Package 'concurve'

December 4, 2019

Type Package

Title Computes and Plots Compatibility (Confidence) Intervals, P-Values, S-Values, & Likelihood Intervals to Form Consonance, Surprisal, & Likelihood Functions

Version 2.3.0

Date 2019-12-04

Maintainer Zad R. Chow <zad@lesslikely.com>

Description Allows one to compute consonance (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 consonance levels rather than being limited to one interval estimate such as 95%. These methods are discussed by Poole C. (1987) <doi:10.2105/AJPH.77.2.195>, Schweder T, Hjort NL. (2002) <doi:10.1111/1467-9469.00285>, Singh K, Xie M, Strawderman WE. (2007) <arXiv:0708.0976>, Rothman KJ, Greenland S, Lash TL. (2008, ISBN:9781451190052), Amrhein V, Trafimow D, Greenland S. (2019) <doi:10.1080/00031305.2018.1543137>, Greenland S. (2019) <doi:10.1080/00031305.2018.1529625>, Chow ZR, Greenland S. (2019) <arXiv:1909.08579>, and Greenland S, Chow ZR. (2019) <arXiv:1909.08583>.

License GPL-3 | file LICENSE

URL https://data.lesslikely.com/concurve/, https://github.com/zadchow/concurve, https://lesslikely.com/

BugReports https://github.com/zadchow/concurve/issues

Imports bcaboot, boot, compiler, dplyr, flextable, ggplot2, knitr, metafor, officer, parallel, pbmcapply, ProfileLikelihood, rlang (>= 0.1.2), scales, survival, survminer, tibble, tidyr, MASS, methods

Suggests covr, roxygen2, spelling, testthat, rmarkdown, Lock5Data

VignetteBuilder knitr

ByteCompile true

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.0.2

X-schema.org-keywords confidence, compatibility, consonance, surprisal, interval, function, curve

Depends R (>= 3.2)

NeedsCompilation no

Author Zad R. Chow [aut, cre] (<https://orcid.org/0000-0003-1545-8199>), Andrew D. Vigotsky [aut] (<https://orcid.org/0000-0003-3166-0688>)

Repository CRAN

Date/Publication 2019-12-04 11:20:03 UTC

R topics documented:

curve_boot 2
curve_compare 4
curve_corr 5
curve_gen 6
curve_lik 7
curve_mean 7
curve_meta 9
curve_rev 10
curve_surv 11
curve_table 12
ggcurve 13
plot_compare 15

Index **18**

`curve_boot` *Generate Consonance Functions via Bootstrapping*

Description

Use the BCa bootstrap method and the t boostrap method from the bcaboot and boot packages to generate consonance distrbutions.

Usage

```
curve_boot(
  data = data,
  func = func,
  method = "bca",
  replicates = 2000,
  steps = 1000,
  table = TRUE
)
```

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 bootstrapped.
method	The boostrap method that will be used to generate the functions. Methods include "bca" which is the default and "t".
replicates	Indicates how many bootstrap replicates are to be performed. The defaultis 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.
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 the dataframe of values in the first list and the table in the second if table = TRUE.

Examples

```
data(diabetes, package = "bcaboot")
Xy <- cbind(diabetes$x, diabetes$y)
rfun <- function(Xy) {
  y <- Xy[, 11]
  X <- Xy[, 1:10]
  return(summary(lm(y ~ X))$adj.r.squared)
}

x <- curve_boot(data = Xy, func = rfun, method = "bca", replicates = 200, steps = 1000)

ggcurve(data = x[[1]])
```

```
ggcurve(data = x[[3]])

plot_compare(x[[1]], x[[3]])
```

---

curve_compare *Compares two functions and produces an AUC score to show the amount of consonance.*

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().

Examples

```
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]]))
(curve_compare(intervalsdf[[1]], randomframe[[1]], type = "s"))
```

---

curve_corr *Computes Consonance Intervals for Correlations*

Description

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

Usage

```
curve_corr(x, y, alternative, method, steps = 10000, 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.
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.

Examples

```
GroupA <- rnorm(50)
GroupB <- rnorm(50)
joe <- curve_corr(x = GroupA, y = GroupB, alternative = "two.sided", method = "pearson")
tibble::tibble(joe[[1]])
```

curve_gen *General Consonance Functions Using Profile Likelihood, Wald, or the bootstrap method for linear models.*

Description

Computes thousands of consonance (confidence) intervals for the chosen parameter in the selected model (ANOVA, ANCOVA, regression, logistic regression) and places the interval limits for each interval level into a data frame along with the corresponding p-values and s-values.

Usage

```
curve_gen(model, var, method = "wald", steps = 1000, 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 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 typicallywhat 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.
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.
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.

Examples

```
# 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")
tibble::tibble(bob[[1]])
```

---

curve_lik *Compute the Profile Likelihood Functions*

Description

Compute the Profile Likelihood Functions

Usage

```
curve_lik(likobject, data, table = TRUE)
```

Arguments

likobject	An object from the ProfileLikelihood package
data	The dataframe that was used to create the likelihood object in the ProfileLikelihood package.
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.

Examples

```
library(ProfileLikelihood)
data(dataglm)
xx <- profilelike.glm(y ~ x1 + x2, dataglm, profile.theta = "group", binomial("logit"))
lik <- curve_lik(xx, dataglm)
tibble::tibble(lik[[1]])
```

---

curve_mean *Mean Interval Consonance Function*

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,
  table = TRUE
)
```

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 insertedinto the function call.
replicates	Indicates how many bootstrap replicates are to be performed. The defaultis 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.
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.

Examples

```
# 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)
tibble::tibble(bob[[1]])
```

curve_meta *Meta-analytic Consonance Function*

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

```
curve_meta(x, measure = "default", steps = 10000, 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.
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.
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.

Examples

```
# 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,
  data = dat, slab = paste(StudyName, sep = ", "),
  method = "FE", digits = 2
)

# Calculate the intervals using the metainterval function

metaf <- curve_meta(res)

tibble::tibble(metaf[[1]])
```

---

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

Description

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 and surprisal function.

Usage

```
curve_rev(
  point,
  LL,
  UL,
  type = "c",
  measure = "default",
  steps = 10000,
  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
type	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".
measure	The type of data being used. If they involve mean differences,
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.
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.

Examples

```
# From a real published study. Point estimate of the result was hazard ratio of 1.61 and
# lower bound of the interval is 0.997 while upper bound of the interval is 2.59.
#
df <- curve_rev(point = 1.61, LL = 0.997, UL = 2.59, measure = "ratio")

tibble::tibble(df[[1]])
```

---

curve_surv *Produce Consonance Intervals 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, 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.
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.

---

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, "tibble", "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.

Examples

```
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 = "tibble"))
(z <- curve_table(intervalsdf[[1]], format = "latex"))
```

---

ggcurve *Plots the P-Value (Consonance), S-value (Surprisal), and Likelihood Function via ggplot2*

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.

Usage

```
ggcurve(
  data,
  type = "c",
  measure = "default",
  levels = 0.95,
  nullvalue = FALSE,
  position = "pyramid",
  title = "Interval Function",
  subtitle = "The function displays intervals at every level.",
  xaxis = expression(Theta ~ "Range of Values"),
  yaxis = "P-value",
  color = "#000000",
  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 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, ggcurve(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, 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").

`yaxis` 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, yxis = "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

Plot with intervals at every consonance level graphed with their corresponding p-values and compatibility levels.

Examples

```
# Simulate random data

library(concurve)

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

RandomData <- data.frame(GroupA, GroupB)

intervalsdf <- curve_mean(GroupA, GroupB, data = RandomData, method = "default")
(function1 <- ggcurve(type = "c", intervalsdf[[1]]))
```

---

plot_compare *Compares the P-Value (Consonance), S-value (Surprisal), and Likelihood Function via ggplot2*

Description

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

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"),
  yaxis = "P-value",
  color = "#000000",
  fill1 = "#239a98",
  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").
yaxis	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, plot_compare(type = "c", data = x, yxis = "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, 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.

Examples

```
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 = "s"))
```

Index

curve_boot, 2
curve_compare, 4
curve_corr, 5
curve_gen, 6
curve_lik, 7
curve_mean, 7
curve_meta, 9
curve_rev, 10
curve_surv, 11
curve_table, 12

ggcurve, 13

plot_compare, 15