

# Package ‘RVCompare’

August 22, 2021

**Type** Package

**Title** Compare Real Valued Random Variables

**Version** 0.1.5

**Author** Etor Arza

**Maintainer** Etor Arza <etorarza@gmail.com>

**Description** A framework with tools to compare two random variables, and determine which of takes produces lower values. See the README.md at <<https://github.com/EtorArza/RVCompare>> for a quick start guide. It can compute the Cp and Cd of two probability distributions, as explained in E. Arza (2021) <<https://github.com/EtorArza/RVCompare-paper/releases>>. Given the observed samples of two random variables X\_A and X\_B, it can compute the cumulative difference-plot (see E. Arza (2021) <<https://github.com/EtorArza/RVCompare-paper/releases>> for details). Uses bootstrap and DKW-bounds to compute the confidence bands of the cumulative distributions. These two methods are described in B. Efron. (1979) <[doi:10.1214/aos/1176344552](https://doi.org/10.1214/aos/1176344552)> and P. Mas-sart (1990) <[doi:10.1214/aop/1176990746](https://doi.org/10.1214/aop/1176990746)>.

**Depends** stats (>= 3.4.4), pracma (>= 2.2.2), ggplot2 (>= 3.2.0),

**Imports** Rcpp (>= 0.11.0), utils (>= 3.4.4)

**License** CC0

**Encoding** UTF-8

**RoxygenNote** 7.1.1

**LinkingTo** Rcpp

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2021-08-22 21:30:12 UTC

## R topics documented:

CdFromDensities . . . . .	2
CdFromProbMassFunctions . . . . .	3

CpFromDensities . . . . .	4
cpp_helper_from_ranks_to_integrable_values . . . . .	5
cumulative_difference_plot . . . . .	6
getEmpiricalCumulativeDistributions . . . . .	9
get_Y_AB_bounds_bootstrap . . . . .	10
get_Y_AB_bounds_DKW . . . . .	13
isFunctionDensity . . . . .	16
isXlimsValid . . . . .	17
mixtureDensity . . . . .	17
normalDensity . . . . .	18
plot_Y_AB . . . . .	19
RVCompare . . . . .	20
sampleFromDensity . . . . .	20
uniformDensity . . . . .	21
xHasEnoughValues . . . . .	21

<b>Index</b>	<b>23</b>
--------------	-----------

---

CdFromDensities	<i>The dominance rate of X_A over X_B given the density functions.</i>
-----------------	--

---

## Description

Returns a real number in the interval [0,1] that represents the dominance rate of X\_A over X\_B. Basically, we are measuring the amount of mass of X\_A in which the cumulative distribution of X\_A is higher minus the amount of mass of X\_B in which the cumulative distribution of X\_B is higher.

## Usage

```
CdFromDensities(densityX_A, densityX_B, xlims, EPSILON = 0.001)
```

## Arguments

densityX_A	The probability density function of the random variable X_A.
densityX_B	The probability density function of the random variable X_B.
xlims	an interval that represents the domain of definition the density functions.
EPSILON	(optional, default = 1e-3) minimum difference between two values.

## Value

Returns the dominance rate of X\_A over X\_B.

## See Also

[CpFromDensities](#)

**Examples**

```

# If two symmetric distributions are centered in the same point (x = 0 in
# this case), then their Cd will be 0.5.

densityX_A <- normalDensity(0,1)
densityX_B <- uniformDensity(c(-2,2))
CdFromDensities(densityX_A, densityX_B, c(-5,5))

### Example 2 ###
# If two distributions are equal, Cd will be 0.5. Cd(X_A,X_A) = 0.5
CdFromDensities(densityX_A, densityX_A, c(-10,10))

### Example 3 ###
# example on https://etorarza.github.io/pages/2021-interactive-comparing-RV.html
tau <- 0.11
densityX_A <- normalDensity(0.05,0.0015)
densityX_B <- mixtureDensity(c(normalDensity(0.05025,0.0015),
                               normalDensity(0.04525, 0.0015)),
                              weights = c(1 - tau, tau))
plot(densityX_A, from=0.03, to=0.07, type="l", col="red", xlab="x", ylab="probability density")
curve(densityX_B, add=TRUE, col="blue", type="l", lty=2)
Cd <- CdFromDensities(densityX_A, densityX_B, c(.03,.07))
mtext(paste("Cd(X_A, X_B) =", format(round(Cd, 3), nsmall = 3)), side=3) # add Cd to plot as text
legend(x = c(0.0325, 0.045), y = c(200, 250), legend=c("X_A", "X_B"),
       col=c("red", "blue"),
       lty=1:2,
       cex=0.8) # add legend

### Example 4 ###
# The dominance factor ignores the mass of the probability where the
# distribution functions are equal.
densityX_A <- uniformDensity(c(0.1, 0.3))
densityX_B <- uniformDensity(c(-0.2,0.5))
CdFromDensities(densityX_A, densityX_B, xlims = c(-2,2))

densityX_A <- mixtureDensity(c(uniformDensity(c(0.1,0.3)), uniformDensity(c(-1,-0.5))))
densityX_B <- mixtureDensity(c(uniformDensity(c(-0.2,0.5)), uniformDensity(c(-1,-0.5))))
CdFromDensities(densityX_A, densityX_B, xlims = c(-2,2))

```

---

CdFromProbMassFunctions

*The dominance rate of X\_A over X\_B for discrete distributions, given the probability mass functions.*

---

**Description**

Returns a real number in the interval [0,1] that represents the dominance rate of X\_A over X\_B.

**Usage**

```
CdFromProbMassFunctions(pMassA, pMassB)
```

**Arguments**

pMassA            The probability mass function where pMassA[[i]] is the probability of x\_i, p\_A(x\_i).  
 pMassB            The probability mass function where pMassB[[i]] is the probability of x\_i, p\_B(x\_i).

**Value**

Returns the dominance rate of X\_A over X\_B for discrete random variables.

**Examples**

```
CdFromProbMassFunctions(c(0.2,0.6,0.2), c(0.3,0.3,0.4))
# > 0.6
# Notice how adding additional mass with the same cumulative distribution in both
# random variables does not change the result.
CdFromProbMassFunctions(c(0.2,0.6,0.2,0.2,0.2)/1.4, c(0.3,0.3,0.4,0.2,0.2)/1.4)
# > 0.6
```

---

CpFromDensities

*The probability that X\_A < X\_B given the density functions.*

---

**Description**

Returns a real number in the interval [0,1] that represents the probability that a sample observed from X\_A is lower than a sample observed from X\_B.

**Usage**

```
CpFromDensities(densityX_A, densityX_B, xlims)
```

**Arguments**

densityX\_A        The probability density function of the random variable X\_A.  
 densityX\_B        The probability density function of the random variable X\_B.  
 xlims             an interval that represents the domain of definition the density functions.

**Value**

Returns the probability that X\_A < X\_B.

**See Also**

[CdFromDensities](#)

**Examples**

```
### Example 1 ###
# If two symmetric distributions are centered in the same point (x = 0 in
# this case), then their Cp will be 0.5.
densityX_A <- normalDensity(0,1)
densityX_B <- uniformDensity(c(-2,2))
Cp = CpFromDensities(densityX_A, densityX_B, c(-5,5))
plot(densityX_A, from=-5, to=5, type="l", col="red", xlab="x", ylab="probability density")
curve(densityX_B, add=TRUE, col="blue", type="l", lty=2)
mtext(paste("Cp(X_A, X_B) =", format(round(Cp, 3), nsmall = 3)), side=3) # add Cp to plot as text
legend(x = c(-4.5, -2), y = c(0.325, 0.4), legend=c("X_A", "X_B"),
      col=c("red", "blue"),
      lty=1:2, cex=0.8) # add legend
```

```
### Example 2 ###
# If two distributions are equal, Cp will be 0.5. Cp(X_A,X_A) = 0.5
CpFromDensities(densityX_A, densityX_A, c(-10,10))
```

```
### Example 3 ###
densityX_A <- normalDensity(-2,1)
densityX_B <- uniformDensity(c(-2,2))
# Cp(X_A,X_B) = 1 - Cp(X_B, X_A)
CpFromDensities(densityX_A, densityX_B, c(-8,4))
1 - CpFromDensities(densityX_B, densityX_A, c(-8,4))
```

---

cpp\_helper\_from\_ranks\_to\_integrable\_values

*Helper function for from\_ranks\_to\_integrable\_values*

---

**Description**

Helper function for from\_ranks\_to\_integrable\_values

**Usage**

```
cpp_helper_from_ranks_to_integrable_values(rank_interval_mult, j_max)
```

**Arguments**

rank\_interval\_mult  
the value of the normalized density of Y\_A.

j\_max  
maximum number of sample points.

---

cumulative\_difference\_plot

*Generate the cumulative difference-plot*

---

### Description

Generate the cumulative difference-plot given the observed samples using the bootstrap method.

### Usage

```
cumulative_difference_plot(
  X_A_observed,
  X_B_observed,
  isMinimizationProblem,
  labelA = "X_A",
  labelB = "X_B",
  alpha = 0.05,
  EPSILON = 1e-20,
  nOfBootstrapSamples = 1000,
  ignoreMinimumLengthCheck = FALSE
)
```

### Arguments

X_A_observed	array of the observed samples (real values) of X_A.
X_B_observed	array of the observed samples (real values) of X_B, it needs to have the same length as X_A.
isMinimizationProblem	a boolean value where TRUE represents that lower values are preferred to larger values.
labelA	(optional, default value "X_A") the label corresponding to X_A.
labelB	(optional, default value "X_B") the label corresponding to X_B.
alpha	(optional, default value 0.05) the error of the confidence interval. If alpha = 0.05 then we have 95 percent confidence interval.
EPSILON	(optional, default value 1e-20) minimum difference between two values to be considered different.
nOfBootstrapSamples	(optional, default value 1e3) how many bootstrap samples to average. Increases computation time.
ignoreMinimumLengthCheck	(optional, default value FALSE) whether to skip the check for a minimum length of 100 in X_A_observed and X_A_observed.

### Value

returns and shows the cumulative difference-plot

**Examples**

```
### Example 1 ###
X_A_observed <- rnorm(100, mean = 2, sd = 1)
X_B_observed <- rnorm(100, mean = 2.1, sd = 0.5)

cumulative_difference_plot(X_A_observed, X_B_observed, TRUE, labelA="X_A", labelB="X_B")

### Example 2 ###
# Comparing the optimization algorithms PL-EDA and PL-GS
# with 400 samples each.
PL_EDA_fitness <- c(
52235, 52485, 52542, 52556, 52558, 52520, 52508, 52491, 52474, 52524,
52414, 52428, 52413, 52457, 52437, 52449, 52534, 52531, 52476, 52434,
52492, 52554, 52520, 52500, 52342, 52520, 52392, 52478, 52422, 52469,
52421, 52386, 52373, 52230, 52504, 52445, 52378, 52554, 52475, 52528,
52508, 52222, 52416, 52492, 52538, 52192, 52416, 52213, 52478, 52496,
52444, 52524, 52501, 52495, 52415, 52151, 52440, 52390, 52428, 52438,
52475, 52177, 52512, 52530, 52493, 52424, 52201, 52484, 52389, 52334,
52548, 52560, 52536, 52467, 52392, 51327, 52506, 52473, 52087, 52502,
52533, 52523, 52485, 52535, 52502, 52577, 52508, 52463, 52530, 52507,
52472, 52400, 52511, 52528, 52532, 52526, 52421, 52442, 52532, 52505,
52531, 52644, 52513, 52507, 52444, 52471, 52474, 52426, 52526, 52564,
52512, 52521, 52533, 52511, 52416, 52414, 52425, 52457, 52522, 52508,
52481, 52439, 52402, 52442, 52512, 52377, 52412, 52432, 52506, 52524,
52488, 52494, 52531, 52471, 52616, 52482, 52499, 52386, 52492, 52484,
52537, 52517, 52536, 52449, 52439, 52410, 52417, 52402, 52406, 52217,
52484, 52418, 52550, 52513, 52530, 51667, 52185, 52089, 51853, 52511,
52051, 52584, 52475, 52447, 52390, 52506, 52514, 52452, 52526, 52502,
52422, 52411, 52171, 52437, 52323, 52488, 52546, 52505, 52563, 52457,
52502, 52503, 52126, 52537, 52435, 52419, 52300, 52481, 52419, 52540,
52566, 52547, 52476, 52448, 52474, 52438, 52430, 52363, 52484, 52455,
52420, 52385, 52152, 52505, 52457, 52473, 52503, 52507, 52429, 52513,
52433, 52538, 52416, 52479, 52501, 52485, 52429, 52395, 52503, 52195,
52380, 52487, 52498, 52421, 52137, 52493, 52403, 52511, 52409, 52479,
52400, 52498, 52482, 52440, 52541, 52499, 52476, 52485, 52294, 52408,
52426, 52464, 52535, 52512, 52516, 52531, 52449, 52507, 52485, 52491,
52499, 52414, 52403, 52398, 52548, 52536, 52410, 52549, 52454, 52534,
52468, 52483, 52239, 52502, 52525, 52328, 52467, 52217, 52543, 52391,
52524, 52474, 52509, 52496, 52432, 52532, 52493, 52503, 52508, 52422,
52459, 52477, 52521, 52515, 52469, 52416, 52249, 52537, 52494, 52393,
52057, 52513, 52452, 52458, 52518, 52520, 52524, 52531, 52439, 52530,
52422, 52649, 52481, 52256, 52428, 52425, 52458, 52488, 52502, 52373,
52426, 52441, 52471, 52468, 52465, 52265, 52455, 52501, 52340, 52457,
52275, 52527, 52574, 52474, 52487, 52416, 52634, 52514, 52184, 52430,
52462, 52392, 52529, 52178, 52495, 52438, 52539, 52430, 52459, 52312,
52437, 52637, 52511, 52563, 52270, 52341, 52436, 52515, 52480, 52569,
52490, 52453, 52422, 52443, 52419, 52512, 52447, 52425, 52509, 52180,
52521, 52566, 52060, 52425, 52480, 52454, 52501, 52536, 52143, 52432,
52451, 52548, 52508, 52561, 52515, 52502, 52468, 52373, 52511, 52516,
```

```

52195, 52499, 52534, 52453, 52449, 52431, 52473, 52553, 52444, 52459,
52536, 52413, 52537, 52537, 52501, 52425, 52507, 52525, 52452, 52499
)
PL_GS_fitness <- c(
52476, 52211, 52493, 52484, 52499, 52500, 52476, 52483, 52431, 52483,
52515, 52493, 52490, 52464, 52478, 52440, 52482, 52498, 52460, 52219,
52444, 52479, 52498, 52481, 52490, 52470, 52498, 52521, 52452, 52494,
52451, 52429, 52248, 52525, 52513, 52489, 52448, 52157, 52449, 52447,
52476, 52535, 52464, 52453, 52493, 52438, 52489, 52462, 52219, 52223,
52514, 52476, 52495, 52496, 52502, 52538, 52491, 52457, 52471, 52531,
52488, 52441, 52467, 52483, 52476, 52494, 52485, 52507, 52224, 52464,
52503, 52495, 52518, 52490, 52508, 52505, 52214, 52506, 52507, 52207,
52531, 52492, 52515, 52497, 52476, 52490, 52436, 52495, 52437, 52494,
52513, 52483, 52522, 52496, 52196, 52525, 52490, 52506, 52498, 52250,
52524, 52469, 52497, 52519, 52437, 52481, 52237, 52436, 52508, 52518,
52490, 52501, 52508, 52476, 52520, 52435, 52463, 52481, 52486, 52489,
52482, 52496, 52499, 52443, 52497, 52464, 52514, 52476, 52498, 52496,
52498, 52530, 52203, 52482, 52441, 52493, 52532, 52518, 52474, 52498,
52512, 52226, 52538, 52477, 52508, 52243, 52533, 52463, 52440, 52246,
52209, 52488, 52530, 52195, 52487, 52494, 52508, 52505, 52444, 52515,
52499, 52428, 52498, 52244, 52520, 52463, 52187, 52484, 52517, 52504,
52511, 52530, 52519, 52514, 52532, 52203, 52485, 52439, 52496, 52443,
52503, 52520, 52516, 52478, 52473, 52505, 52480, 52196, 52492, 52527,
52490, 52493, 52252, 52470, 52493, 52533, 52506, 52496, 52519, 52492,
52509, 52530, 52213, 52499, 52492, 52528, 52499, 52526, 52521, 52488,
52485, 52502, 52515, 52470, 52207, 52494, 52527, 52442, 52200, 52485,
52489, 52499, 52488, 52486, 52232, 52477, 52485, 52490, 52524, 52470,
52504, 52501, 52497, 52489, 52152, 52527, 52487, 52501, 52504, 52494,
52484, 52213, 52449, 52490, 52525, 52476, 52540, 52463, 52200, 52471,
52479, 52504, 52526, 52533, 52473, 52475, 52518, 52507, 52500, 52499,
52512, 52478, 52523, 52453, 52488, 52523, 52240, 52505, 52532, 52504,
52444, 52194, 52514, 52474, 52473, 52526, 52437, 52536, 52491, 52523,
52529, 52535, 52453, 52522, 52519, 52446, 52500, 52490, 52459, 52467,
52456, 52490, 52521, 52484, 52508, 52451, 52231, 52488, 52485, 52215,
52493, 52475, 52474, 52508, 52524, 52477, 52514, 52452, 52491, 52473,
52441, 52520, 52471, 52466, 52475, 52439, 52483, 52491, 52204, 52500,
52488, 52489, 52519, 52495, 52448, 52453, 52466, 52462, 52489, 52471,
52484, 52483, 52501, 52486, 52494, 52473, 52481, 52502, 52516, 52223,
52490, 52447, 52222, 52469, 52509, 52194, 52490, 52484, 52446, 52487,
52476, 52509, 52496, 52459, 52474, 52501, 52516, 52223, 52487, 52468,
52534, 52522, 52474, 52227, 52450, 52506, 52193, 52429, 52496, 52493,
52493, 52488, 52190, 52509, 52434, 52469, 52510, 52481, 52520, 52504,
52230, 52500, 52487, 52517, 52473, 52488, 52450, 52203, 52215, 52490,
52479, 52515, 52210, 52485, 52516, 52504, 52521, 52499, 52503, 52526)
# Considering that the LOP is a maximization problem, we need isMinimizationProblem=FALSE.

cumulative_difference_plot(PL_EDA_fitness,
                           PL_GS_fitness,
                           isMinimizationProblem=FALSE,
                           labelA="PL-EDA",
                           labelB="PL-GS")

```



---

```
getEmpiricalCumulativeDistributions
```

*Get the empirical distribution from samples.*

---

### Description

Given the observed samples of X\_A (or X\_B) returns the empirical cumulative distribution function of Y\_A (or Y\_B)

### Usage

```
getEmpiricalCumulativeDistributions(
  X_A_observed,
  X_B_observed,
  nOfEstimationPoints,
  EPSILON,
  trapezoid = TRUE
)
```

### Arguments

X\_A\_observed    array of the observed samples (real values) of X\_A.  
 X\_B\_observed    array of the observed samples (real values) of X\_B.  
 nOfEstimationPoints    the number of points in the interval [0,1] in which the cumulative density is estimated + 2.  
 EPSILON    (optional, default value 1e-20) minimum difference between two values to be considered different.  
 trapezoid    (optional, default TRUE) if trapezoid=FALSE the non smooth empirical distribution is given. This is what the WDK uses the empirical as the estimation.

### Value

a list with two fields: the empirical distributions of X'A and X'B.

### Examples

```
### Example 1 ###
c <- getEmpiricalCumulativeDistributions(c(1:5),c(1:3,2:3), 170, EPSILON=1e-20, trapezoid=FALSE)
plot(c$p, c$Y_A_cumulative_estimation, type="l")
lines(x=c$p, y=c$Y_B_cumulative_estimation, col="red")
```

---

```
get_Y_AB_bounds_bootstrap
```

*Estimate Y\_A and Y\_B bounds with bootstrap*

---

### Description

Estimate the confidence intervals for the cumulative distributions of Y\_A and Y\_B using bootstrap. Much slower than the Dvoretzky–Kiefer–Wolfowitz approach.

### Usage

```
get_Y_AB_bounds_bootstrap(  
  X_A_observed,  
  X_B_observed,  
  alpha = 0.05,  
  EPSILON = 1e-20,  
  nOfBootstrapSamples = 1000,  
  ignoreMinimumLengthCheck = FALSE  
)
```

### Arguments

X_A_observed	array of the observed samples (real values) of X_A.
X_B_observed	array of the observed samples (real values) of X_B, it needs to have the same length as X_A.
alpha	(optional, default value 0.05) the error of the confidence interval. If alpha = 0.05 then we have 95 percent confidence interval.
EPSILON	(optional, default value 1e-20) minimum difference between two values to be considered different.
nOfBootstrapSamples	(optional, default value 1e3) how many bootstrap samples to average. Increases computation time.
ignoreMinimumLengthCheck	(optional, default value FALSE) wether to check for a minimum length in X_A and X_B.

### Value

Returns a list with the following fields:

- p: values in the interval [0,1] that represent the nOfEstimationPoints points in which the densities are estimated. Useful for plotting.
- Y\_A\_cumulative\_estimation: an array with the estimated cumulative diustribution function of Y\_A from 0 to p[[i]].
- Y\_A\_cumulative\_upper: an array with the upper bounds of confidence 1 - alpha of the cumulative density of Y\_A

- Y\_A\_cumulative\_lower: an array with the lower bounds of confidence  $1 - \alpha$  of the cumulative density of Y\_A
- Y\_B\_cumulative\_estimation: The same as Y\_A\_cumulative\_estimation for Y\_B.
- Y\_B\_cumulative\_upper: The same as Y\_A\_cumulative\_upper for Y\_B
- Y\_B\_cumulative\_lower: The same as Y\_A\_cumulative\_lower for Y\_B
- diff\_estimation: Y\_A\_cumulative\_estimation - Y\_B\_cumulative\_estimation
- diff\_upper: an array with the upper bounds of confidence  $1 - \alpha$  of the difference between the cumulative distributions
- diff\_lower: an array with the lower bounds of confidence  $1 - \alpha$  of the difference between the cumulative distributions

## Examples

```
library(ggplot2)

### Example 1 ###
X_A_observed <- rnorm(100, mean = 2, sd = 1)
X_B_observed <- rnorm(100, mean = 2.1, sd = 0.5)

res <- get_Y_AB_bounds_bootstrap(X_A_observed, X_B_observed)

fig1 = plot_Y_AB(res, plotDifference=FALSE)+ ggplot2::ggtitle("Example 1")
print(fig1)

### Example 2 ###
# Comparing the estimations with the actual distributions for two normal distributions.
#####
## sample size = 100 #####
#####
X_A_observed <- rnorm(100,mean = 1, sd = 1)
X_B_observed <- rnorm(100,mean = 1.3, sd = 0.5)
res <- get_Y_AB_bounds_bootstrap(X_A_observed, X_B_observed)

X_A_observed_large_sample <- sort(rnorm(1e4, mean = 1, sd = 1))
X_B_observed_large_sample <- sort(rnorm(1e4, mean = 1.3, sd = 0.5))
actualDistributions <- getEmpiricalCumulativeDistributions(
  X_A_observed_large_sample,
  X_B_observed_large_sample,
  nOfEstimationPoints=1e4,
  EPSILON=1e-20)

actualDistributions$Y_A_cumulative_estimation <- lm(Y_A_cumulative_estimation ~
  p + I(p^2) + I(p^3)+ I(p^4)+ I(p^5)+ I(p^6)+I(p^7)+ I(p^8),
  data = actualDistributions)$fitted.values
actualDistributions$Y_B_cumulative_estimation <- lm(Y_B_cumulative_estimation ~
  p + I(p^2) + I(p^3)+ I(p^4)+ I(p^5)+ I(p^6)+I(p^7)+ I(p^8),
```

```

    data = actualDistributions)$fitted.values

fig = plot_Y_AB(res, plotDifference=FALSE) +

geom_line(data=as.data.frame(actualDistributions),
aes(x=p, y=Y_A_cumulative_estimation, colour = "Actual Y_A", linetype="Actual Y_A")) +

geom_line(data=as.data.frame(actualDistributions),
aes(x=p, y=Y_B_cumulative_estimation, colour = "Actual Y_B", linetype="Actual Y_B")) +

scale_colour_manual("", breaks = c("X_A", "X_B", "Actual Y_A", "Actual Y_B"),
values = c("X_A"="#00BFC4", "X_B"="#F8766D", "Actual Y_A"="#0000FF", "Actual Y_B"="#FF0000"))+

scale_linetype_manual("", breaks = c("X_A", "X_B", "Actual Y_A", "Actual Y_B"),
values = c("X_A"="solid", "X_B"="dashed", "Actual Y_A"="solid", "Actual Y_B"="solid"))+

ggtitle("100 samples used in the estimation")
print(fig)

#####
## sample size = 300 #####
#####
X_A_observed <- rnorm(300,mean = 1, sd = 1)
X_B_observed <- rnorm(300,mean = 1.3, sd = 0.5)
res <- get_Y_AB_bounds_bootstrap(X_A_observed, X_B_observed)

X_A_observed_large_sample <- sort(rnorm(1e4, mean = 1, sd = 1))
X_B_observed_large_sample <- sort(rnorm(1e4, mean = 1.3, sd = 0.5))
actualDistributions <- getEmpiricalCumulativeDistributions(
  X_A_observed_large_sample,
  X_B_observed_large_sample,
  nOfEstimationPoints=1e4,
  EPSILON=1e-20)

actualDistributions$Y_A_cumulative_estimation <- lm(Y_A_cumulative_estimation ~
  p + I(p^2) + I(p^3)+ I(p^4)+ I(p^5)+ I(p^6)+I(p^7)+ I(p^8),
  data = actualDistributions)$fitted.values

actualDistributions$Y_B_cumulative_estimation <- lm(Y_B_cumulative_estimation ~
  p + I(p^2) + I(p^3)+ I(p^4)+ I(p^5)+ I(p^6)+I(p^7)+ I(p^8),
  data = actualDistributions)$fitted.values

fig = plot_Y_AB(res, plotDifference=FALSE) +

geom_line(data=as.data.frame(actualDistributions),
aes(x=p, y=Y_A_cumulative_estimation, colour = "Actual Y_A", linetype="Actual Y_A")) +

geom_line(data=as.data.frame(actualDistributions),
aes(x=p, y=Y_B_cumulative_estimation, colour = "Actual Y_B", linetype="Actual Y_B")) +

scale_colour_manual("", breaks = c("X_A", "X_B", "Actual Y_A", "Actual Y_B"),
values = c("X_A"="#00BFC4", "X_B"="#F8766D", "Actual Y_A"="#0000FF", "Actual Y_B"="#FF0000"))+

```

```

scale_linetype_manual("", breaks = c("X_A", "X_B", "Actual Y_A", "Actual Y_B"),
values = c("X_A"="solid", "X_B"="dashed", "Actual Y_A"="solid", "Actual Y_B"="solid"))+

ggtitle("300 samples used in the estimation")
print(fig)

```

---

get\_Y\_AB\_bounds\_DKW     *Estimate Y\_A and Y\_B bounds with Dvoretzky–Kiefer–Wolfowitz*

---

### Description

Estimate the confidence intervals for the cumulative distributions of Y\_A and Y\_B with Dvoretzky–Kiefer–Wolfowitz.

### Usage

```

get_Y_AB_bounds_DKW(
  X_A_observed,
  X_B_observed,
  nOfEstimationPoints = 1000,
  alpha = 0.05,
  EPSILON = 1e-20,
  ignoreMinimumLengthCheck = FALSE
)

```

### Arguments

X\_A\_observed     array of the observed samples (real values) of X\_A.

X\_B\_observed     array of the observed samples (real values) of X\_B.

nOfEstimationPoints  
                   (optional, default 1000) the number of points in the interval [0,1] in which the density is estimated.

alpha             (optional, default value 0.05) the error of the confidence interval. If alpha = 0.05 then we have 95 percent confidence interval.

EPSILON           (optional, default value 1e-20) minimum difference between two values to be considered different.

ignoreMinimumLengthCheck  
                   (optional, default value FALSE) wether to check for a minimum length in X\_A and X\_B.

**Value**

Returns a list with the following fields:

- p: values in the interval [0,1] that represent the nOfEstimationPoints points in which the densities are estimated. Useful for plotting.
- Y\_A\_cumulative\_estimation: an array with the empirical cumulative distribution function of Y\_A from 0 to p[[i]].
- Y\_A\_cumulative\_upper: an array with the upper bounds of confidence 1 - alpha of the cumulative density of Y\_A
- Y\_A\_cumulative\_lower: an array with the lower bounds of confidence 1 - alpha of the cumulative density of Y\_A
- Y\_B\_cumulative\_estimation: The same as Y\_A\_cumulative\_estimation for Y\_B.
- Y\_B\_cumulative\_upper: The same as Y\_A\_cumulative\_upper for Y\_B
- Y\_B\_cumulative\_lower: The same as Y\_A\_cumulative\_lower for Y\_B
- diff\_estimation: Y\_A\_cumulative\_estimation - Y\_B\_cumulative\_estimation
- diff\_upper: an array with the upper bounds of confidence 1 - alpha of the difference between the cumulative distributions
- diff\_lower: an array with the lower bounds of confidence 1 - alpha of the difference between the cumulative distributions

**Examples**

```
library(ggplot2)
### Example 1 ###
X_A_observed <- rnorm(100, mean = 2, sd = 1)
X_B_observed <- rnorm(100, mean = 2.1, sd = 0.5)
res <- get_Y_AB_bounds_DKW(X_A_observed, X_B_observed)
fig1 = plot_Y_AB(res, plotDifference=FALSE) + ggtitle("Example 1")
print(fig1)

### Example 2 ###
# Comparing the estimations with the actual distributions for two normal distributions.
#####
## sample size = 100 #####
#####
X_A_observed <- rnorm(100,mean = 1, sd = 1)
X_B_observed <- rnorm(100,mean = 1.3, sd = 0.5)
res <- get_Y_AB_bounds_DKW(X_A_observed, X_B_observed)

X_A_observed_large_sample <- sort(rnorm(1e4, mean = 1, sd = 1))
X_B_observed_large_sample <- sort(rnorm(1e4, mean = 1.3, sd = 0.5))
actualDistributions <- getEmpiricalCumulativeDistributions(X_A_observed_large_sample,
  X_B_observed_large_sample, nOfEstimationPoints=1e4, EPSILON=1e-20)

actualDistributions$Y_A_cumulative_estimation <- lm(Y_A_cumulative_estimation ~
  p + I(p^2) + I(p^3)+ I(p^4)+ I(p^5)+ I(p^6)+I(p^7)+ I(p^8),
```

```

      data = actualDistributions)$fitted.values
actualDistributions$Y_B_cumulative_estimation <- lm(Y_B_cumulative_estimation ~
  p + I(p^2) + I(p^3)+ I(p^4)+ I(p^5)+ I(p^6)+I(p^7)+ I(p^8),
  data = actualDistributions)$fitted.values

fig = plot_Y_AB(res, plotDifference=FALSE) +

geom_line(data=as.data.frame(actualDistributions),
  aes(x=p, y=Y_A_cumulative_estimation, colour = "Actual Y_A", linetype="Actual Y_A")) +

geom_line(data=as.data.frame(actualDistributions),
  aes(x=p, y=Y_B_cumulative_estimation, colour = "Actual Y_B", linetype="Actual Y_B")) +

scale_colour_manual("", breaks = c("X_A", "X_B", "Actual Y_A", "Actual Y_B"),
  values = c("X_A"="#00BFC4", "X_B"="#F8766D", "Actual Y_A"="#0000FF", "Actual Y_B"="#FF0000"))+

scale_linetype_manual("", breaks = c("X_A", "X_B", "Actual Y_A", "Actual Y_B"),
  values = c("X_A"="solid", "X_B"="dashed", "Actual Y_A"="solid", "Actual Y_B"="solid"))+

ggtitle("100 samples used in the estimation")
print(fig)

#####
## sample size = 300 #####
#####
X_A_observed <- rnorm(300,mean = 1, sd = 1)
X_B_observed <- rnorm(300,mean = 1.3, sd = 0.5)
res <- get_Y_AB_bounds_DKW(X_A_observed, X_B_observed)

X_A_observed_large_sample <- sort(rnorm(1e4, mean = 1, sd = 1))
X_B_observed_large_sample <- sort(rnorm(1e4, mean = 1.3, sd = 0.5))
actualDistributions <- getEmpiricalCumulativeDistributions(X_A_observed_large_sample,
  X_B_observed_large_sample, nOfEstimationPoints=1e4, EPSILON=1e-20)

actualDistributions$Y_A_cumulative_estimation <- lm(Y_A_cumulative_estimation ~
  p + I(p^2) + I(p^3)+ I(p^4)+ I(p^5)+ I(p^6)+I(p^7)+ I(p^8),
  data = actualDistributions)$fitted.values
actualDistributions$Y_B_cumulative_estimation <- lm(Y_B_cumulative_estimation ~
  p + I(p^2) + I(p^3)+ I(p^4)+ I(p^5)+ I(p^6)+I(p^7)+ I(p^8),
  data = actualDistributions)$fitted.values

fig = plot_Y_AB(res, plotDifference=FALSE) +

geom_line(data=as.data.frame(actualDistributions),
  aes(x=p, y=Y_A_cumulative_estimation, colour = "Actual Y_A", linetype="Actual Y_A")) +

geom_line(data=as.data.frame(actualDistributions),
  aes(x=p, y=Y_B_cumulative_estimation, colour = "Actual Y_B", linetype="Actual Y_B")) +

scale_colour_manual("", breaks = c("X_A", "X_B", "Actual Y_A", "Actual Y_B"),
  values = c("X_A"="#00BFC4", "X_B"="#F8766D", "Actual Y_A"="#0000FF", "Actual Y_B"="#FF0000"))+

```

```
scale_linetype_manual("", breaks = c("X_A", "X_B", "Actual Y_A", "Actual Y_B"),
values = c("X_A"="solid", "X_B"="dashed", "Actual Y_A"="solid", "Actual Y_B"="solid"))+
ggtitle("300 samples used in the estimation")
print(fig)
```

---

isFunctionDensity	<i>Check if a function is a (non-discrete) probability density function in a given domain.</i>
-------------------	--

---

### Description

This function checks if an input function  $f$  is a non-discrete probability density function. For this to be the case, the function needs to only return real values. The function also needs to be bounded, positive, and its integral in the domain of definition needs to be 1.

### Usage

```
isFunctionDensity(f, xlims, tol = 0.001)
```

### Arguments

$f$	the function to be checked.
$xlims$	an interval that represents the domain of definition of $f$ .
$tol$	(optional parameter, default = 0.001) the integral of $f$ is allowed to be in the interval $(1-tol, 1+tol)$ , to account for some reasonable error in the integration.

### Value

Returns True if the function is a non discrete probability density function. Otherwise, returns False.

### Examples

```
dist1 <- normalDensity(0,1)
# the integral of the density of the normal distribution is too low in the interval (-2,2)
isFunctionDensity(dist1, c(-2,2))
isFunctionDensity(dist1, c(-5,5)) # it is close enough from 1 in the interval (-5,5)
dist2 <- uniformDensity(c(0,1))
isFunctionDensity(dist2, xlims=c(-2,2))
isFunctionDensity(dist2, xlims=c(0.5,2)) # the integral is not 1

dist3 <- function(x) 0.5/sqrt(x)
# The integral of the function being 1 is not enough to be considered a density function.
# It also needs to be bounded.
isFunctionDensity(dist3, c(1e-14,1))
```



---

isXlimsValid	<i>Check if xlims is a tuple that represents a valid bounded interval in the real space.</i>
--------------	--

---

**Description**

Check if xlims is a tuple that represents a valid bounded interval in the real space.

**Usage**

```
isXlimsValid(xlims)
```

**Arguments**

xlims            the tuple to be checked.

**Value**

TRUE if it is a valid tuple. Otherwise prints error message and returns FALSE

---

mixtureDensity	<i>A mixture of two or more distributions</i>
----------------	---

---

**Description**

Returns the density function of the mixture distribution. The returned function is a single parameter function that returns the probability of the mixture in that point.

**Usage**

```
mixtureDensity(densities, weights = NULL)
```

**Arguments**

densities        the probability density functions to be combined.  
weights         (optional) the weights of the distributions in the mixture. If it is not give, equal weights are assumed.

**Value**

Returns a callable function with a single parameter that returns the probability of the mixture distribution each point.

## Examples

```
#If parameter weights not given, equal weights are assumed.
dist1 <- mixtureDensity(c(normalDensity(-2,1), normalDensity(2,1)))
plot(dist1, xlim = c(-5,5), xlab="x", ylab = "Probability density",
      main="Mixture of two Gaussians with equal weights", cex.main=0.85)

dist2 <- mixtureDensity(c(normalDensity(-2,1), normalDensity(2,1)), weights=c(0.8,0.2))
plot(dist2, xlim = c(-5,5), xlab="x", ylab = "Probability density",
      main="Mixture of two Gaussians with different weights", cex.main=0.85)
```

---

normalDensity

*The probability density function of the normal distribution*

---

## Description

Returns the density function of the normal distribution with mean  $\mu$  and standard deviation  $\sigma$ . The returned function is a single parameter function that returns the probability of the normal distribution in that point. It is just a convenient wrapper of `dnorm` from the package 'stat' with some parameter checks.

## Usage

```
normalDensity(mu, sigma)
```

## Arguments

<code>mu</code>	the mean of the normal distribution.
<code>sigma</code>	the standard deviation of the normal distribution.

## Value

Returns a callable function with a single parameter that describes the probability of the normal distribution in that point.

## See Also

Other probability density distributions: [uniformDensity\(\)](#)

## Examples

```
dist <- normalDensity(0,1)
dist(0)
```

---

`plot_Y_AB`*Plot the estimated cdf of Y\_A and Y\_B or their difference*

---

### Description

returns a ggplot2 with the estimations of Y\_A and Y\_B or the difference in cumulative distribution function.

### Usage

```
plot_Y_AB(  
  estimated_Y_AB_bounds,  
  labels = c("X_A", "X_B"),  
  plotDifference = TRUE  
)
```

### Arguments

`estimated_Y_AB_bounds` the bounds estimated with `get_Y_AB_bounds_bootstrap` or `get_Y_AB_bounds_DKW`.

`labels` (optional, default=c("X\_A","X\_B")) a string vector of length 2 with the labels of X\_A and X\_B, in that order.

`plotDifference` (optional, default=TRUE) plots the difference (Y\_A - Y\_B) instead of each of the random variables on their own.

### Value

the ggplot figure object.

### Examples

```
### Example 1 ###  
  
X_A_observed <- rnorm(800,mean = 1, sd = 1)  
X_B_observed <- rnorm(800,mean = 1.3, sd = 0.5)  
res <- get_Y_AB_bounds_DKW(X_A_observed, X_B_observed)  
densitiesPlot = plot_Y_AB(res, plotDifference=TRUE)  
print(densitiesPlot)
```

RVCompare

*RVCompare: Compare Real Valued Random Variables***Description**

A framework with tools to compare two random variables, and determine which of them produces lower values. It can compute the Cp and Cd of theoretical of probability distributions, as explained in E. Arza (2021) <<https://github.com/EtorArza/RVCompare-paper/releases>>. Given the observed samples of two random variables X\_A and X\_B, it can compute the confidence bands of the cumulative distributions of X'\_A and X'\_B (see E. Arza (2021) <<https://github.com/EtorArza/RVCompare-paper>> for details) based on the observed samples of X\_A and X\_B. Uses bootstrap and DKW-bounds to compute the confidence bands of the cumulative distributions. These two methods are described in B. Efron. (1979) <[doi:10.1214/aos/1176344552](https://doi.org/10.1214/aos/1176344552)> and P. Massart (1990) <[doi:10.1214/aop/1176990746](https://doi.org/10.1214/aop/1176990746)>.

**Author(s)**Etor Arza <[etorarza@gmail.com](mailto:etorarza@gmail.com)>

sampleFromDensity

*Get sample given the density function***Description**

Returns an array with samples given the probability density function.

**Usage**

```
sampleFromDensity(density, nSamples, xlims, nIntervals = 1e+05)
```

**Arguments**

density	the probability density function.
nSamples	the number of samples to generate.
xlims	the domain of definition of the random variable.
nIntervals	(optional, default = 1e4) the number of intervals from which to draw samples. A higher value implies more accuracy but also more computation time.

**Value**

Returns an array of samples.

**Examples**

```
normDens <- normalDensity(0,1)
samples <- sampleFromDensity(normDens, 1e4, c(-4,4))
hist(samples, breaks=20)
```

---

uniformDensity	<i>The probability density function of the uniform distribution</i>
----------------	---

---

**Description**

Returns the density function of the uniform distribution in the interval (xlims[[1]], xlims[[2]]). The returned function is a single parameter function that returns the probability of the uniform distribution in that point. It is just a convenient wrapper of dunif from the package 'stat' with some parameter checks.

**Usage**

```
uniformDensity(xlims)
```

**Arguments**

xlims            a tuple representing the interval of nonzero probability of the distribution.

**Value**

Returns a callable function with a single parameter that returns the probability of the uniform distribution in each point.

**See Also**

Other probability density distributions: [normalDensity\(\)](#)

**Examples**

```
dist <- uniformDensity(c(-2,2))
dist(-3)
dist(0)
dist(1)
```

---

xHasEnoughValues	<i>Check for enough values.</i>
------------------	---------------------------------

---

**Description**

This function checks if there are at least minRequiredValues values in the introduced vector.

**Usage**

```
xHasEnoughValues(X, minRequiredValues)
```

**Arguments**

X                    the array with the values.  
minRequiredValues                    the minimum number values required to return TRUE.

**Value**

Returns TRUE if the values are OK. FALSE, if there are not enough values.

**Examples**

```
xHasEnoughValues(c(1,2,2,3,1,5,8,9,67,8.5,4,8.3), 6)
```

# Index

## \* **probability density distributions**

- normalDensity, [18](#)
- uniformDensity, [21](#)

CdFromDensities, [2, 4](#)

CdFromProbMassFunctions, [3](#)

CpFromDensities, [2, 4](#)

cpp\_helper\_from\_ranks\_to\_integrable\_values,  
[5](#)

cumulative\_difference\_plot, [6](#)

get\_Y\_AB\_bounds\_bootstrap, [10, 19](#)

get\_Y\_AB\_bounds\_DKW, [13, 19](#)

getEmpiricalCumulativeDistributions, [9](#)

isFunctionDensity, [16](#)

isXlimsValid, [17](#)

mixtureDensity, [17](#)

normalDensity, [18, 21](#)

plot\_Y\_AB, [19](#)

RVCompare, [20](#)

sampleFromDensity, [20](#)

uniformDensity, [18, 21](#)

xHasEnoughValues, [21](#)