Package: RSD (via r-universe)

June 4, 2026

Type Package

Title Compares Random Distributions using Stochastic Dominance

Version 0.3.0

Maintainer Shayan Tohidi <shayant@iastate.edu>

Description The Stochastic Dominance (SD) is the classical way of comparing two random prospects, using their distribution functions. Almost Stochastic Dominance (ASD) has also been developed to cover the SD failures due to the extreme utility functions. This package focuses on classical and heuristic methods for testing the first and second SD and ASD methods given the probability mass function (PMF) of the random prospects. The goal is to apply these methods easily, efficiently, and effectively on real-world datasets. For more details see Hanoch and Levy (1969) <doi:10.2307/2296431>, Leshno and Levy (2002) <doi:10.1287/mnsc.48.8.1074.169>, and Tzeng et al. (2012) <doi:10.1287/mnsc.1120.1616>.

License GPL (>= 3)

Encoding UTF-8

LazyData true

URL https://github.com/ShayanTohidi/RSD

BugReports https://github.com/ShayanTohidi/RSD/issues

RoxygenNote 7.3.2

Imports dplyr, tidyr, ggplot2, magrittr, methods

Depends R (>= 3.5)

Config/pak/sysreqs libicu-dev

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

Date/Publication 2025-08-08 18:55:42 UTC

RemoteUrl https://github.com/shayantohidi/rsd

RemoteRef HEAD

RemoteSha 15980941b8a80d586db15e0e5fcd4c634e3bf9e9

Contents

`afsd.test` *Compares prospects based on AFSD*

Description

It compares two prospects using AFSD criteria, that is the prospect having the minimum violation area from a classic FSD.

Usage

```
afsd.test(sd.obj)
```

Arguments

`sd.obj`	StochasticDominance object.

Details

The returned list has six elements: ‘winner‘ indicates the dominant prospect index. It will be zero if neither dominates the other. ‘epsilon‘ is the ratio of violated area to the total area between the CDFs. ‘area‘ is a vector, where the values show the area between the CDFs correspond to each segment. ‘total.area‘ is the total area between the CDFs. ‘positive.area‘ is the amount of area where the ‘area‘ vector is positive, meaning the ‘cdf1‘ is larger than ‘cdf2‘. ‘negative.area‘ is like ‘positive.area‘ for negative values.

If neither distribution dominates the other by AFSD, the ‘winner‘ output will be zero, and it happens only when the expected value of distributions are equal.

The ‘epsilon‘ and ‘winner‘ output parameters are the ones that should be taken most. The others are the calculation details and are provided for further investigation. A lower the ‘epsilon‘, lower the violation ratio of the dominant distribution, lower the eliminated extreme utilities, higher the number of decision-makers who agree on the dominant distribution.

Value

A list, including all the calculation details.

See Also

[area.btwn.cdfs.calc()] for area calculations.

Examples

```
sd = createStochasticDominance(outcome1 = c(1,4,7),
                               outcome2 = c(2,3,5),
                               prob1 = c(1/3,1/3,1/3),
                               prob2 = c(1/6,1/6,2/3))
afsd.test(sd)
```

area.below.ssd.calc *Calculates area below SSD function*

Description

For every segment, the area below SSD function will be computed.

Usage

```
area.below.ssd.calc(outcome, ssd)
```

Arguments

outcome	Numeric vector, including outcome values.
ssd	Numeric vector, including SSD values.

Value

Numeric vector, including area below every segment of SSD function.

See Also

[calc.area.below.line()] for area below line.

area.btwn.cdfs.calc *Calculates area between CDFs*

Description

It calculates the area between the CDFs of two prospects divided by the CDFs and outcomes segments.

Usage

```
area.btwn.cdfs.calc(sd.obj)
```

Arguments

sd.obj	StochasticDominance object.

Value

A numeric vector, including the area between the segments of CDFs.

area.btwn.ssd.calc *Calculates area between SSD functions*

Description

For every segments, the area between SSD functions will be computed.

Usage

```
area.btwn.ssd.calc(sd.obj)
```

Arguments

sd.obj	StochasticDominance object.

Value

Numeric vector, including area differences in every segments.

See Also

[modif.outcome.ssd.calc(), area.below.ssd.calc()] for more details.

asd.screen *Create the inefficient set by an ASD rule*

Description

It uses the test result, corresponding epsilon, and the epsilon threshold to create the inefficient set.

Usage

```
asd.screen(data, test, epsilon.threshold)
```

Arguments

data	A data frame, including variable pairs, distributions, and results of the tests.
test	A string that indicates the name of the test.
epsilon.threshold	A number that indicates the threshold for considering the result to be valid or not.

Details

The inefficient set includes all variable names that are dominated by at least one other variable. The domination defines as the test result where the corresponding epsilon is smaller than or equal to epsilon threshold (‘epsilon‘). If the epsilon is smaller than the threshold, it is assumed that by some relaxation the domination exists because a very small ratio of the decision-makers do not agree with this result.

Value

A character vector as the inefficient set of variables based on the ‘test‘ result and ‘epsilon.threshold‘.

---

assd.ll.test *Compares random prospects by ASSD-LL*

Description

It uses the positive and negative areas that are computed by ASSD-LL and the expected values of the prospects to compare them based on the ASSD-LL rule. If the violation area ratio is less than 0.5 for a prospect, and its expected value is larger, it dominates the other by ASSD-LL.

Usage

```
assd.ll.test(sd.obj)
```

Arguments

sd.obj	StochasticDominance object.

Details

epsilon shows the ratio of the violation. Smaller epsilon means more decision-makers agree with the result.

The returned list has six elements: ‘winner‘ indicates the dominant prospect index. It will be zero if neither dominates the other. ‘epsilon‘ is the ratio of violated area to the total area between the CDFs. ‘area‘ is a vector, where the values show the area between the CDFs correspond to each segment. ‘total.area‘ is the total area between the CDFs. ‘positive.area‘ is the amount of area where the ‘area‘ vector is positive, meaning the ‘cdf1‘ is larger than ‘cdf2‘ and ‘ssd1‘ is larger than ‘ssd2‘. ‘negative.area‘ is like ‘positive.area‘ for negative values.

If neither distribution dominates the other by ASSD-LL, the ‘winner‘ output will be zero, and it happens only when the distribution with a higher expected value has the ‘epsilon‘ which is larger than 0.5.

Value

A list, including all the calculation details.

See Also

[expected.values(), pos.neg.area.assd.ll(), afsd.test()] for more details.

`assd.test` *Compares prospects based on ASSD methods*

Description

It compares two prospects using ASSD criteria, that is the prospect having the minimum violation area from a classic SSD.

Usage

```
assd.test(sd.obj, type)
```

Arguments

`sd.obj`	StochasticDominance object.
`type`	A character vector, including the name of ASSD methods.

Details

The ‘type‘ argument must be one of the ’ll’ or ’ths’, otherwise it will raise an error.

The ‘epsilon‘ and ‘winner‘ output parameters are the ones that should be taken most. The others are the calculation details and are provided for further investigation. A lower the ‘epsilon‘, lower the violation ratio of the dominant distribution, lower the eliminated extreme utilities, higher the number of decision-makers who agree on the dominant distribution.

Value

A list, including calculation details.

See Also

[assd.ll.test(), assd.ths.test] for more details.

Examples

```
sd = createStochasticDominance(outcome1 = c(1,4,7),
                               outcome2 = c(2,3,5),
                               prob1 = c(1/3,1/3,1/3),
                               prob2 = c(1/6,1/6,2/3))
assd.test(sd, 'll')
assd.test(sd, 'ths')
```

`assd.ths.test` *Compares random prospects by ASSD-THS*

Description

It uses the areas between the SSD functions, and the expected values of the prospects to compare them based on the ASSD-THS rule. If the violation area ratio is less than 0.5 for a prospect, and its expected value is larger, it dominates the other by ASSD-THS.

Usage

```
assd.ths.test(sd.obj)
```

Arguments

`sd.obj`	StochasticDominance object.

Details

epsilon shows the ratio of the violation. Smaller epsilon means more decision-makers agree with the result.

The returned list has six elements: ‘winner‘ indicates the dominant prospect index. It will be zero if neither dominates the other. ‘epsilon‘ is the ratio of violated area to the total area between the SSDs. ‘area‘ is a vector, where the values show the area between the SSDs correspond to each segment. ‘total.area‘ is the total area between the SSDs. ‘positive.area‘ is the amount of area where the ‘area‘ vector is positive, meaning the ‘ssd1‘ is larger than ‘ssds2‘. ‘negative.area‘ is like ‘positive.area‘ for negative values.

If neither distribution dominates the other by ASSD-THS, the ‘winner‘ output will be zero, and it happens only when the distribution with a higher expected value has the ‘epsilon‘ which is larger than 0.5.

Value

A list, including all the calculation details.

See Also

[expected.values(), area.btwn.ssd.calc()] for more details.

calc.area.below.line *Calculates the area between x-axis and a straight line*

Description

It takes four parameters as the coordinates of two points at the beginning and end of a straight line, and calculates the area exactly below the line (between the line and the x-axis).

Usage

```
calc.area.below.line(x1, x2, y1, y2)
```

Arguments

x1, x2	Float values, indicating the first coordinates of start and end points, respectively.
y1, y2	Float values, indicating the second coordinates of start and end points, respectively.

Value

A float.

calc.intersection *Calculates the intersection point of two lines.*

Description

Given the start and end coordinates of two straight lines, we calculate the intersection point of them.

Usage

```
calc.intersection(x1, x2, y11, y12, y21, y22)
```

Arguments

x1, x2	Float values.
y11, y12	Float values, corresponding to the first line.
y21, y22	Float values, corresponding to the second line.

Details

Having intersection point has been checked before.

Value

A list, including the coordinates of the intersection point.

See Also

[has.intersect()]

`compare.all` *Comparing all pairs by all rules, and finding the sets*

Description

It creates a data frame of unique unordered pairs of variables, collects corresponding distributions, tests all SD and ASD rules on every pairs, and finds the efficient and inefficient sets for all those rules.

Usage

```
compare.all(
  variable,
  probability,
  outcome,
  afsd.epsilon.threshold = 0.1,
  assd.ll.epsilon.threshold = 0.1,
  assd.ths.epsilon.threshold = 0.1,
  include.details = TRUE
)
```

Arguments

`variable`	A character vector containing the name of all variables.
`probability`	A numeric vector containing the probability for each variable to achieve a particular outcome.
`outcome`	A numeric vector containing the outcome values.
`afsd.epsilon.threshold`	A number that shows the upper limit for epsilon of afsd rule. If the epsilon is smaller than or equal to this value, the result is accepted. The default value is '0.1'.
`assd.ll.epsilon.threshold`	A number that shows the upper limit for epsilon of assd.ll rule. If the epsilon is smaller than or equal to this value, the result is accepted. The default value is '0.1'.
`assd.ths.epsilon.threshold`	A number that shows the upper limit for epsilon of assd.ths rule. If the epsilon is smaller than or equal to this value, the result is accepted. The default value is '0.1'.
`include.details`	A logical that affects the output data to contain distributions. The default value is 'TRUE'.

Details

If the type of each input parameter is incorrect, it will raise an error.

The output is a list that contains 6 elements: ‘data‘ is a data frame including all unique unordered pairs of variables, corresponding distributions, result of each rule, and epsilon of ASD rule. The rest 5 elements are lists as well, corresponding to each rule that includes the efficient and inefficient sets.

The ‘data‘ is a nested data frame, and stores the distribution of each variable in a single cell. The distribution contains two columns: probability and outcome.

Value

A list of 6 elements, including a data frame and the efficient and inefficient sets of all rules.

See Also

[compare.paired.distributions(), screen.by.sd(), screen.by.asd()]

Examples

```
result = compare.all(data_ex$gen, rep(1/29,377), data_ex$yield)
```

---

compare.paired.distributions

Creating paired distributions and execute all SD and ASD rules on them

---

Description

It creates a data frame including all unordered pairs of the variables, where the first variable has higher or equal mean outcome. Then, all available SD and ASD rules are tested on them.

Usage

```
compare.paired.distributions(
  variable,
  probability,
  outcome,
  include.details = TRUE
)
```

Arguments

variable	A character vector containing the name of all variables.
probability	A numeric vector containing the probability for each variable to achieve a particular outcome.
outcome	A numeric vector containing the outcome values.
include.details	A logical that affects the output data to contain distributions. The default value is 'TRUE'.

Details

If the type of each input parameter is incorrect, it will raise an error.

The length of the input parameters 'variable', 'probability', and 'outcome' must be equal, otherwise it will raise an error.

Value

A data frame, including all distribution pairs and the results of the tests.

See Also

[create.paired.distributions(), sd.asd.test.all()]

Examples

```
result = compare.paired.distributions(data_ex$gen, rep(1/29,377), data_ex$yield)
```

---

comparison *Comparing two numeric vectors*

Description

This function compares two numeric vectors. The vector whose all elements smaller or equal to all elements of the other one where at least one element is smaller, will be the winner.

Usage

```
comparison(x, y)
```

Arguments

x, y	Numeric vectors.

Details

The function returns the index of the winner, meaning 1 or 2, corresponding to parameters 'x' and 'y', respectively. If no vector meets the condition, 0 will be returned.

Value

An integer, among 0, 1, 2.

`create.dataframe` *create a data frame from input parameters*

Description

create a data frame from input parameters

Usage

```
create.dataframe(variable, probability, outcome)
```

Arguments

`variable`	A character vector.
`probability`	A numeric vector.
`outcome`	A numeric vector.

Value

A data frame with three columns.

`create.paired.distributions`
Creating paired distributions

Description

with the input parameters and four other functions, it will create a data frame that includes all pairs of variables and their corresponding probability and outcome values.

Usage

```
create.paired.distributions(variable, probability, outcome)
```

Arguments

`variable`	A character vector.
`probability`	A numeric vector.
`outcome`	A numeric vector.

Details

Here, we first sort variables based on their outcome in ascending order, and then create unordered unique pairs. So, the first element (variable) of any pair has larger mean outcome than the second element.

The output has four columns. The first two represent the two elements of each pair. The last two are the corresponding probability and outcome for the elements, respectively. These are in nested format, meaning each cell of the last two columns includes a data frame with two columns: probability and outcome.

Value

A data frame with four columns.

See Also

[create.dataframe(), sort.variables(), pair.variables(), pair.distributions()]

`createStochasticDominance`
Constructor of StochasticDominance Class

Description

It is much easier to use this constructor to create an instance of the StochasticDominance class. It handles calculation of the cdf and ssd values in an efficient way.

Usage

```
createStochasticDominance(outcome1, outcome2, prob1, prob2)
```

Arguments

`outcome1`, `outcome2`	Numeric vectors. The outcomes corresponding to each prospect.
`prob1`, `prob2`	Numeric vectors. The probabilities corresponding to each prospect.

Value

StochasticDominance object.

See Also

[StochasticDominance()]

Examples

```
createStochasticDominance(outcome1 = c(1,4,7),
                          outcome2 = c(2,3,5),
                          prob1 = c(1/3,1/3,1/3),
                          prob2 = c(1/6,1/6,2/3))
```

---

data_ex *The example dataset*

Description

It is a real dataset in agriculture, which includes planted cultivars in multi-environment and the resulting yield.

Usage

```
data_ex
```

Format

A data frame with 377 rows and 3 columns:

gen Character vector of genotype (cultivar) names

env Character vector of environment names

yield Numeric vector of yield of planting each cultivar in each environment

Source

The phenotypic data from G2F initiative in 2018: <https://doi.org/10.25739/anqq-sg86>

---

expected.values *Calculating Expected value*

Description

It calculates the expected value of both prospects given their distributions.

Usage

```
expected.values(sd.obj)
```

Arguments

sd.obj	StochasticDominance object.

Value

A list, including two double elements as the expected value of each prospect.

fsd.plot *Drawing the CDFs*

Description

It visualizes the CDFs of both prospects.

Usage

```
fsd.plot(sd.obj, names = c("1", "2"))
```

Arguments

sd.obj	StochasticDominance object.
names	A character vector, including the names of prospects in order.

Details

The parameter ‘names‘ only accepts character vector, otherwise an error will be raised.

The function shows the step plot, and returns its object for further modifications.

Value

A list, including plot elements.

Examples

```
sd = createStochasticDominance(outcome1 = c(1,4,7),
                               outcome2 = c(2,3,5),
                               prob1 = c(1/3,1/3,1/3),
                               prob2 = c(1/6,1/6,2/3))
fsd.plot(sd, names = c('First', 'Second'))
```

fsd.test *Compares random prospects by FSD*

Description

It compares two random prospects by the first-order stochastic dominance (FSD).

Usage

```
fsd.test(sd.obj)
```

Arguments

sd.obj	StochasticDominance object.

Details

A prospect dominates when its CDF is below the other one. It means that all element of the CDF vector must be equal or smaller, and at least one element should be smaller for the dominant prospect.

If neither prospect dominates the other by FSD, it returns 0. It means that the CDFs intersect each other.

Value

An integer, indicating the index of the dominant prospect.

Examples

```
sd = createStochasticDominance(outcome1 = c(1,4,7),
                               outcome2 = c(2,3,5),
                               prob1 = c(1/3,1/3,1/3),
                               prob2 = c(1/6,1/6,2/3))
fsd.test(sd)
```

---

has.intersection *If two lines have intersection or no.*

Description

It determines if two lines have intersection point or not. The start and end points of both lines have equal first coordinate value (the one corresponds to the x-axis).

Usage

```
has.intersection(x1, x2, y11, y12, y21, y22)
```

Arguments

x1, x2	Float values.
y11, y12	Float values, corresponding to the first line.
y21, y22	Float values, corresponding to the second line.

Value

A Boolean.

modif.outcome.ssd.calc

Modify outcome and SSD vectors

Description

It modifies outcome vector to have all original plus intersection points in ascending order. The corresponding SSD vectors will also be returned.

Usage

```
modif.outcome.ssd.calc(sd.obj)
```

Arguments

sd.obj	StochasticDominance object.

Value

A list of three elements.

pair.distributions

Paring all the distributions

Description

It uses paired variables and collects all the related information (probability and outcome) corresponding to each variable of every pairs.

Usage

```
pair.distributions(org.data, paired.vars)
```

Arguments

org.data	A data frame that includes all variables, probabilities, and outcomes.
paired.vars	A data frame that includes all unique pairs of variables.

Value

A data frame that includes all unique pairs of variables with their corresponding distributions in the nested format.

pair.variables *Paring all the variables*

Description

It uses a vector of variables and creates unique unordered pairs.

Usage

```
pair.variables(variables)
```

Arguments

variables	A character vector, including the variable names.

Details

These pairs are unique unordered pairs, meaning the order of variables does not matter. So if we have pair {x,y} we do not create another pair {y,x}, because they are technically the same. If the number of variables is n, the number of pairs is n(n-1)/2.

Value

A data frame, that includes two columns; every row represents a pair of variables.

pos.neg.area.assd.ll *Calculates positive and negative area between CDFs for ASSD-LL*

Description

It calculates the positive and negative areas between CDFs. The positive is where both CDF and SSD of the first prospect is larger, and vice versa for the negative case.

Usage

```
pos.neg.area.assd.ll(sd.obj)
```

Arguments

sd.obj	StochasticDominance object.

Value

A list, including three elements corresponding to the positive and negative areas, respectively, and the area vector at the end.

See Also

[modif.outcome.ssd.calc()] for more details.

`screen` *A wrapper for computing inefficient and efficient sets*

Description

A wrapper for computing inefficient and efficient sets

Usage

```
screen(data, test, epsilon.threshold)
```

Arguments

`data`	A data frame, including variable pairs, distributions, and results of the tests.
`test`	A string that indicates the name of the test.
`epsilon.threshold`	A number that indicates the threshold for considering the result to be valid or not for ASD tests.

Value

A list of two elements, efficient and inefficient sets.

See Also

[sd.screen(), asd.screen()]

`screen.by.asd` *Screening by all ASD rules*

Description

It checks all available ASD rules (i.e. afsd, assd.ll, and assd.ths) based on their results, and creates the corresponding efficient and inefficient sets.

Usage

```
screen.by.asd(
  data,
  afsd.epsilon.threshold = 0.1,
  assd.ll.epsilon.threshold = 0.1,
  assd.ths.epsilon.threshold = 0.1
)
```

Arguments

`data`	A data frame, including the results of all ASD rules.
`afsd.epsilon.threshold`	A number that shows the upper limit for epsilon of afsd rule. If the epsilon is smaller than or equal to this value, the result is accepted. The default value is '0.1'.
`assd.ll.epsilon.threshold`	A number that shows the upper limit for epsilon of assd.ll rule. If the epsilon is smaller than or equal to this value, the result is accepted. The default value is '0.1'.
`assd.ths.epsilon.threshold`	A number that shows the upper limit for epsilon of assd.ths rule. If the epsilon is smaller than or equal to this value, the result is accepted. The default value is '0.1'.

Details

The input parameter 'data' must contain columns corresponding to ASD rules. These columns contain the index of the dominated variable, where 1 or 2 means the first or the second variable dominates, and 0 means the domination does not exist. Also, corresponding to each rule, we must have a column that includes the epsilon values for each rule. The best practice is to use the output of 'compare.paired.distributions'.

Value

A list, including three elements corresponding to each ASD rule.

See Also

[screen()]

Examples

```
data = compare.paired.distributions(data_ex$gen, rep(1/29,377), data_ex$yield)
asd.sets = screen.by.asd(data)
```

`screen.by.sd` *Screening by all SD rules*

Description

It checks all available SD rules (i.e. fsd and ssd) based on their results, and creates the corresponding efficient and inefficient sets.

Usage

```
screen.by.sd(data)
```

Arguments

data	A data frame, including the results of all SD rules.

Details

The input parameter ‘data‘ must contain columns corresponding to SD rules. These columns contain the index of the dominated variable, where 1 or 2 means the first or the second variable dominates, and 0 means the domination does not exist. The best practice is to use the output of ‘compare.paired.distributions‘.

Value

A list, including two elements corresponding to each SD rule.

See Also

[screen()]

Examples

```
data = compare.paired.distributions(data_ex$gen, rep(1/29,377), data_ex$yield)
sd.sets = screen.by.sd(data)
```

---

sd.asd.test *Performing all SD and ASD methods on a pair*

Description

Performing all SD and ASD methods on a pair

Usage

```
sd.asd.test(data1, data2)
```

Arguments

data1, data2	Data frames corresponding to each element of a pair, respectively.

Details

The input parameters are two data frames corresponding to each element of a pair, respectively. Each data frame has two columns: probability and outcome, which represents the distribution of the element (variable).

First, a stochastic dominance object is created using the columns of the two input parameters. Then, all available SD and ASD tests that are implemented inside the package are performed.

The output list contains 8 elements: ‘fsd‘, ‘ssd‘, ‘afsd‘, ‘assd.ll‘, and ‘assd.ths‘ indicate the index of the dominant element (variable). If the first variable dominates, they show 1, else they show 2.

If neither dominates they return 0. The other output parameters that end with 'eps' (e.g. 'afsd.eps', 'assd.ll.eps', and 'assd.ths.eps') show the epsilon corresponding to each of those ASD tests.

Value

A list of 8 elements indicating the result of each test and the epsilon values for almost tests.

See Also

[createStochasticDominance(), fsd.test(), ssd.test(), afsd.test(), assd.test()]

`sd.asd.test.all` *Performing SD and ASD tests on distribution pairs*

Description

Performing SD and ASD tests on distribution pairs

Usage

```
sd.asd.test.all(paired.dists, include.details)
```

Arguments

`paired.dists`	A data frame that includes paired distributions.
`include.details`	A Boolean.

Details

'paired.dists' is a data frame including the distributions of all pairs. 'include.details' is a Boolean value. If it is TRUE, the distributions are included in the output, otherwise they will be discarded.

The output includes the SD and ASD test results, which are the index of the dominant variable and the epsilon of the ASD tests.

Value

A data frame that includes the result of tests for each pair.

See Also

[sd.test()]

sd.screen *Create the inefficient set by an SD rule*

Description

It uses the test result to create the inefficient set.

Usage

```
sd.screen(data, test)
```

Arguments

data	A data frame, including variable pairs, distributions, and results of the tests.
test	A string that indicates the name of the test.

Details

The inefficient set includes all variable names that are dominated by at least one other variable. The domination defines as the test result of the corresponding ‘test‘ column.

Value

A character vector as the inefficient set of variables based on the ‘test‘ result.

sorting.variables *Sorting all variables based on their outcomes*

Description

Sorting all variables based on their outcomes

Usage

```
sorting.variables(variable, outcome)
```

Arguments

variable	A character vector, includes variable names.
outcome	A numeric vector, includes outcome values.

Value

A character vector, includes all variable names sorted by their outcomes.

ssd.calc *Calculates the SSD values for a prospect.*

Description

Calculates the SSD values for a prospect.

Usage

```
ssd.calc(outcome, cdf)
```

Arguments

outcome	Numeric vector, indicating the outcome values.
cdf	Numeric vector, indicating the cumulative probabilities.

Value

Numeric vector, indicating the SSD values.

ssd.plot *Drawing the SSD*

Description

It visualize the SSD values of both prospects.

Usage

```
ssd.plot(sd.obj, names = c("1", "2"))
```

Arguments

sd.obj	StochasticDominance object.
names	A character vector, including the names of prospects in order.

Details

The parameter ‘names‘ only accepts character vector, otherwise an error will be raised.

The function shows the line plot and returns its object for further modification.

Value

A list, including plot elements.

Examples

```
sd = createStochasticDominance(outcome1 = c(1,4,7),
                               outcome2 = c(2,3,5),
                               prob1 = c(1/3,1/3,1/3),
                               prob2 = c(1/6,1/6,2/3))
ssd.plot(sd, names = c('First', 'Second'))
```

---

ssd.test *Compares random prospects by SSD*

Description

It compares two random prospects by the second-order stochastic dominance (SSD).

Usage

```
ssd.test(sd.obj)
```

Arguments

sd.obj	StochasticDominance object.

Details

A prospect dominates when its SSD is below the other one. It means that all element of the SSD vector must be equal or smaller, and at least one element should be smaller for the dominant prospect.

If neither prospect dominates the other by SSD, it returns 0. It means the SSDs intersect each other.

Value

A number. Indicating dominant prospect index.

See Also

[ssd.calc()] for the calculation.

Examples

```
sd = createStochasticDominance(outcome1 = c(1,4,7),
                               outcome2 = c(2,3,5),
                               prob1 = c(1/3,1/3,1/3),
                               prob2 = c(1/6,1/6,2/3))
ssd.test(sd)
```

`StochasticDominance-class` *StochasticDominance Class*

Description

Represents two distributions (prospects) that are going to be compared using Stochastic Dominance (SD).

Details

It contains the input validation needed for comparing two prospects. For example, having sorted 'outcome', each of 'prob1' and 'prob2' adds up to one, arguments having the same lengths, and having matched probability, cumulative, and ssd arguments.

Slots

`outcome` Numeric vector. The combined outcome values in ascending order.

`prob1,prob2` Numeric vectors. Probabilities corresponding to the prospects.

`cdf1,cdf2` Numeric vectors. Cumulative values corresponding to the prospects.

`ssd1,ssd2` Numeric vectors. SSD values corresponding to the prospects.

Index