Package ‘concurve’

October 12, 2020

**Type** Package

**Title** Computes & Plots Compatibility (Confidence), Surprisal, & Likelihood Distributions

**Version** 2.7.7

**Date** 2020-10-12

**Maintainer** Zad Rafi <zad@lesslikely.com>

**Description**

Computes compatibility (confidence) distributions along with their corresponding P-values, S-values, and likelihoods. The intervals can be plotted to form the distributions themselves. Functions can be compared to one another to see how much they overlap. Results can be exported to Microsoft Word, Powerpoint, and TeX documents. The package currently supports resampling methods, computing differences, generalized linear models, mixed-effects models, survival analysis, and meta-analysis. These methods are discussed by Schweder T, Hjort NL. (2016, ISBN:9781316445051) and Rafi Z, Greenland S. (2020) <doi:10.1186/s12874-020-01105-9>.

**License** GPL-3 | file LICENSE

**URL** https://data.lesslikely.com/concurve/, https://github.com/zadrafi/concurve

**BugReports** https://github.com/zadrafi/concurve/issues

**Depends** R (>= 4.0.0)

**Imports** methods, bcaboot, boot, dplyr, flextable, ggplot2, knitr, metafor, officer, parallel, pbmcapply, ProfileLikelihood, scales, colorspace, survival, survminer, tibble, tidyr

**Suggests** MASS, lme4, covr, roxygen2, spelling, testthat, rmarkdown, Lock5Data, carData, bench, rms, brms, rstan, rstanarm, bayesplot, vdiffr, ggtext, daewr, svglite, data.table, nlme, simstudy, patchwork, cowplot, reprex

**ByteCompile** true

**Encoding** UTF-8

**Language** en-US
Description

Allows one to compute compatibility (confidence) intervals for various statistical tests along with their corresponding P-values, S-values, and likelihoods. The intervals can be plotted to create consonance, surprisal, and likelihood functions allowing one to see what effect sizes are compatible with the test model at various compatibility levels rather than being limited to one interval estimate such as 95%.

Package: concurve
Details

Accepts most modeling functions that produce confidence intervals to construct distributions.

See the following articles:

- Comparison to Bayesian Posterior Distributions
- The Bootstrap and Consonance Functions
- Background Literature
- Customizing Plots
- Examples in R
- Logistic Regression in R
- Profile Likelihoods
- Meta-Analysis Examples
- Survival Modeling
- S-values
- Generating Tables
- Troubleshooting
- Consonance Functions for Linear Mixed-Effects Models
- Wish List

Author(s)

Zad Rafi, Andrew D. Vigotsky

References


See Also

curve_gen, ggcurve, curve_table

curve_boot

Generate Consonance Functions via Bootstrapping

Description

Use the Bca bootstrap method and the t-bootstrap method from the bcaboot and boot packages to
generate consonance distributions.

Usage

curve_boot(data = data, func = func, method = "bca", t0, tt, bb,
replicates = 2000, steps = 1000, cores = getOption("mc.cores", 1L),
table = TRUE)
curve_compare

Arguments

data
  Dataset that is being used to create a consonance function.

func
  Custom function that is used to create parameters of interest that will be bootstraped.

method
  The bootstrap method that will be used to generate the functions. Methods include "bca" which is the default, "bcapar", which is parametric bootstrapping using the bca method and "t", for the t-bootstrap/percentile method.

t0
  Only used for the "bcapar" method. Observed estimate of theta, usually by maximum likelihood.

tt
  Only used for the "bcapar" method. A vector of parametric bootstrap replications of theta of length B, usually large, say B = 2000

bb
  Only used for the "bcapar" method. A B by p matrix of natural sufficient vectors, where p is the dimension of the exponential family.

replicates
  Indicates how many bootstrap replicates are to be performed. The default is currently 20000 but more may be desirable, especially to make the functions more smooth.

steps
  Indicates how many consonance intervals are to be calculated at various levels. For example, setting this to 100 will produce 100 consonance intervals from 0 to 100. Setting this to 10000 will produce more consonance levels. By default, it is set to 1000. Increasing the number substantially is not recommended as it will take longer to produce all the intervals and store them into a dataframe.

cores
  Select the number of cores to use in order to compute the intervals The default is 1 core.

table
  Indicates whether or not a table output with some relevant statistics should be generated. The default is TRUE and generates a table which is included in the list object.

Value

A list with 7 items where the dataframe of standard values is in the first list and the table for it in the second if table = TRUE. The Bca intervals and table are found in the third and fourth list. The values for the density function are in the fifth object, while the Bca stats are in the sixth and seventh objects.

curve_compare(data1, data2, type = "c", plot = TRUE, ...)

Description

Compares the p-value/s-value, and likelihood functions and computes an AUC number.

Usage

curve_compare(data1, data2, type = "c", plot = TRUE, ...)
Arguments

data1  The first dataframe produced by one of the interval functions in which the intervals are stored.
data2  The second dataframe produced by one of the interval functions in which the intervals are stored.
type   Choose whether to plot a "consonance" function, a "surprisal" function or "likelihood". The default option is set to "c". The type must be set in quotes, for example curve_compare (type = "s") or curve_compare(type = "c"). Other options include "pd" for the consonance distribution function, and "cd" for the consonance density function, "l1" for relative likelihood, "l2" for log-likelihood, "l3" for likelihood and "d" for deviance function.
plot   by default it is set to TRUE and will use the plot_compare() function to plot the two functions.
...   Can be used to pass further arguments to plot_compare().

Value

Computes an AUC score and returns a plot that graphs two functions.

See Also

plot_compare()
ggcurve()
curve_table()

Examples

## Not run:
library(concurve)
GroupA <- rnorm(50)
GroupB <- rnorm(50)
RandomData <- data.frame(GroupA, GroupB)
intervalsdf <- curve_mean(GroupA, GroupB, data = RandomData)
GroupA2 <- rnorm(50)
GroupB2 <- rnorm(50)
RandomData2 <- data.frame(GroupA2, GroupB2)
model <- lm(GroupA2 ~ GroupB2, data = RandomData2)
randomframe <- curve_gen(model, "GroupB2")
curve_compare(intervalsdf[[1]], randomframe[[1]])

## End(Not run)
Consonance Functions for Correlations

Description

Computes consonance intervals to produce P- and S-value functions for correlational analyses using the cor.test function in base R and places the interval limits for each interval level into a data frame along with the corresponding p-values and s-values.

Usage

curve_corr(x, y, alternative, method, steps = 10000, cores = getOption("mc.cores", 1L), table = TRUE)

Arguments

x A vector that contains the data for one of the variables that will be analyzed for correlational analysis.

y A vector that contains the data for one of the variables that will be analyzed for correlational analysis.

alternative Indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less". You can specify just the initial letter. "greater" corresponds to positive association, "less" to negative association.

method A character string indicating which correlation coefficient is to be used for the test. One of "pearson", "kendall", or "spearman", can be abbreviated.

steps Indicates how many consonance intervals are to be calculated at various levels. For example, setting this to 100 will produce 100 consonance intervals from 0 to 100. Setting this to 10000 will produce more consonance levels. By default, it is set to 1000. Increasing the number substantially is not recommended as it will take longer to produce all the intervals and store them into a dataframe.

cores Select the number of cores to use in order to compute the intervals. The default is 1 core.

table Indicates whether or not a table output with some relevant statistics should be generated. The default is TRUE and generates a table which is included in the list object.

Value

A list with 3 items where the dataframe of values is in the first object, the values needed to calculate the density function in the second, and the table for the values in the third if table = TRUE.
Examples

```r
## Not run:
GroupA <- rnorm(50)
GroupB <- rnorm(50)
joe <- curve_corr(x = GroupA, y = GroupB, alternative = "two.sided", method = "pearson")
## End(Not run)
```

curve_gen

Consonance Functions For Linear Models, Generalized Linear Models, and Robust Linear Models

Description

Computes thousands of consonance (confidence) intervals for the chosen parameter in the selected model (linear models, general linear models, robust linear models, and generalized least squares and places the interval limits for each interval level into a data frame along with the corresponding p-values and s-values. Can also adjust for multiple comparisons. It is generally recommended to wrap this function using suppressMessages() due to the long list of profiling messages.

Usage

```r
curve_gen(model, var, method = "lm", log = FALSE, penalty = NULL, 
m = NULL, steps = 1000, cores = getOption("mc.cores", 1L),
  table = TRUE)
```

Arguments

- `model` The statistical model of interest (ANOVA, regression, logistic regression) is to be indicated here.
- `var` The variable of interest from the model (coefficients, intercept) for which the intervals are to be produced.
- `method` Chooses the method to be used to calculate the consonance intervals. There are currently five methods: "lm", rms::ols objects can be used with the "lm" option, "rlm", "glm" and "aov", and "gls". The "lm" method uses the profile likelihood method to compute intervals and can be used for models created by the 'lm' function. It is typically what most people are familiar with when computing intervals based on the calculated standard error. The ols function from the rms package can also be used for this option. The "rlm" method is designed for usage with the "rlm" function from the MASS package. The "glm" method allows this function to be used for specific scenarios like logistic regression and the 'glm' function. Similarly, the Glim function from the rms package can also be used for this option. The gls method allows objects from gls() or from Gls() from the rms package.
Determines whether the coefficients will be exponentiated or not. By default, it is off and set to FALSE or F, but changing this to TRUE or T, will exponentiate the results which may be useful if trying to view the results from a logistic regression on a scale that is not logarithmic.

An input to specify whether the confidence intervals should be corrected for multiple comparisons. The default is NULL, so there is no correction. Other options include "bonferroni" and "sidak".

Indicates how many comparisons are being done and the number that should be used to correct for multiple comparisons. The default is NULL.

Indicates how many consonance intervals are to be calculated at various levels. For example, setting this to 100 will produce 100 consonance intervals from 0 to 100. Setting this to 10000 will produce more consonance levels. By default, it is set to 1000. Increasing the number substantially is not recommended as it will take longer to produce all the intervals and store them into a data frame.

Select the number of cores to use in order to compute the intervals. The default is 1 core.

Indicates whether or not a table output with some relevant statistics should be generated. The default is TRUE and generates a table which is included in the list object.

A list with 3 items where the dataframe of values is in the first object, the values needed to calculate the density function in the second, and the table for the values in the third if table = TRUE.

## Not run:
# Simulate random data
GroupA <- rnorm(50)
GroupB <- rnorm(50)
RandomData <- data.frame(GroupA, GroupB)
rob <- lm(GroupA ~ GroupB, data = RandomData)
bob <- curve_gen(rob, "GroupB")

## End(Not run)

### curve_lik

**Compute Profile Likelihood Functions**

Compute Profile Likelihood Functions

**Usage**

curve_lik(likobject, data, table = TRUE)
curve_lmer

Consonance Functions For Linear & Non-Linear Mixed-Effects Models.

Description
Computes thousands of consonance (confidence) intervals for the chosen parameter in the selected lme4 model and places the interval limits for each interval level into a data frame along with the corresponding p-values and s-values. It is generally recommended to wrap this function using suppressMessages() due to the long list of profiling messages.

Usage

curve_lmer(object, parm, method = "profile", zeta = NULL, nsim = NULL,
FUN = NULL, boot.type = NULL, steps = 1000,
cores = getOption("mc.cores", 1L), table = FALSE)

Arguments

object The statistical model of interest from lme4 is to be indicated here.
parm The variable of interest from the model (coefficients, intercept) for which the intervals are to be produced.
method Chooses the method to be used to calculate the consonance intervals. There are currently four methods: "default", "wald", "lm", and "boot". The "default" method uses the profile likelihood method to compute intervals and can be used for models created by the 'lm' function. The "wald" method is typically what
most people are familiar with when computing intervals based on the calculated standard error. The "lm" method allows this function to be used for specific scenarios like logistic regression and the 'glm' function. The "boot" method allows for bootstrapping at certain levels.

\( zeta \) (for method = "profile" only:) likelihood cutoff (if not specified, as by default, computed from level).

\( nsim \) number of simulations for parametric bootstrap intervals.

\( FUN \) function; if NULL, an internal function that returns the fixed-effect parameters as well as the random-effect parameters on the standard deviation/correlationscale will be used.

\( boot.type \) bootstrap confidence interval type, as described in boot.c i. Methods stud and bca are unavailable because they require additional components to be calculated.

\( steps \) Indicates how many consonance intervals are to be calculated at various levels. For example, setting this to 100 will produce 100 consonance intervals from 0 to 100. Setting this to 10000 will produce more consonance levels. By default, it is set to 1000. Increasing the number substantially is not recommended as it will take longer to produce all the intervals and store them into a dataframe.

\( cores \) Select the number of cores to use in order to compute the intervals The default is 1 core.

\( table \) Indicates whether or not a table output with some relevant statistics should be generated. The default is TRUE and generates a table which is included in the list object.

Value

A list with 3 items where the dataframe of values is in the first object, the values needed to calculate the density function in the second, and the table for the values in the third if table = TRUE.

### curve_mean

Consonance Functions For Mean Differences

Description

Computes thousands of consonance (confidence) intervals for the chosen parameter in a statistical test that compares means and places the interval limits for each interval level into a data frame along with the corresponding p-values and s-values.

Usage

```
curve_mean(x, y, data, paired = F, method = "default", replicates = 1000, steps = 10000, cores = getOption("mc.cores", 1L), table = TRUE)
```
curve_mean

Arguments

x Variable that contains the data for the first group being compared.

y Variable that contains the data for the second group being compared.

data Data frame from which the variables are being extracted from.

paired Indicates whether the statistical test is a paired difference test. By default, it is set to "F", which means the function will be an unpaired statistical test comparing two independent groups. Inserting "paired" will change the test to a paired difference test.

method By default this is turned off (set to "default"), but allows for bootstrapping if "boot" is inserted into the function call.

replicates Indicates how many bootstrap replicates are to be performed. The default is currently 20000 but more may be desirable, especially to make the functions more smooth.

steps Indicates how many consonance intervals are to be calculated at various levels. For example, setting this to 100 will produce 100 consonance intervals from 0 to 100. Setting this to 10000 will produce more consonance levels. By default, it is set to 1000. Increasing the number substantially is not recommended as it will take longer to produce all the intervals and store them into a dataframe.

cores Select the number of cores to use in order to compute the intervals. The default is 1 core.

table Indicates whether or not a table output with some relevant statistics should be generated. The default is TRUE and generates a table which is included in the list object.

Value

A list with 3 items where the dataframe of values is in the first object, the values needed to calculate the density function in the second, and the table for the values in the third if table = TRUE.

Examples

```r
## Not run:
# Simulate random data
GroupA <- runif(100, min = 0, max = 100)
GroupB <- runif(100, min = 0, max = 100)
RandomData <- data.frame(GroupA, GroupB)
bob <- curve_mean(GroupA, GroupB, RandomData)

## End(Not run)
```
**Description**

Computes thousands of consonance (confidence) intervals for the chosen parameter in the meta-analysis done by the metafor package and places the interval limits for each interval level into a data frame along with the corresponding p-values and s-values.

**Usage**

```r
curve_meta(x, measure = "default", method = "uni", parm = NULL, robust = FALSE, cluster = NULL, adjust = FALSE, steps = 1000, cores = getOption("mc.cores", 1L), table = TRUE)
```

**Arguments**

- `x`: Object where the meta-analysis parameters are stored, typically a list produced by `metafor`.
- `measure`: Indicates whether the object has a log transformation or is normal/default. The default setting is "default". If the measure is set to "ratio", it will take logarithmically transformed values and convert them back to normal values in the dataframe. This is typically a setting used for binary outcomes such as risk ratios, hazard ratios, and odds ratios.
- `method`: Indicates which meta-analysis metafor function is being used. Currently supports `rma.uni` ("uni"), which is the default, `rma.mh` ("mh"), and `rma.peto` ("peto").
- `parm`: Typically ignored, but needed sometimes in order to specify which variable to produce function for.
- `robust`: a logical indicating whether to produce cluster robust interval estimates. Default is `FALSE`.
- `cluster`: a vector specifying a clustering variable to use for constructing the sandwich estimator of the variance-covariance matrix. Default setting is `NULL`.
- `adjust`: logical indicating whether a small-sample correction should be applied to the variance-covariance matrix. Default is `FALSE`.
- `steps`: Indicates how many consonance intervals are to be calculated at various levels. For example, setting this to 100 will produce 100 consonance intervals from 0 to 100. Setting this to 10000 will produce more consonance levels. By default, it is set to 1000. Increasing the number substantially is not recommended as it will take longer to produce all the intervals and store them into a dataframe.
- `cores`: Select the number of cores to use in order to compute the intervals. The default is 1 core.
- `table`: Indicates whether or not a table output with some relevant statistics should be generated. The default is `TRUE` and generates a table which is included in the list object.
Value

A list with 3 items where the dataframe of values is in the first object, the values needed to calculate the density function in the second, and the table for the values in the third if table = TRUE.

Examples

```r
## Not run:
# Simulate random data for two groups in two studies
GroupAData <- runif(20, min = 0, max = 100)
GroupAMean <- round(mean(GroupAData), digits = 2)
GroupASD <- round(sd(GroupAData), digits = 2)

GroupBData <- runif(20, min = 0, max = 100)
GroupBMean <- round(mean(GroupBData), digits = 2)
GroupBSD <- round(sd(GroupBData), digits = 2)

GroupCData <- runif(20, min = 0, max = 100)
GroupCMean <- round(mean(GroupCData), digits = 2)
GroupCSD <- round(sd(GroupCData), digits = 2)

GroupDData <- runif(20, min = 0, max = 100)
GroupDMean <- round(mean(GroupDData), digits = 2)
GroupDSD <- round(sd(GroupDData), digits = 2)

# Combine the data

StudyName <- c("Study1", "Study2")
MeanTreatment <- c(GroupAMean, GroupCMean)
MeanControl <- c(GroupBMean, GroupDMean)
SDTreatment <- c(GroupASD, GroupCSD)
SDControl <- c(GroupBSD, GroupDSD)
NTreatment <- c(20, 20)
NControl <- c(20, 20)

metadf <- data.frame(
  StudyName, MeanTreatment, MeanControl,
  SDTreatment, SDControl, NTreatment, NControl)

# Use metafor to calculate the standardized mean difference

library(metafor)

dat <- escalc(
  measure = "SMD", m1i = MeanTreatment, sd1i = SDTreatment,
  n1i = NTreatment, m2i = MeanControl, sd2i = SDControl,
  n2i = NControl, data = metadf)

# Pool the data using a particular method. Here "FE" is the fixed-effects model

res <- rma(yi, vi,
```

Reverse Engineer Consonance / Likelihood Functions Using the Point Estimate and Confidence Limits

Using the confidence limits and point estimates from a dataset, one can use these estimates to compute thousands of consonance intervals and graph the intervals to form a consonance, surprisal, and likelihood functions. The intervals are calculated from the approximated normal distribution, however, users should be cautious as this this function is currently designed for similar situations (involving ratios and normal approximations), nevertheless the function also works for means but should be used skeptically, as it can break down in many situations and give implausible numbers. Computations of likelihood functions for means is currently not supported.

Usage

```r
curve_rev(point, LL = NULL, UL = NULL, se = NULL, conf.level = 0.95, type = "c", measure = "ratio", steps = 10000, cores = getOption("mc.cores", 1L), table = TRUE)
```

Arguments

- **point**: The point estimate from an analysis. Ex: 1.20
- **LL**: The lower confidence limit from an analysis Ex: 1.0
- **UL**: The upper confidence limit from an analysis Ex: 1.4
- **se**: The standard error of the point estimate. Ex: 0.05
- **conf.level**: Indicates whether the produced result should be a consonance function or a likelihood function. The default is "c" for consonance and likelihood can be set via "l".
- **type**: The type of data being used. If they involve mean differences, then the "mean" option should be used. If the data are ratios, then the "ratio" option should be used. "ratio" is currently the default option. Currently, this function is designed to be used with ratios and normal approximations rather than means.
curve_surv

Consonance Functions For Survival Data

Description

Computes thousands of consonance (confidence) intervals for the chosen parameter in the Cox model computed by the 'survival' package and places the interval limits for each interval level into a data frame along with the corresponding p-value and s-value.

Usage

curve_surv(data, x, steps = 10000, cores = getOption("mc.cores", 1L),
            table = TRUE)
 Arguments

data  Object where the Cox model is stored, typically a list produced by the 'survival' package.
x  Predictor of interest within the survival model for which the consonance intervals should be computed.
steps  Indicates how many consonance intervals are to be calculated at various levels. For example, setting this to 100 will produce 100 consonance intervals from 0 to 100. Setting this to 10000 will produce more consonance levels. By default, it is set to 1000. Increasing the number substantially is not recommended as it will take longer to produce all the intervals and store them into a dataframe.
cores  Select the number of cores to use in order to compute the intervals. The default is 1 core.
table  Indicates whether or not a table output with some relevant statistics should be generated. The default is TRUE and generates a table which is included in the list object.

Value

A list with 3 items where the dataframe of values is in the first object, the values needed to calculate the density function in the second, and the table for the values in the third if table = TRUE.

Examples

```r
## Not run:
library(carData)
Rossi[1:5, 1:10]
library(survival)
mod.allison <- coxph(Surv(week, arrest) ~ fin + age + race + wexp + mar + paro + prio, 
data = Rossi
)
mod.allison
z <- curve_surv(mod.allison, "prio")
## End(Not run)
```

curve_table  Produce Tables For concurve Functions

 Description

Produces publication-ready tables with relevant statistics of interest for functions produced from the concurve package.

Usage

curve_table(data, levels, type = "c", format = "data.frame")
Arguments

data  Dataframe from a concurve function to produce a table for
levels  Levels of the consonance intervals or likelihood intervals that should be included in the table.
type  Indicates whether the table is for a consonance function or likelihood function. The default is set to "c" for consonance and can be switched to "l" for likelihood.
format  The format of the tables. The options include "data.frame" which is the default, "docx" (which creates a table for a word document), "pptx" (which creates a table for powerpoint), "latex", (which creates a table for a TeX document), and "image", which produces an image of the table.

See Also

ggcurve()
curve_compare()
plot_compare()

Examples

## Not run:
library(concurve)

GroupA <- rnorm(500)
GroupB <- rnorm(500)

RandomData <- data.frame(GroupA, GroupB)

intervalsdf <- curve_mean(GroupA, GroupB, data = RandomData, method = "default")

(z <- curve_table(intervalsdf[[1]], format = "data.frame"))
(z <- curve_table(intervalsdf[[1]], format = "latex"))
(z <- curve_table(intervalsdf[[1]], format = "image"))

## End(Not run)

---

**Plots Consonance, Surprisal, and Likelihood Functions**

Description

Takes the dataframe produced by the interval functions and plots the p-values/s-values, consonance (confidence) levels, and the interval estimates to produce a p-value/s-value function using ggplot2 graphics.
### ggcurve

**Usage**

```r
ggcurve(data, type = "c", measure = "default", levels = 0.95, nullvalue = NULL, position = "pyramid", title = "Consonance Function", subtitle = "The function displays intervals at every level.", xaxis = expression(theta == ~"Range of Values"), yaxis1 = expression(paste(italic(p), "-value")), yaxis2 = "Levels for CI (%)", color = darken("#009E73", 0.5), fill = "#239a98")
```

**Arguments**

- `data` The dataframe produced by one of the interval functions in which the intervals are stored.
- `type` Choose whether to plot a "consonance" function, a "surprisal" function or "likelihood". The default option is set to "c". The type must be set in quotes, for example `ggcurve(type = "s")` or `ggcurve(type = "c")`. Other options include "pd" for the consonance distribution function, and "cd" for the consonance density function, "l1" for relative likelihood, "l2" for log-likelihood, "l3" for likelihood and "d" for deviance function.
- `measure` Indicates whether the object has a log transformation or is normal/default. The default setting is "default". If the measure is set to "ratio", it will take logarithmically transformed values and convert them back to normal values in the dataframe. This is typically a setting used for binary outcomes and their measures such as risk ratios, hazard ratios, and odds ratios.
- `levels` Indicates which interval levels should be plotted on the function. By default it is set to 0.95 to plot the 95% interval on the consonance function, but more levels can be plotted by using the `c()` function for example, `levels = c(0.50, 0.75, 0.95)`.
- `nullvalue` Indicates whether the null value for the measure should be plotted. By default, it is set to NULL, meaning it will not be plotted as a vertical line. Changing this to a numerical vector will specify the region where a line should be plotted or an area that should be shaded. The input must be a numerical vector, for example `c(-0.5, 0.5)` or a single numerical vector such as 0 or 1.
- `position` Determines the orientation of the P-value (consonance) function. By default, it is set to "pyramid", meaning the p-value function will stand right side up, like a pyramid. However, it can also be inverted via the option "inverted". This will also change the sequence of the y-axes to match the orientation. This can be set as such, `ggcurve(type = "c", data = df, position = "inverted")`.
- `title` A custom title for the graph. By default, it is set to "Consonance Function". In order to set a title, it must be in quotes. For example, `ggcurve(type = "c", data = x, title = "Custom Title")`.
- `subtitle` A custom subtitle for the graph. By default, it is set to "The function contains consonance/confidence intervals at every level and the P-values." In order to set a subtitle, it must be in quotes. For example, `ggcurve(type = "c", data = x, subtitle = "Custom Subtitle")`.
- `xaxis` A custom x-axis title for the graph. By default, it is set to "Range of Values. In order to set a x-axis title, it must be in quotes. For example, `ggcurve(type = "c", data = x, xaxis = "Hazard Ratio")".```
plot_compare

**yaxis1**
A custom y-axis title for the graph. By default, it is set to "Consonance Level". In order to set a y-axis title, it must be in quotes. For example, `ggcurve(type = "c", data = x, yaxis1= "Confidence Level")`.

**yaxis2**
A custom y-axis title for the graph. By default, it is set to "Levels for CI". In order to set a y-axis title, it must be in quotes. For example, `ggcurve(type = "c", data = x, yaxis2= "Confidence Level")`.

**color**
Item that allows the user to choose the color of the points and the ribbons in the graph. By default, it is set to color = "#555555". The inputs must be in quotes. For example, `ggcurve(type = "c", data = x, color = "#333333")`.

**fill**
Item that allows the user to choose the color of the ribbons in the graph. By default, it is set to fill = "#239a98". The inputs must be in quotes. For example, `ggcurve(type = "c", data = x, fill = "#333333")`.

**Value**
A plot with intervals at every consonance level graphed with their corresponding p-values and compatibility levels.

**See Also**
`plot_compare()`

**Examples**

```r
# Not run:
# Simulate random data

library(concurve)

GroupA <- rnorm(500)
GroupB <- rnorm(500)
RandomData <- data.frame(GroupA, GroupB)

intervalsdf <- suppressMessages(curve_mean(GroupA, GroupB, data = RandomData, method = "default"))
ggcurve(type = "c", intervalsdf[[1]], nullvalue = c(0))
```

---

**plot_compare**

*Graph and Compare Consonance, Surprisal, and Likelihood Functions*

**Description**

Compares the p-value/s-value, and likelihood functions using ggplot2 graphics.
plot_compare

Usage

plot_compare(data1, data2, type = "c", measure = "default", 
nullvalue = FALSE, position = "pyramid", title = "Interval Functions", 
subtitle = "The function displays intervals at every level.", 
xaxis = expression(theta == ~"Range of Values"), 
yaxis1 = expression(paste(italic(p), "-value")), 
yaxis2 = "Levels for CI (%)", color1 = darken("#D55E00", 0.2), 
color2 = darken("#009E73", 0.2), fill1 = "#99c7c7", fill2 = "#d46c5b")

Arguments

data1 The first dataframe produced by one of the interval functions in which the intervals are stored.
data2 The second dataframe produced by one of the interval functions in which the intervals are stored.
type Choose whether to plot a "consonance" function, a "surprisal" function or "likelihood". The default option is set to "c". The type must be set in quotes, for example plot_compare(type = "s") or plot_compare(type = "c"). Other options include "pd" for the consonance distribution function, and "cd" for the consonance density function, "l1" for relative likelihood, "l2" for log-likelihood, "l3" for likelihood and "d" for deviance function.
measure Indicates whether the object has a log transformation or is normal/default. The default setting is "default". If the measure is set to "ratio", it will take logarithmically transformed values and convert them back to normal values in the dataframe. This is typically a setting used for binary outcomes and their measures such as risk ratios, hazard ratios, and odds ratios.
nullvalue Indicates whether the null value for the measure should be plotted. By default, it is set to FALSE, meaning it will not be plotted as a vertical line. Changing this to TRUE, will plot a vertical line at 0 when the measure is set to "default" and a vertical line at 1 when the measure is set to "ratio". For example, plot_compare(type = "c", data = df, measure = "ratio", nullvalue = "present"). This feature is not yet available for surprisal functions.
position Determines the orientation of the P-value (consonance) function. By default, it is set to "pyramid", meaning the p-value function will stand right side up, like a pyramid. However, it can also be inverted via the option "inverted". This will also change the sequence of the y-axes to match the orientation. This can be set as such, plot_compare(type = "c", data = df, position = "inverted").
title A custom title for the graph. By default, it is set to "Consonance Function". In order to set a title, it must be in quotes. For example, plot_compare(type = "c", data = x, title = "Custom Title").
subtitle A custom subtitle for the graph. By default, it is set to "The function contains consonance/confidence intervals at every level and the P-values." In order to set a subtitle, it must be in quotes. For example, plot_compare(type = "c", data = x, subtitle = "Custom Subtitle").
xaxis A custom x-axis title for the graph. By default, it is set to "Range of Values. In order to set a x-axis title, it must be in quotes. For example, plot_compare(type = "c", data = x, xaxis = "Hazard Ratio").
plot_compare

yaxis1  A custom y-axis title for the graph. By default, it is set to "Consonance Level". In order to set a y-axis title, it must be in quotes. For example, ggcurve(type = "c", data = x, yaxis1 = "Confidence Level").

yaxis2  A custom y-axis title for the graph. By default, it is set to "Levels for CI". In order to set a y-axis title, it must be in quotes. For example, ggcurve(type = "c", data = x, yaxis2 = "Confidence Level").

color1  Item that allows the user to choose the color of the points and the ribbons in the graph. By default, it is set to darken("#D55E00", 0.4). The inputs must be in quotes.

color2  Item that allows the user to choose the color of the points and the ribbons in the graph. By default, it is set to darken("#009E73", 0.4). The inputs must be in quotes. For example, plot_compare(type = "c", data = x, color = ":"#333333".

fill1  Item that allows the user to choose the color of the ribbons in the graph for data1. By default, it is set to fill1 = ":#239a98". The inputs must be in quotes. For example, plot_compare(type = "c", data = x, fill1 = ":#333333").

fill2  Item that allows the user to choose the color of the ribbons in the graph for data1. By default, it is set to fill2 = ":#d46c5b". The inputs must be in quotes. For example, plot_compare(type = "c", data = x, fill2 = ":#333333").

Value

A plot that compares two functions.

See Also

ggcurve()
curve_compare()

Examples

## Not run:
library(concurve)

GroupA <- rnorm(50)
GroupB <- rnorm(50)
RandomData <- data.frame(GroupA, GroupB)
intervalsdf <- curve_mean(GroupA, GroupB, data = RandomData)
GroupA2 <- rnorm(50)
GroupB2 <- rnorm(50)
RandomData2 <- data.frame(GroupA2, GroupB2)
model <- lm(GroupA2 ~ GroupB2, data = RandomData2)
randomframe <- curve_gen(model, "GroupB2")

plot_compare(intervalsdf[[1]], randomframe[[1]], type = "c")

## End(Not run)
RobustMax

Robust Max, an alternative to max() that doesn’t throw a warning

Description
Robust Max, an alternative to max() that doesn’t throw a warning

Usage
RobustMax(x)

Arguments
x A vector to find the maximum value of

Value
The max value from a vector

RobustMin

Robust Min, an alternative to max() that doesn’t throw a warning

Description
Robust Min, an alternative to max() that doesn’t throw a warning

Usage
RobustMin(x)

Arguments
x A vector find the minimum value of

Value
The minimum value from the vector
Index

concurve (concurve-package), 2
concurve-package, 2
curve_boot, 4
curve_compare, 5
curve_compare(), 16, 18, 22
curve_corr, 7
curve_gen, 4, 8
curve_lik, 9
curve_lmer, 10
curve_mean, 11
curve_meta, 13
curve_rev, 15
curve Surv, 16
curve_table, 4, 17
curve_table(), 6
ggcurve, 4, 18
ggcurve(), 6, 16, 18, 22
plot_compare, 20
plot_compare(), 6, 16, 18, 20
RobustMax, 23
RobustMin, 23