ncol(pol_ind))
pol_degrees_n <- length(pol_degrees)

# Assign coefficient 2 to the polynomial element(x1 ^ 1)*(x2 ^ 0)*(x3 ^ 2)
pol_coefficients[which(colSums(pol_ind == c(1, 0, 2)) == pol_degrees_n)] <- 2

# Visualize correspondence between polynomial elements and their coefficients
as.data.frame(rbind(pol_ind, pol_coefficients),
 row.names = c("x1 power", "x2 power", "x3 power", "coefficients"),
 optional = TRUE)
```

```
printPolynomial(pol_degrees, pol_coefficients)

# Calculate idf approximation gradient respect to
# polynomial coefficients at points x_lower and x_upper
ihpaDiff(x_lower = x_lower, x_upper = x_upper,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd)

# Condition second component to be 0.7
# Substitute x second component with conditional value 0.7
x_upper <- matrix(c(0.4, 0.7, 0.6), nrow = 1)

# Set TRUE to the second component indicating that it is conditioned
given_ind <- c(FALSE, TRUE, FALSE)

# Calculate conditional(on x2 = 0.5) idf approximation
# respect to polynomial coefficients at points x_lower and x_upper
ihpaDiff(x_lower = x_lower, x_upper = x_upper,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd,
given_ind = given_ind)

# Consider third component marginal distribution
# conditioned on the second component 0.7 value
# Set TRUE to the first component indicating that it is omitted
omit_ind <- c(TRUE, FALSE, FALSE)

# Calculate conditional (on x2=0.5) marginal (for x3) idf approximation
# respect to polynomial coefficients at points x_lower and x_upper
ihpaDiff(x_lower = x_lower, x_upper = x_upper,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd,
given_ind = given_ind, omit_ind = omit_ind)
```

itrhpa *Truncated interval distribution function hermite polynomial approximation for truncated distribution*

Description

This function calculates truncated interval distribution function hermite polynomial approximation for truncated distribution.

Usage

```
itrhpa(
  x_lower = matrix(1, 1),
  x_upper = matrix(1, 1),
  tr_left = matrix(1, 1),
  tr_right = matrix(1, 1),
  pol_coefficients = numeric(0),
  pol_degrees = numeric(0),
  given_ind = logical(0),
  omit_ind = logical(0),
  mean = numeric(0),
  sd = numeric(0),
  is_parallel = FALSE
)
```

Arguments

x_lower	numeric matrix of lower integration limits. Note that x_lower rows are observations while variables are columns.
x_upper	numeric matrix of upper integration limits. Note that x_upper rows are observations while variables are columns.
tr_left	numeric matrix of left (lower) truncation limits. Note that tr_right rows are observations while variables are columns. If tr_left or tr_right is single row matrix then the same truncation limits would be applied to all observations that are determined by the first rows of these matrices.
tr_right	numeric matrix of right (upper) truncation limits. Note that tr_right rows are observations while variables are columns. If tr_left or tr_right is single row matrix then the same truncation limits would be applied to all observations that are determined by the first rows of these matrices.
pol_coefficients	numeric vector of polynomial coefficients.
pol_degrees	non-negative integer vector of polynomial degrees.
given_ind	logical vector indicating wheather corresponding component is conditioned. By default it is a logical vector of FALSE values.
omit_ind	logical vector indicating wheather corresponding component is omitted. By default it is a logical vector of FALSE values.
mean	numeric vector of expected values.
sd	positive numeric vector of standard deviations.
is_parallel	if TRUE then multiple cores will be used for some calculations. It usually provides speed advantage for large enough samples (about more than 1000 observations).

Details

Interval distribution function represents probability that random vector components will be greater then values given in `x_lower` and lower then values that are in `x_upper`.

Densities hermite polynomial approximation approach has been proposed by A. Gallant and D. W. Nychka in 1987. The main idea is to approximate unknown distribution density with hermite polynomial of degree `pol_degree`. In this framework hermite polynomial represents adjusted (to insure integration to 1) product of squared polynomial and normal distribution densities. Parameters `mean` and `sd` determine means and standard deviations of normal distribution density functions which are parts of this polynomial. For more information please refer to the literature listed below.

Parameters mean, sd, given_ind, omit_ind should have the same length as pol_degrees parameter.

Value

This function returns interval distribution function (idf) hermite polynomial approximation at point x for truncated distribution.

References

A. Gallant and D. W. Nychka (1987) <doi:10.2307/1913241>

Examples

```
##Let's approximate some three truncated random variables joint interval distribution function
##at lower and upper points (0,1, 0.2, 0.3) and  (0,4, 0.5, 0.6) correspondingly
##with hermite polynomial of (1,2,3) degrees which polynomial coefficients equals 1 except
##coefficient related to x1*(x^3) polynomial element which equals 2. Also suppose that normal
##density related mean vector equals (1.1, 1.2, 1.3) while standard deviations vector is
##(2.1, 2.2, 2.3). Suppose that lower and upper truncation are (-1.1,-1.2,-1.3) and
##(1.1,1.2,1.3) correspondingly.

#Prepare initial values
x_lower <- matrix(c(0.1, 0.2, 0.3), nrow=1)
x_upper <- matrix(c(0.4, 0.5, 0.6), nrow=1)
tr_left = matrix(c(-1.1,-1.2,-1.3), nrow = 1)
tr_right = matrix(c(1.1,1.2,1.3), nrow = 1)
mean <- c(1.1, 1.2, 1.3)
sd <- c(2.1, 2.2, 2.3)
pol_degrees <- c(1, 2, 3)

#Create polynomial powers and indexes correspondence matrix
pol_ind <- polynomialIndex(pol_degrees)
#Set all polynomial coefficients to 1
pol_coefficients <- rep(1, ncol(pol_ind))
pol_degrees_n <- length(pol_degrees)

#Assign coefficient 2 to the polynomial element(x1 ^ 1)*(x2 ^ 0)*(x3 ^ 2)
pol_coefficients[which(colSums(pol_ind == c(1, 0, 2)) == pol_degrees_n)] <- 2
#Visualize correspondence between polynomial elements and their coefficients
as.data.frame(rbind(pol_ind, pol_coefficients),
row.names = c("x1 power", "x2 power", "x3 power", "coefficients"),
optional = TRUE)
printPolynomial(pol_degrees, pol_coefficients)

#Calculate idf approximation at points x_lower and x_upper
itrhpa(x_lower = x_lower, x_upper = x_upper,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd,
    tr_left = tr_left, tr_right = tr_right)

#Condition second component to be 0.7
```

```
#Substitute x second component with conditional value 0.7
x_upper <- matrix(c(0.4, 0.7, 0.6), nrow = 1)
#Set TRUE to the second component indicating that it is conditioned
given_ind <- c(FALSE, TRUE, FALSE)

#Calculate conditional(on x2 = 0.5) idf approximation at points x_lower and x_upper
itrhpa(x_lower = x_lower, x_upper = x_upper,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd,
given_ind = given_ind,
    tr_left = tr_left, tr_right = tr_right)

#Consider third component marginal distribution
#conditioned on the second component 0.7 value
#Set TRUE to the first component indicating that it is omitted
omit_ind <- c(TRUE, FALSE, FALSE)

#Calculate conditional (on x2=0.5) marginal (for x3) idf approximation at points x_lower and x_upper
itrhpa(x_lower = x_lower, x_upper = x_upper,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd,
given_ind = given_ind, omit_ind = omit_ind,
    tr_left = tr_left, tr_right = tr_right)
```

logLik.hpaBinary — *Calculates log-likelihood for "hpaBinary" object*

Description

This function calculates log-likelihood for "hpaBinary" object

Usage

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

Arguments

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

logLik.hpaML *Calculates log-likelihood for "hpaML" object*

Description

This function calculates log-likelihood for "hpaML" object

Usage

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

Arguments

object	Object of class "hpaML"
...	further arguments (currently ignored)

logLik.hpaSelection *Calculates log-likelihood for "hpaSelection" object*

Description

This function calculates log-likelihood for "hpaSelection" object

Usage

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

Arguments

object	Object of class "hpaSelection"
...	further arguments (currently ignored)

`logLik_hpaBinary` *Calculates log-likelihood for "hpaBinary" object*

Description

This function calculates log-likelihood for "hpaBinary" object

Usage

```
logLik_hpaBinary(object)
```

Arguments

`object`	Object of class "hpaBinary"

`logLik_hpaML` *Calculates log-likelihood for "hpaML" object*

Description

This function calculates log-likelihood for "hpaML" object

Usage

```
logLik_hpaML(object)
```

Arguments

`object`	Object of class "hpaML"

`logLik_hpaSelection` *Calculates log-likelihood for "hpaSelection" object*

Description

This function calculates log-likelihood for "hpaSelection" object

Usage

```
logLik_hpaSelection(object)
```

Arguments

`object`	Object of class "hpaSelection"

`mecdf` *Calculates multivariate empirical cumulative distribution function*

Description

This function calculates multivariate empirical cumulative distribution function at each point of the sample

Usage

```
mecdf(x)
```

Arguments

`x`	numeric matrix which rows are observations

`normalMoment` *Calculate k-th order moment of normal distribution*

Description

This function iteratively calculates k-th order moment of normal distribution.

Usage

```
normalMoment(
  k = 0L,
  mean = 0,
  sd = 1,
  return_all_moments = FALSE,
  is_validation = TRUE,
  is_central = FALSE
)
```

Arguments

`k`	non-negative integer moment order.
`mean`	numeric expected value.
`sd`	positive numeric standard deviation.
`return_all_moments`	logical; if `TRUE`, function returns (k+1)-dimensional numeric vector of moments of normaly distributed random variable with mean = `mean` and standard deviation = `sd`. Note that i-th vector's component value corresponds to the (i-1)-th moment.
`is_validation`	bool value indicating whether function input arguments should be validated. Set it to FALSE for slight perfomance boost (default value is TRUE).
`is_central`	logical; if `TRUE`, then central moments will be calculated.

Details

This function estimates `k`-th order moment of normal distribution which mean equals to `mean` and standard deviation equals to `sd`.

Note that parameter `k` value automatically converts to integer. So passing non-integer `k` value will not cause any errors but the calculations will be performed for rounded `k` value only.

Value

This function returns `k`-th order moment of normal distribution which mean equals to `mean` and standard deviation is `sd`. If `return_all_moments` is `TRUE` then see this argument description above for output details.

Examples

```
## Calculate 5-th order moment of normal random variable which
## mean equals to 3 and standard deviation is 5.

# 5-th moment
normalMoment(k = 5, mean = 3, sd = 5)

# (0-5)-th moments
normalMoment(k = 5, mean = 3, sd = 5, return_all_moments = TRUE)
```

---

phpa *Distribution function hermite polynomial approximation*

Description

This function calculates cumulative distribution function hermite polynomial approximation.

Usage

```
phpa(
  x = matrix(1, 1),
  pol_coefficients = numeric(0),
  pol_degrees = numeric(0),
  given_ind = logical(0),
  omit_ind = logical(0),
  mean = numeric(0),
  sd = numeric(0),
  is_parallel = FALSE
)
```

Arguments

`x`	numeric matrix of cumulative distribution function arguments. Note that `x` rows are observations while variables are columns.
`pol_coefficients`	numeric vector of polynomial coefficients.
`pol_degrees`	non-negative integer vector of polynomial degrees.
`given_ind`	logical vector indicating wheather corresponding component is conditioned. By default it is a logical vector of FALSE values.
`omit_ind`	logical vector indicating wheather corresponding component is omitted. By default it is a logical vector of FALSE values.
`mean`	numeric vector of expected values.
`sd`	positive numeric vector of standard deviations.
`is_parallel`	if TRUE then multiple cores will be used for some calculations. It usually provides speed advantage for large enough samples (about more than 1000 observations).

Details

Densities hermite polynomial approximation approach has been proposed by A. Gallant and D. W. Nychka in 1987. The main idea is to approximate unknown distribution density with hermite polynomial of degree `pol_degree`. In this framework hermite polynomial represents adjusted (to insure integration to 1) product of squared polynomial and normal distribution densities. Parameters `mean` and `sd` determine means and standard deviations of normal distribution density functions which are parts of this polynomial. For more information please refer to the literature listed below.

Parameters `mean`, `sd`, `given_ind`, `omit_ind` should have the same length as `pol_degrees` parameter.

Value

This function returns cumulative distribution function hermite polynomial approximation at point `x`.

References

A. Gallant and D. W. Nychka (1987) <doi:10.2307/1913241>

Examples

```
##Let's approximate some three random variables joint cumulative distribution function (cdf)
##at point (0,1, 0.2, 0.3)
##with hermite polynomial of (1,2,3) degrees which polynomial coefficients equals 1 except
##coefficient related to x1*(x^3) polynomial element which equals 2. Also suppose that normal
##density related mean vector equals (1.1, 1.2, 1.3) while standard deviations
##vector is (2.1, 2.2, 2.3).

##Prepare initial values
x <- matrix(c(0.1, 0.2, 0.3), nrow=1)
```

```
mean <- c(1.1, 1.2, 1.3)
sd <- c(2.1, 2.2, 2.3)
pol_degrees <- c(1, 2, 3)

#Create polynomial powers and indexes correspondence matrix
pol_ind <- polynomialIndex(pol_degrees)

#Set all polynomial coefficients to 1
pol_coefficients <- rep(1, ncol(pol_ind))
pol_degrees_n <- length(pol_degrees)

#Assign coefficient 2 to the polynomial element(x1 ^ 1)*(x2 ^ 0)*(x3 ^ 2)
pol_coefficients[which(colSums(pol_ind == c(1, 0, 2)) == pol_degrees_n)] <- 2

#Visualize correspondence between polynomial elements and their coefficients
as.data.frame(rbind(pol_ind, pol_coefficients),
row.names = c("x1 power", "x2 power", "x3 power", "coefficients"),
optional = TRUE)
printPolynomial(pol_degrees, pol_coefficients)

#Calculate cdf approximation at point x
phpa(x = x,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd)

#Condition second component to be 0.5
#Substitute x second component with conditional value 0.5
x <- matrix(c(0.1, 0.5, 0.3), nrow = 1)

#Set TRUE to the second component indicating that it is conditioned
given_ind <- c(FALSE, TRUE, FALSE)

#Calculate conditional(on x2 = 0.5) cdf approximation at point x
phpa(x = x,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd,
given_ind = given_ind)

#Consider third component marginal distribution
#conditioned on the second component 0.5 value

#Set TRUE to the first component indicating that it is omitted
omit_ind <- c(TRUE, FALSE, FALSE)

#Calculate conditional (on x2=0.5) marginal (for x3) cdf approximation at point x
phpa(x = x,
pol_coefficients = pol_coefficients, pol_degrees = pol_degrees,
mean = mean, sd = sd,
given_ind = given_ind, omit_ind = omit_ind)
```

plot.hpaBinary *Plot hpaBinary random errors approximated density*

Description

Plot hpaBinary random errors approximated density

Usage

```
## S3 method for class 'hpaBinary'
plot(x, y = NULL, ...)
```

Arguments

x	Object of class "hpaBinary"
y	this parameter currently ignored
...	further arguments (currently ignored)

plot.hpaSelection *Plot hpaSelection random errors approximated density*

Description

Plot hpaSelection random errors approximated density

Usage

```
## S3 method for class 'hpaSelection'
plot(x, y = NULL, ..., is_outcome = TRUE)
```

Arguments

x	Object of class "hpaSelection"
y	this parameter currently ignored
...	further arguments (currently ignored)
is_outcome	logical; if TRUE then function plots the graph for outcome equation random errors. Otherwise plot for selection equation random errors will be plotted.

Value

This function returns the list containing random error's expected value `errors_exp` and variance `errors_var` estimates for selection (if `is_outcome = TRUE`) or outcome (if `is_outcome = FALSE`) equation.

plot_hpaBinary *Plot hpaBinary random errors approximated density*

Description

Plot hpaBinary random errors approximated density

Usage

```
plot_hpaBinary(x)
```

Arguments

x	Object of class "hpaBinary"

plot_hpaSelection *Plot hpaSelection random errors approximated density*

Description

Plot hpaSelection random errors approximated density

Usage

```
plot_hpaSelection(x, is_outcome = TRUE)
```

Arguments

x	Object of class "hpaSelection"
is_outcome	logical; if TRUE then function plots the graph for outcome equation random errors. Otherwise plot for selection equation random errors will be plotted.

Value

This function returns the list containing random error's expected value `errors_exp` and variance `errors_var` estimates for selection (if `is_outcome = TRUE`) or outcome (if `is_outcome = FALSE`) equation.

`pnorm_parallel` *Calculate normal cdf in parallel*

Description

Calculate in parallel for each value from vector `x` distribution function of normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

Usage

```
pnorm_parallel(x, mean = 0, sd = 1, is_parallel = FALSE)
```

Arguments

`x`	vector of quantiles: should be numeric vector, not just double value.
`mean`	double value.
`sd`	double positive value.
`is_parallel`	if TRUE then multiple cores will be used for some calculations. It usually provides speed advantage for large enough samples (about more than 1000 observations).

`polynomialIndex` *Returns matrix of polynomial indexes*

Description

Returns matrix of polynomial indexes for the polynomial with degrees (orders) vector `pol_degrees`.

Usage

```
polynomialIndex(pol_degrees = 0L)
```

Arguments

`pol_degrees`	non-negative integer vector of polynomial degrees.

Details

This function motivation is to have an opportunity to iterate through the columns of polynomial indexes matrix in order to access polynomial elements being aware of their powers.

Value

This function returns polynomial indexes matrix which rows are responsible for variables while columns are related to powers.

Examples

```
## Get polynomial indexes matrix for the polynomial which degrees are (1, 3, 5)

polynomialIndex(c(1, 3, 5))


## Consider polynomial of degrees (2, 1) such that coefficients
## for elements which powers sum is even are 2 and for those which powers
## are odd are 5. So the polynomial is 2+5y+5x+2xy+2x^2+5yx^2.

# Let's represent its powers (not coefficients) in a matrix form
pol_matrix <- polynomialIndex(c(2, 1))

# Suppose we want to calculate this polynomial coefficients sum:
powers_sum <- 0

# For pedagogical reasons iterate throught the pol_matrix columns
pol_matrix_length = dim(pol_matrix)[2]

for (i in 1:pol_matrix_length)
{
if ((pol_matrix[1, i] + pol_matrix[2, i]) %% 2 == 0)
{
  powers_sum <- powers_sum + 2
} else {
  powers_sum <- powers_sum + 5
}
}
#powers_sum value will be 21
```

`predict.hpaBinary` *Predict method for hpaBinary*

Description

Predict method for hpaBinary

Usage

```
## S3 method for class 'hpaBinary'
predict(object, ..., newdata = NULL, is_prob = TRUE)
```

Arguments

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

`newdata`	An optional data frame (for hpaBinary and hpaSelection) or numeric matrix (for hpaML) in which to look for variables with which to predict. If omitted, the original dataframe (matrix) used.
`is_prob`	logical; if TRUE (default) then function returns predicted probabilities. Otherwise latent variable (single index) estimates will be returned.

Value

This function returns predicted probabilities based on `hpaBinary` estimation results.

`predict.hpaML` *Predict method for hpaML*

Description

Predict method for hpaML

Usage

```
## S3 method for class 'hpaML'
predict(object, ..., newdata = matrix(c(0)))
```

Arguments

`object`	Object of class "hpaML"
`...`	further arguments (currently ignored)
`newdata`	An optional data frame (for hpaBinary and hpaSelection) or numeric matrix (for hpaML) in which to look for variables with which to predict. If omitted, the original dataframe (matrix) used.

Value

This function returns predictions based on `hpaML` estimation results.

`predict.hpaSelection` *Predict outcome and selection equation values from hpaSelection model*

Description

This function predicts outcome and selection equation values from hpaSelection model.

Usage

```
## S3 method for class 'hpaSelection'
predict(
  object,
  ...,
  newdata = NULL,
  method = "HPA",
  is_cond = TRUE,
  is_outcome = TRUE
)
```

Arguments

`object`	Object of class "hpaSelection"
`...`	further arguments (currently ignored)
`newdata`	An optional data frame (for hpaBinary and hpaSelection) or numeric matrix (for hpaML) in which to look for variables with which to predict. If omitted, the original dataframe (matrix) used.
`method`	string value indicating prediction method based on hermite polynomial approximation "HPA" or Newey method "Newey".
`is_cond`	logical; if `TRUE` (default) then conditional predictions will be estimated. Otherwise unconditional predictions will be returned.
`is_outcome`	logical; if `TRUE` (default) then predictions for selection equation will be estimated using "HPA" method. Otherwise selection equation predictions (probabilities) will be returned.

Details

Note that Newey method can't predict conditional outcomes for zero selection equation value. Conditional probabilities for selection equation could be estimated only when dependent variable from outcome equation is observable.

Value

This function returns the list which structure depends on `method`, `is_probit` and `is_outcome` values.

`predict_hpaBinary` *Predict method for hpaBinary*

Description

Predict method for hpaBinary

Usage

```
predict_hpaBinary(object, newdata = NULL, is_prob = TRUE)
```

Arguments

`object`	Object of class "hpaBinary"
`newdata`	An optional data frame (for hpaBinary and hpaSelection) or numeric matrix (for hpaML) in which to look for variables with which to predict. If omitted, the original dataframe (matrix) used.
`is_prob`	logical; if TRUE (default) then function returns predicted probabilities. Otherwise latent variable (single index) estimates will be returned.

Value

This function returns predicted probabilities based on `hpaBinary` estimation results.

`predict_hpaML` *Predict method for hpaML*

Description

Predict method for hpaML

Usage

```
predict_hpaML(object, newdata = matrix(1, 1))
```

Arguments

`object`	Object of class "hpaML"
`newdata`	An optional data frame (for hpaBinary and hpaSelection) or numeric matrix (for hpaML) in which to look for variables with which to predict. If omitted, the original dataframe (matrix) used.

Value

This function returns predictions based on `hpaML` estimation results.

`predict_hpaSelection` *Predict outcome and selection equation values from hpaSelection model*

Description

This function predicts outcome and selection equation values from hpaSelection model.

Usage

```
predict_hpaSelection(
  object,
  newdata = NULL,
  method = "HPA",
  is_cond = TRUE,
  is_outcome = TRUE
)
```

Arguments

`object`	Object of class "hpaSelection"
`newdata`	An optional data frame (for hpaBinary and hpaSelection) or numeric matrix (for hpaML) in which to look for variables with which to predict. If omitted, the original dataframe (matrix) used.
`method`	string value indicating prediction method based on hermite polynomial approximation "HPA" or Newey method "Newey".
`is_cond`	logical; if `TRUE` (default) then conditional predictions will be estimated. Otherwise unconditional predictions will be returned.
`is_outcome`	logical; if `TRUE` (default) then predictions for selection equation will be estimated using "HPA" method. Otherwise selection equation predictions (probabilities) will be returned.

Details

Note that Newey method can't predict conditional outcomes for zero selection equation value. Conditional probabilities for selection equation could be estimated only when dependent variable from outcome equation is observable.

Value

This function returns the list which structure depends on `method`, `is_probit` and `is_outcome` values.

print.summary.hpaBinary *Summary for hpaBinary output*

Description

Summary for hpaBinary output

Usage

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

Arguments

x	Object of class "hpaML"
...	further arguments (currently ignored)

print.summary.hpaML *Summary for hpaML output*

Description

Summary for hpaML output

Usage

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

Arguments

x	Object of class "hpaML"
...	further arguments (currently ignored)

print.summary.hpaSelection *Summary for hpaSelection output*

Description

Summary for hpaSelection output

Usage

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

Arguments

`x`	Object of class "hpaSelection"
`...`	further arguments (currently ignored)

printPolynomial *Print polynomial given it's degrees and coefficients*

Description

This function prints polynomial given it's degrees and coefficients.

Usage

```
printPolynomial(pol_degrees, pol_coefficients)
```

Arguments

`pol_degrees`	non-negative integer vector of polynomial degrees.
`pol_coefficients`	numeric vector of polynomial coefficients.

Details

Function automatically removes polynomial elements which coefficient are zero and variables which power is zero. Output may contain long coefficients representation as they are not rounded.

Value

This function returns the string which contains polynomial symbolic representation.

Examples

```
##Let's represent polynomial 0.3+0.5x2-x2^2+2x1+1.5x1x2+x1x2^2

pol_degrees <- c(1, 2)
pol_coefficients <- c(0.3, 0.5, -1, 2, 1.5, 1)

printPolynomial(pol_degrees, pol_coefficients)
```

`print_summary_hpaBinary` *Summary for hpaBinary output*

Description

Summary for hpaBinary output

Usage

```
print_summary_hpaBinary(x)
```

Arguments

`x`	Object of class "hpaML"

`print_summary_hpaML` *Summary for hpaML output*

Description

Summary for hpaML output

Usage

```
print_summary_hpaML(x)
```

Arguments

`x`	Object of class "hpaML"

`print_summary_hpaSelection` *Summary for hpaSelection output*

Description

Summary for hpaSelection output

Usage

```
print_summary_hpaSelection(x)
```

Arguments

`x`	Object of class "hpaSelection"

`summary.hpaBinary` *Summarizing hpaBinary Fits*

Description

Summarizing hpaBinary Fits

Usage

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

Arguments

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

Value

This function returns the same list as `hpaBinary` function changing it's class to "summary.hpaBinary".

`summary.hpaML` *Summarizing hpaML Fits*

Description

Summarizing hpaML Fits

Usage

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

Arguments

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

Value

This function returns the same list as `hpaML` function changing it's class to "summary.hpaML".

`summary.hpaSelection` *Summarizing hpaSelection Fits*

Description

This function summarizing hpaSelection Fits

Usage

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

Arguments

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

Value

This function returns the same list as `hpaSelection` function changing it's class to "summary.hpaSelection".

`summary_hpaBinary` *Summarizing hpaBinary Fits*

Description

Summarizing hpaBinary Fits

Usage

```
summary_hpaBinary(object)
```

Arguments

`object`	Object of class "hpaBinary"

Value

This function returns the same list as `hpaBinary` function changing it's class to "summary.hpaBinary".

`summary_hpaML` *Summarizing hpaML Fits*

Description

Summarizing hpaML Fits

Usage

```
summary_hpaML(object)
```

Arguments

`object`	Object of class "hpaML"

Value

This function returns the same list as `hpaML` function changing it's class to "summary.hpaML".

`summary_hpaSelection` *Summarizing hpaSelection Fits*

Description

This function summarizing hpaSelection Fits

Usage

```
summary_hpaSelection(object)
```

Arguments

`object`	Object of class "hpaSelection"

Value

This function returns the same list as `hpaSelection` function changing it's class to "summary.hpaSelection".

`truncatedNormalMoment` *Calculate k-th order moment of truncated normal distribution*

Description

This function iteratively calculates k-th order moment of truncated normal distribution.

Usage

```
truncatedNormalMoment(
  k = 1L,
  x_lower = numeric(0),
  x_upper = numeric(0),
  mean = 0,
  sd = 1,
  pdf_lower = numeric(0),
  cdf_lower = numeric(0),
  pdf_upper = numeric(0),
  cdf_upper = numeric(0),
  cdf_difference = numeric(0),
  return_all_moments = FALSE,
  is_validation = TRUE,
  is_parallel = FALSE
)
```

Arguments

`k`	non-negative integer moment order.
`x_lower`	numeric vector of lower trancation points.
`x_upper`	numeric vector of upper trancation points.
`mean`	numeric expected value.
`sd`	positive numeric standard deviation.
`pdf_lower`	non-negative numeric matrix of precalculated normal density functions with mean `mean` and standard deviation `sd` at points given by `x_lower`.
`cdf_lower`	non-negative numeric matrix of precalculated normal cumulative distribution functions with mean `mean` and standard deviation `sd` at points given by `x_lower`.
`pdf_upper`	non-negative numeric matrix of precalculated normal density functions with mean `mean` and standard deviation `sd` at points given by `x_upper`.
`cdf_upper`	non-negative numeric matrix of precalculated normal cumulative distribution functions with mean `mean` and standard deviation `sd` at points given by `x_upper`.
`cdf_difference`	non-negative numeric matrix of predcalculated `cdf_upper-cdf_lower` values.
`return_all_moments`	logical; if `TRUE`, function returns the matrix of moments of normaly distributed random variable with mean = `mean` and standard deviation = `sd` under lower and upper truncation points `x_lower` and `x_upper` correspondingly. Note that element in i-th row and j-th column of this matrix corresponds to the i-th observation (j-1)-th order moment.
`is_validation`	bool value indicating whether function input arguments should be validated. Set it to FALSE for slight perfomance boost (default value is TRUE).
`is_parallel`	if TRUE then multiple cores will be used for some calculations. It usually provides speed advantage for large enough samples (about more than 1000 observations).

Details

This function estimates `k`-th order moment of normal distribution which mean equals to `mean` and standard deviation equals to `sd` truncated at points given by `x_lower` and `x_upper`. Note that the function is vectorized so you can provide `x_lower` and `x_upper` as vectors of equal size. If vectors values for `x_lower` and `x_upper` are not provided then their default values will be set to `-(.Machine$double.xmin * 0.99)` and `(.Machine$double.xmax * 0.99)` correspondingly.

Note that parameter `k` value automatically converts to integer. So passing non-integer `k` value will not cause any errors but the calculations will be performed for rounded `k` value only.

If you have precaulculated density or cumulative distribution functions at standartized truncation points (substract `mean` and then divide by `sd`) then provide them throught `pdf_lower`, `pdf_upper`, `cdf_lower` and `cdf_upper` arguments in order to decrease number of calculations.

Value

This function returns vector of k-th order moments for normaly distributed random variable with mean = `mean` and standard deviation = `sd` under x_lower and x_upper truncation points `x_lower` and `x_upper` correspondingly. If `return_all_moments` is `TRUE` then see this argument description above for output details.

Examples

```
## Calculate 5-th order moment of three truncated normal random variables (x1,x2,x3)
## which mean is 5 and standard deviation is 3.
## These random variables truncation points are given as follows:-1<x1<1, 0<x2<2, 1<x3<3.
k <- 3
x_lower <- c(-1, 0, 1)
x_upper <- c(1, 2 ,3)
mean <- 3
sd <- 5

# get the moments
truncatedNormalMoment(k, x_lower, x_upper, mean, sd)

# get matrix of (0-5)-th moments (columns) for each variable (rows)
truncatedNormalMoment(k, x_lower, x_upper, mean, sd, return_all_moments = TRUE)
```

Index