Package ‘utilityFunctionTools’

July 22, 2025

Type Package

Title P-Spline Regression for Utility Functions and Derived Measures

Description Predicts a smooth and continuous (individual) utility function from utility points, and computes measures of intensity for risk and higher order risk measures (or any other measure computed with user-written function) based on this utility function and its derivatives according to the method introduced in Schneider (2017) <http://hdl.handle.net/21.11130/00-1735-0000-002E-E306-0>.

Version 0.1.1

Maintainer Sebastian O. Schneider <sschneider@coll.mpg.de>

URL https://www.sebastianoschneider.com

License GPL-3

Imports spatstat.geom

Encoding UTF-8

RoxygenNote 7.1.1

NeedsCompilation no

Author Sebastian O. Schneider [aut, cre, prg],
Giulia Baldini [edt, prg],
Paul Eilers [ctb] (Functions tpower and bbase, see Package JOPS at
http://statweb.lsu.edu/faculty/marx/JOPS_0.1.0.tar.gz)

Repository CRAN

Date/Publication 2021-03-29 09:20:05 UTC

Contents

`bbase` *Constructs a B-spline basis of degree 'deg' (Code by Paul Eilers, Package JOPS, http://statweb.lsu.edu/faculty/marx/JOPS_0.1.0.tar.gz).*

Description

Constructs a B-spline basis of degree 'deg' (Code by Paul Eilers, Package JOPS, http://statweb.lsu.edu/faculty/marx/JOPS_0.

Usage

```
bbase(x, xl = min(x), xr = max(x), ndx = 20, deg = 6)
```

Arguments

`x`	values for the x axis.
`xl`	minimum value, default is the minimum value of the x-values.
`xr`	maximum value, default is maximum value of the x-values.
`ndx`	number of intervals to partition the distance between xl and xr.
`deg`	degree of the B-spline basis.

Value

a B-spline basis of degree deg and ndx + 1 internal knots.

Examples

```
x_finegrid <- seq(0.001, 1.0, (1.0 - 0.001) / 1000)
bbase(x_finegrid)
```

`compute_function` *Computes a continuous and smooth utility function from the given utility points*

Description

Computes a continuous and smooth utility function from the given utility points

Usage

```
compute_function(
  x,
  y,
  ids = NULL,
  mode = 1,
  penalty_order = 4,
  lambda_max = 10000,
  current_lambda = 1,
  ndx = 20,
  deg = 6,
  verbose = 0
)
```

Arguments

`x` a matrix or dataframe containing the certainty equivalents (x-values of utility points) for a given participant in each use case.

`y` can be a vector or a matrix representing the corresponding utility values (y-values of utility points).

`ids` a list containing the IDs of the participants. If not given, a list with IDs from 1 to n_observations will be created.

`mode` an integer between 0, 1, 2 representing the three possible modes: multiple imputation, optimal classification or 'weak' classification. Default is optimal classification (1).

`penalty_order` highest dimension (i.e., derivative) to penalize. Must be lower than deg.

`lambda_max` maximum lambda used for computing the optimal lambda. It is used only in multiple imputation (mode = 0) and optimal (mode = 1). The default value is 10000.

`current_lambda` lambda considered in the current iteration. Only used in multiple imputation (mode = 0) to create the combinations and as actual lambda value in 'weak' classification mode (mode = 2). The default value is 1.

`ndx` number of intervals to partition the distance between the lowest and highest x-values of the utility points.

`deg` degree of the B-spline basis. Determines the degree of the function to be estimated. If deg = 2, the estimated utility function will consist of quadratic functions.

`verbose` shows some information while the program is running.

Value

A smooth and continuous utility function.

Examples

```
x <- matrix(c(24.60938,34.76074,78.75,81.86035,128.5156,
              7.109375,80.4248,113.75,115.083,135.0781,
              3.828125,7.211914,8.75,124.1064,131.7969,
             1.640625,2.084961,8.75,36.94824,98.98438), nrow = 4, ncol = 5, byrow = TRUE)
y <- c(0.25, 0.375, 0.5, 0.625, 0.75)
compute_function(x, y, verbose = 1)
```

compute_higher_order_risk_preferences

Computes a continuous and smooth function according to the given utility points

Description

Computes a continuous and smooth function according to the given utility points

Usage

```
compute_higher_order_risk_preferences(
  x,
  y,
  ids = NULL,
  mode = 1,
  penalty_orders = c(4),
  ndx = 20,
  deg = 6,
  measures = c("risk-arrow-pratt", "crainich-eeckhoudt", "denuit-eeckhoudt"),
  ...,
  root_filename = NULL,
  verbose = 0
)
```

Arguments

`x`
a matrix or dataframe containing the certainty equivalents (x-values of utility points) for a given participant in each use case.

`y`
can be a vector or a matrix representing the corresponding utility values (y-values of utility points).

`ids`
a list containing the IDs of the participants. If not given, a list with IDs from 1 to n_observations will be created.

`mode`
an integer between 0, 1, 2 representing the three possible modes: multiple imputation, optimal classification or 'weak' classification. Default is optimal classification (1).

`penalty_orders` vector or constant that contains the derivates that will be smoothened. The values in this vector should not be larger than 4.

`ndx` number of intervals to partition the distance between the lowest and highest x-values of the utility points.

`deg` degree of the B-spline basis. Determines the degree of the function to be estimated. If deg = 2, the estimated utility function will consist of quadratic functions.

`measures` the utility based (intensity) measures to be computed.

`...` additional parameters for user-defined measures.

`root_filename` filename containing the location of where the output files are going to be saved.

`verbose` shows some information while the program is running.

Value

A smooth and continuous function.

Examples

```
x <- matrix(c(24.60938,34.76074,78.75,81.86035,128.5156,
              7.109375,80.4248,113.75,115.083,135.0781,
              3.828125,7.211914,8.75,124.1064,131.7969,
             1.640625,2.084961,8.75,36.94824,98.98438), nrow = 4, ncol = 5, byrow = TRUE)
y <- c(0.25, 0.375, 0.5, 0.625, 0.75)
compute_higher_order_risk_preferences(x, y, mode = 1)

# could be used with root_filename argument:
# Linux
# outfile <- paste(dirname(getwd()), "/out", sep="")
# Win
# outfile <- paste(dirname(getwd()), "\out", sep="")
compute_higher_order_risk_preferences(x, y, mode = 2, verbose = 1)
```

`compute_measures` *Given a set of smooth and continuous functions, computes predefined and user-defined measures.*

Description

Given a set of smooth and continuous functions, computes predefined and user-defined measures.

Usage

```
compute_measures(
  x_grids,
  coeffs,
  ids = NULL,
  ndx = 20,
  deg = 6,
  measures = c("risk-arrow-pratt", "crainich-eeckhoudt", "denuit-eeckhoudt"),
  ...
)
```

Arguments

`x_grids`	a dataframe of vectors of x values for a smooth and continuous function.
`coeffs`	a dataframe of coefficients for a smooth and continous function for each participant.
`ids`	a list containing the IDs of the participants. If not given, a list with IDs from 1 to n_observations will be created.
`ndx`	number of intervals to partition the distance between the lowest and highest x-values of the utility points.
`deg`	degree of the B-spline basis. Determines the degree of the function to be estimated. If deg = 2, the estimated utility function will consist of quadratic functions.
`measures`	a vector of measures to be computed.
`...`	additional parameters for user-defined measures.

Value

A set of measurements.

Examples

```
x <- rbind(seq(0.000002, 1.0, (1.0 - 0.000002) / 1000),
           seq(0.001, 1.0, (1.0 - 0.001) / 1000),
           seq(0.0004, 1.0, (1.0 - 0.0004) / 1000))
y <- rbind(seq(0.000002, 1.0, (1.0 - 0.000002) / 15),
           seq(0.001, 1.0, (1.0 - 0.001) / 15),
           seq(0.0004, 1.0, (1.0 - 0.0004) / 15))
compute_measures(x, y, ndx = 10, deg = 6)
# x_finegrid, coeff, ndx, deg are always there to be used
# The function should have additional unknown arguments (...) if the given parameters are not used
risk_arrow_pratt <- function(x_finegrid, coeff, ndx, deg){
  dy_rd <- derivative(x_finegrid, coeff, 1, ndx, deg)
  ddy_rd <- derivative(x_finegrid, coeff, 2, ndx, deg)
  return (-mean(ddy_rd, na.rm = TRUE) / mean(dy_rd, na.rm = TRUE))
}
measures = c("crainich-eeckhoudt", "denuit-eeckhoudt", risk_arrow_pratt)
compute_measures(x, y, ndx = 10, deg = 6, measures=measures)
```

derivative *Computes the derivative of a function*

Description

Computes the derivative of a function

Usage

```
derivative(x, coeffs, degree = 1, ndx = 20, deg = 6)
```

Arguments

`x` the x values for which the derivative should be computed.

`coeffs` the coefficient.

`degree` the degree of the derivative.

`ndx` number of intervals to partition the distance between the lowest and highest x-values of the utility points.

`deg` degree of the B-spline basis. Determines the degree of the function to be estimated. If deg = 2, the estimated utility function will consist of quadratic functions.

Value

the derivative of the specified degree.

Examples

```
coeffs <- seq(0.000002, 1.0, (1.0 - 0.000002) / 25)
x <- seq(0.01, 1.0, (1.0 - 0.01) / 5)
derivative(x, coeffs)
```

estimate_model *Estimates the model*

Description

Estimates the model

Usage

```
estimate_model(
  xi,
  yi,
  lambda = 1,
  n_penalty_dimensions = 1,
  penalty_order = 4,
  ndx = 20,
  deg = 6,
  cross_validation_mode = 0,
  return_estimate = 0,
  left_out_xi = c(),
  left_out_yi = c()
)
```

Arguments

`xi` a vector containing the certainty equivalents (x-values of utility points) for a given participant in each use case.

`yi` can be a vector or a matrix representing the corresponding utility values (y-values of utility points).

`lambda` lambda is the penalization weight used to compute the initial estimate. The default value is 1.

`n_penalty_dimensions` number of dimensions (i.e., derivatives) to penalize. Possible values are 1 or 2. The default value is 1.

`penalty_order` highest dimension (i.e., derivative) to penalize. Must be lower than deg.

`ndx` number of intervals to partition the distance between the lowest and highest x-values of the utility points.

`deg` degree of the B-spline basis. Determines the degree of the function to be estimated. If deg = 2, the estimated utility function will consist of quadratic functions.

`cross_validation_mode` determines which cross validation mode should be used. If 0, then the cross validation method is leave-one-third-out. If 1, then the cross validation method is a theoretical leave-one-out, i.e., based on a formula. The default value is 1.

`return_estimate` parameter that indicates whether or not to return the (initially) estimated coefficients. Default is false.

`left_out_xi` needed for cross validation: the x-values of the points that are left out for fitting the model, so that they can be predicted

`left_out_yi` needed for cross validation: the y-values of the points that are left out for fitting the model, so that they can be predicted

Value

Returns the sum of residuals of the prediction of the left-out points using cross validation. If specified, additionally returns the estimated coefficients of the utility function (in the B-spline basis).

Examples

```
x <- c(0.0000000, 0.2819824, 0.3007812, 0.4375000, 0.5231934, 0.7784882, 0.8945312, 1.0000000)
y <- c(0.0000, 0.1250, 0.2500, 0.5000, 0.6250, 0.6875, 0.7500, 1.0000)
estimate_model(x, y, .5)
```

`evaluate_cross_validation` *Evaluates the cross validation function.*

Description

Evaluates the cross validation function.

Usage

```
evaluate_cross_validation(
  xi,
  yi,
  lambda = 1,
  n_penalty_dimensions = 1,
  penalty_order = 4,
  ndx = 20,
  deg = 6,
  cross_validation_mode = 0
)
```

Arguments

`xi` a vector containing the certainty equivalents (x-values of utility points) for a given participant in each use case.

`yi` can be a vector or a matrix representing the corresponding utility values (y-values of utility points).

`lambda` lambda is the penalization weight used to compute the initial estimate. The default value is 1.

`n_penalty_dimensions` number of dimensions (i.e., derivatives) to penalize. Possible values are 1 or 2. The default value is 1.

`penalty_order` highest dimension (i.e., derivative) to penalize. Must be lower than deg.

`ndx` number of intervals to partition the distance between the lowest and highest x-values of the utility points.

`deg` degree of the B-spline basis. Determines the degree of the function to be estimated. If deg = 2, the estimated utility function will consist of quadratic functions.

`cross_validation_mode` determines which cross validation mode should be used. If 0, then the cross validation method is leave-one-third-out. If 1, then the cross validation method is a theoretical leave-one-out, i.e., based on a formula. The default value is 1.

Value

Returns, for the given utility points and (possibly default) settings, the predictive quality of the estimated utility function according to cross validation as a function of a specified penalty weight lambda.

Examples

```
x <- c(0.0000000, 0.2819824, 0.3007812, 0.4375000, 0.5231934, 0.7784882, 0.8945312, 1.0000000)
y <- c(0.0000, 0.1250, 0.2500, 0.5000, 0.6250, 0.6875, 0.7500, 1.0000)
evaluate_cross_validation(x, y, .5)
```

find_optimal_lambda *Finds an optimal penalty weight lambda given the parameters*

Description

Finds an optimal penalty weight lambda given the parameters

Usage

```
find_optimal_lambda(
  xi,
  yi,
  lambda_max = 10000,
  n_penalty_dimensions = 1,
  penalty_order = 4,
  ndx = 20,
  deg = 6,
  cross_validation_mode = 0,
  grid_dim = 5
)
```

Arguments

`xi` a vector containing the certainty equivalents (x-values of utility points) for a given participant in each use case.

`yi` can be a vector or a matrix representing the corresponding utility values (y-values of utility points).

`lambda_max` maximum lambda used for computing the optimal lambda. The default value is 10000.

`n_penalty_dimensions` number of dimensions (i.e., derivatives) to penalize. Possible values are 1 or 2. The default value is 1.

`penalty_order` highest dimension (i.e., derivative) to penalize. Must be lower than deg.

`ndx` number of intervals to partition the distance between the lowest and highest x-values of the utility points.

`deg` degree of the B-spline basis. Determines the degree of the function to be estimated. If deg = 2, the estimated utility function will consist of quadratic functions.

`cross_validation_mode` determines which cross validation mode should be used. If 0, then the cross validation method is leave-one-third-out. If 1, then the cross validation method is a theoretical leave-one-out, i.e., based on a formula. The default value is 1.

`grid_dim` dimension of the search grid for the initial grid search before the actual optimization. Default value is 5.

Value

the optimal lambda for the given set of utility points and (possibly default) settings according to the specified cross validation method.

Examples

```
x <- c(0.0000000, 0.2819824, 0.3007812, 0.4375000, 0.5231934, 0.7784882, 0.8945312, 1.0000000)
y <- c(0.0000, 0.1250, 0.2500, 0.5000, 0.6250, 0.6875, 0.7500, 1.0000)
find_optimal_lambda(x, y)
```

`tpower` *Truncated p-th power function. Helper function for creating the B-Spline basis (Code by Paul Eilers, Package JOPS, http://statweb.lsu.edu/faculty/marx/JOPS_0.1.0.tar.gz)*

Description

Truncated p-th power function. Helper function for creating the B-Spline basis (Code by Paul Eilers, Package JOPS, http://statweb.lsu.edu/faculty/marx/JOPS_0.1.0.tar.gz)

Usage

```
tpower(x, t, p)
```

Arguments

`x` Function value.

`t` Point of truncation.

`p` degree of the truncated polynomial function.

Value

Returns a piece-wise defined basis functions for x > t.

Examples

```
tpower(1, 2, 3)
```

Index