Package 'scdtb'

July 23, 2025

Title Single Case Design Tools

Version 0.2.0

Description In some situations where researchers would like to demonstrate causal effects, it is hard to obtain a sample size that would allow for a well-powered randomized controlled trial. Single case designs are experimental designs that can be used to demonstrate causal effects with only one participant or with only a few participants. The 'scdtb' package provides a suite of tools for analyzing data from studies that use single case designs. The nap() function can be used to compute the nonoverlap of all pairs as outlined by the What Works Clearinghouse (2022) <https://ies.ed.gov/ncee/wwc/Handbooks>. The package also offers the mixed_model_analysis() and cross_lagged() functions which implement mixed effects models and cross lagged analyses as described in Maric & van der Werff (2020) <doi:10.4324/9780429273872-9>. The randomization_test() function implements randomization tests based on methods presented in Onghena (2020) <doi:10.4324/9780429273872-8>. The scdtb() 'shiny' application can be used to upload single case design data and access various 'scdtb' tools for plotting and analysis.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

URL https://github.com/mightymetrika/scdtb

BugReports https://github.com/mightymetrika/scdtb/issues

Depends R (>= 2.10)

LazyData true

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Imports broom.mixed, DT, ggplot2, MASS, mmcards, mmints, nlme, shiny, shinythemes, sn

NeedsCompilation no

Author Mackson Ncube [aut, cre], mightymetrika, LLC [cph, fnd]

Maintainer Mackson Ncube <macksonncube.stats@gmail.com>

Repository CRAN

Date/Publication 2024-09-20 16:30:02 UTC

Contents

`basic_scd` *Basic Single Case Design*

Description

This is the data for the Basic Single Case Design Example presented in Figure 14 of the What Works Clearinghouse Procedures and Standards Handbook, Version 5.0

Usage

```
basic_scd
```

Format

`basic_scd`**:**
A data frame with 21 rows and 3 columns:

phase Study phase

time Time of data collection

socbehavs Social behaviors score

Source

https://ies.ed.gov/ncee/WWC/Docs/referenceresources/Final_WWC-HandbookVer5_0-0-508.pdf

---

`cross_lagged` *Cross-Lagged Correlation*

Description

Computes cross-lagged correlations between two variables in a dataframe.

Usage

```
cross_lagged(
  .df,
  .x,
  .y,
  lag.max = 5,
  na.action = stats::na.fail,
  conf.level = 0.95,
  ...
)
```

Arguments

`.df`	A dataframe containing the variables for analysis.
`.x`	The name of the first variable (as a string) to be analyzed.
`.y`	The name of the second variable (as a string) to be analyzed.
`lag.max`	The maximum lag at which to calculate the cross-correlation or covariance. Defaults to 5.
`na.action`	A function to specify the action to be taken if NAs are found. Defaults to `stats::na.fail`.
`conf.level`	The confidence level for the confidence intervals. Defaults to 0.95.
`...`	Additional arguments to be passed to stats::ccf().

Details

This function calculates the cross-lagged correlation between two variables in a given dataframe up to a specified maximum lag. It returns an object containing the cross-correlation function, confidence intervals, and other related information. The function calls stats::ccf() internally.

Value

An object of class "cross_lagged" containing the cross-correlation function, upper and lower confidence intervals, the number of observations used, and the names of the variables analyzed.

Examples

```
# Creating a sample dataset
reversal_withdrawal <- data.frame(
 phase = c(rep("baseline1", 6), rep("treatment1", 5), rep("baseline2", 5), rep("treatment2", 5)),
  time = 1:21,
  extbehavs = c(15, 10, 14, 17, 13, 12, 2, 1, 1, 0, 0, 9, 9, 11, 15, 20, 1, 0, 4, 0, 1)
)

reversal_withdrawal$synth <- sapply(reversal_withdrawal$time, function(x) {
  stats::rpois(1, x)
})

reversal_withdrawal <- as.data.frame(reversal_withdrawal)

# Using the cross_lagged function
cl_result <- cross_lagged(reversal_withdrawal, .x = "time", .y = "synth")
```

`efficacy_of_CBT` *Fictional Single Case Design Efficacy of CBT Example*

Description

This is the data set used for the clinical case example in Maric & van der Werf (2020).

Usage

```
efficacy_of_CBT
```

Format

`efficacy_of_CBT`**:**
A data frame with 10 rows and 4 columns:

phase Study phase
time Time of data collection
Anxious Outcome measure
CATS_N Outcome measure

Source

doi:10.4324/9780429273872-9

References

Maric, M., & van der Werff, V. (2020). Single-Case Experimental Designs in Clinical Intervention Research. In R. van de Schoot & M. Miocevic (Eds.), Small Sample Size Solutions: A Guide for Applied Researchers and Practitioners (1st ed., pp. 10). Routledge. doi:10.4324/9780429273872-9

`mixed_model_analysis` *Mixed Model Analysis*

Description

Performs a mixed model analysis on a dataset, allowing for the specification of a dependent variable, time variable, phase variable, participant identification, and covariates. It supports reverse timing within phases, custom phase levels and labels, and adds covariates to the fixed effects model. The function fits a model using generalized least squares and returns a list containing the modified dataset, the fitted model, and a plot of predicted values with phase annotations.

Usage

```
mixed_model_analysis(
  .df,
  .dv,
  .time,
  .phase,
  .participant = NULL,
  rev_time_in_phase = FALSE,
  phase_levels = NULL,
  phase_labels = NULL,
  covs = NULL,
  ...
)
```

Arguments

`.df`	A data frame containing the dataset to be analyzed.
`.dv`	The name of the dependent variable in the dataset.
`.time`	The name of the time variable in the dataset.
`.phase`	The name of the phase variable in the dataset.
`.participant`	(optional) The name of the participant identifier variable in the dataset. If not provided, a default factor will be used.
`rev_time_in_phase`	(optional) A boolean indicating whether to reverse the timing within each phase. Defaults to FALSE.
`phase_levels`	(optional) A vector of phase levels to be used for the phase variable. If NULL, the unique values in the phase variable will be used.
`phase_labels`	(optional) A vector of labels corresponding to the phase_levels. If NULL, phase_levels will be used as labels.
`covs`	(optional) A vector of names of covariates to be added to the fixed effect model.
`...`	Additional arguments passed to the `gls` function.

Value

A list containing three elements: - `data`: The modified dataset with added time variables and predicted values. - `fitted_mod`: The fitted model object from `nlme::gls`. - `plot`: A ggplot object showing the predicted values and phase annotations.

References

Maric, M., & van der Werff, V. (2020). Single-Case Experimental Designs in Clinical Intervention Research. In R. van de Schoot & M. Miocevic (Eds.), Small Sample Size Solutions: A Guide for Applied Researchers and Practitioners (1st ed., pp. 10). Routledge. doi:10.4324/9780429273872-9

Examples

```
res <- mixed_model_analysis(efficacy_of_CBT, .dv = "Anxious", .time = "time",
                            .phase = "phase",rev_time_in_phase = TRUE,
                            phase_levels = c(0, 1),
                            phase_labels = c("Exposure", "Exposure + CT"))

summary(res$fitted_mod)
```

---

nap *Non-overlap of All Pairs (NAP) Analysis*

Description

This function performs a Non-overlap of All Pairs (NAP) analysis on a given data frame, considering specified phases, improvement direction, and analysis type (reversability or trend). It is designed to assess the distinctiveness of data across phases or trends within the data, based on the concept outlined in the What Works Clearinghouse Procedures and Standards Handbook.

Usage

```
nap(
  .df,
  .y,
  .phase,
  .time,
  type = c("reversability", "trend"),
  last_m = NULL,
  phases,
  improvement = c("positive", "negative")
)
```

Arguments

`.df`	A data frame containing the data to be analyzed.
`.y`	Character string specifying the variable in `.df` to be analyzed.
`.phase`	Character string specifying the variable in `.df` that defines the phases.
`.time`	Character string specifying the time variable in `.df`.
`type`	Character string indicating the type of analysis to be conducted: either "reversability" or "trend".
`last_m`	An integer specifying the number of measurements from the end to be considered in a trend analysis. Leave as NULL if `type` is set to "reversability".
`phases`	Vector specifying the phases to be included in the analysis. If `type` is "reversability", two phases must be specified. If `type` is "trend", one phase must be specified.
`improvement`	Character vector indicating the direction of improvement to consider: either "positive" or "negative".

Details

The NAP analysis is a method used to evaluate the effectiveness of interventions by analyzing the non-overlap between data points across different phases or trends within a dataset. It is a useful statistical tool for educational research and is detailed in the What Works Clearinghouse Procedures and Standards Handbook, Version 5.0.

Value

A numeric value representing the NAP score, reflecting the proportion of non-overlapping data points between the specified phases or trends.

References

What Works Clearinghouse. (2022). What Works Clearinghouse procedures and standards handbook, version 5.0. U.S. Department of Education, Institute of Education Sciences, National Center for Education Evaluation and Regional Assistance (NCEE). This report is available on the What Works Clearinghouse website at https://ies.ed.gov/ncee/wwc/Handbooks.

Examples

```
nap(.df = reversal_withdrawal, .y = "extbehavs", .phase = "phase",
    .time = "time", type = "reversability",
    phases = list("baseline1", "baseline2"), improvement = "negative")
```

napjack *Nap Jack: A Single Case Design Card Game*

Description

This function creates a Shiny application that implements the Nap Jack card game. Nap Jack is a single case design card game where the player deals cards in phases and tries to achieve a winning score based on the analysis of the dealt cards.

Usage

```
napjack()
```

Details

The game consists of four phases: baseline 1, treatment 1, baseline 2, and treatment 2. In each phase, the player deals six cards and has the option to swap cards within a row once per phase. After all four phases are completed, the game is scored based on the analysis of the dealt cards using non-overlap of all pairs (NAP) and mixed effects modeling.

The game utilizes the following internal helper functions:

- `deal_phase()`: Deals a phase of cards from the game deck.
- `render_card_grid()`: Renders a grid of card images based on the dealt cards.
- `swapper()`: Allows swapping of cards within a row of the card matrix.

The game also uses the following external functions for analysis:

- `raw_plot()`: Plots the raw data of the dealt cards.
- `nap()`: Performs non-overlap of all pairs analysis.
- `mixed_model_analysis()`: Performs mixed effects modeling analysis.

The player's objective is to achieve a winning score by strategically dealing and swapping cards to optimize the analysis results.

Value

A Shiny application object that represents the Nap Jack game.

Examples

```
# To run the Shiny app
if(interactive()){
  napjack()
}
```

plot.cross_lagged *Plot Cross-Lagged Correlation Results*

Description

Plots the cross-lagged correlation results calculated by cross_lagged(). This method generates a bar plot showing the autocorrelation function (ACF) values for different lags, along with dashed lines indicating the upper and lower confidence intervals. The plot is created using ggplot2.

Usage

```
## S3 method for class 'cross_lagged'
plot(x, ...)
```

Arguments

x	An object of class "cross_lagged" containing the results of a cross-lagged correlation analysis performed by cross_lagged().
...	Additional parameters (currently ignored).

Value

A ggplot object representing the cross-lagged correlation analysis results. This plot includes bars for ACF values at different lags and dashed lines for the upper and lower confidence intervals.

Examples

```
#Creating a sample dataset
reversal_withdrawal <- data.frame(
 phase = c(rep("baseline1", 6), rep("teratment1", 5), rep("baseline2", 5), rep("teratment2", 5)),
  time = 1:21,
  extbehavs = c(15, 10, 14, 17, 13, 12, 2, 1, 1, 0, 0, 9, 9, 11, 15, 20, 1, 0, 4, 0, 1)
)

reversal_withdrawal$synth <- sapply(reversal_withdrawal$time, function(x) {
  stats::rpois(1, x)
})

reversal_withdrawal <- as.data.frame(reversal_withdrawal)

# Using the cross_lagged function
cl_result <- cross_lagged(reversal_withdrawal, .x = "time", .y = "synth")

# Plot the cross-lagged correlation results
plot(cl_result)
```

plot.replext_gls *Plot Results from Replications and Extension of GLS Simulation*

Description

This method creates a ggplot2 visualization of results from a replext_gls object. It allows for flexible plotting of different metrics across various conditions.

Usage

```
## S3 method for class 'replext_gls'
plot(
  x,
  term,
  metric = "rejection_rates",
  x_axis = "tppp",
  color = "rho",
  title = NULL,
  x_label = NULL,
  y_label = NULL,
  ...
)
```

Arguments

x	An object of class "replext_gls", typically the output from replext_gls().
term	Character. The name of the model term to plot.
metric	Character. The metric to plot on the y-axis. Default is "rejection_rates". Other options include "mean_estimates", "mean_bias", etc.
x_axis	Character. The variable to plot on the x-axis. Default is "tppp" (time points per phase).
color	Character. The variable to use for color grouping. Default is "rho" (autocorrelation parameter).
title	Character. The plot title. If NULL, a default title is generated.
x_label	Character. The x-axis label. If NULL, a default label is generated.
y_label	Character. The y-axis label. If NULL, a default label is generated.
...	Pass extra arguments to ggplot.

Details

This method creates a line plot with points, where each line represents a different level of the color variable (default is rho). If the metric is "rejection_rates", error bars are added to represent the standard error of the rejection rates.

Value

A ggplot2 object representing the plot.

Examples

```
results <- replext_gls(
  n_timepoints_list = c(10),
  rho_list = c(0.2),
  iterations = 10,
  betas = c("(Intercept)" = 0, "phase1" = 1),
  formula = y ~ phase
)
plot(results, term = "phase1", metric = "rejection_rates")
```

---

print.cross_lagged *Print Method for Cross-Lagged Objects*

Description

Prints a summary of cross-lagged correlation analysis results. This method formats the results into a data frame showing lags, autocorrelation function (ACF) values, and indicates significance based on the confidence intervals. Significant ACF values, which fall outside the upper or lower confidence intervals, are marked with an asterisk (*).

Usage

```
## S3 method for class 'cross_lagged'
print(x, ...)
```

Arguments

x	An object of class "cross_lagged" containing the results from running `cross_lagged`. This object includes information on lags, ACF values, and confidence intervals.
...	Additional parameters (currently ignored).

Value

Invisibly returns a data frame with columns for lag, ACF values, and a significance indicator. This data frame is printed to the console for the user.

Examples

```
#Creating a sample dataset
reversal_withdrawal <- data.frame(
 phase = c(rep("baseline1", 6), rep("teratment1", 5), rep("baseline2", 5), rep("teratment2", 5)),
  time = 1:21,
  extbehavs = c(15, 10, 14, 17, 13, 12, 2, 1, 1, 0, 0, 9, 9, 11, 15, 20, 1, 0, 4, 0, 1)
```

```
)

reversal_withdrawal$synth <- sapply(reversal_withdrawal$time, function(x) {
  stats::rpois(1, x)
})

reversal_withdrawal <- as.data.frame(reversal_withdrawal)

# Using the cross_lagged function
cl_result <- cross_lagged(reversal_withdrawal, .x = "time", .y = "synth")

# Print the summary of cross-lagged analysis
print(cl_result)
```

`randomization_test` *Randomization Test for Single-Case Experiments*

Description

Performs a randomization test on data from single-case experiments. This function allows for the assessment of treatment effects by comparing the observed outcome difference to a distribution of differences obtained through permutation. It supports various constraints on permutation sequences, such as fixed or observed maximum and minimum consecutive sequences of a particular condition. The function also provides graphical summaries of the raw data and the distribution of mean differences.

Usage

```
randomization_test(
  .df,
  .out,
  .cond,
  .time,
  num_permutations = NULL,
  consec = c("observed", "fixed"),
  max_consec = NULL,
  min_consec = NULL,
  cond_levels = NULL,
  cond_labels = NULL,
  conf.level = 0.95,
  .bins = 30
)
```

Arguments

`.df`	A data frame containing the variables of interest.
`.out`	The name of the outcome variable in `.df`.

`.cond` The name of the condition variable in `.df`. This variable should have two levels.

`.time` The name of the time variable in `.df`.

`num_permutations` The number of permutations to perform. If `NULL`, all possible permutations are considered.

`consec` Specifies the constraint on consecutive sequences for permutation. Can be `"observed"` for the observed sequence length or `"fixed"` for a specified sequence length. Defaults to `"observed"`.

`max_consec` The maximum number of consecutive observations of the same condition to allow in permutations. If `NULL`, no maximum is enforced. To implement `max_consec`, `consec` must be set to `"fixed"`.

`min_consec` The minimum number of consecutive observations of the same condition to allow in permutations. If `NULL`, no minimum is enforced. To implement `min_consec`, `consec` must be set to `"fixed"`.

`cond_levels` Explicitly sets the levels of the condition variable. If `NULL`, the levels are derived from the data.

`cond_labels` Labels for the condition levels. If `NULL`, levels are used as labels.

`conf.level` The confidence level for the confidence interval calculation. Defaults to 0.95.

`.bins` The number of bins to use for the histogram of the test statistic distribution. Defaults to 30.

Value

A list containing the original difference in means, the p-value of the test, the distribution of test statistics under the null hypothesis, confidence intervals, and plots of the distribution of mean differences and the raw data.

References

Onghena, P. (2020). One by One: The design and analysis of replicated randomized single-case experiments.In R. van de Schoot & M. Mioecvic (Eds.), Small Sample Size Solutions: A Guide for Applied Researchers and Practitioners (1st ed., pp. 15). Routledge. doi:10.4324/9780429273872-8

Examples

```
result <- randomization_test(sleeping_pills, .out = "sever_compl",
                             .cond = "treatment", .time = "day",
                             num_permutations = 100,
                             cond_levels = c("C", "E"))

result$conf_int
```

raw_plot *Plot Raw Data with Optional Phase and Condition Annotations*

Description

This function generates a plot of raw data from a specified data frame. It supports optional annotations based on phase and condition variables, and can facet the plot by participant. The plot is customizable with parameters for setting factor levels and labels for both phase and condition variables. It utilizes ggplot2 for plotting.

Usage

```
raw_plot(
  .df,
  .out,
  .time,
  .phase = NULL,
  .cond = NULL,
  .participant = NULL,
  phase_levels = NULL,
  phase_labels = NULL,
  cond_levels = NULL,
  cond_labels = NULL,
  label_raise = 2
)
```

Arguments

.df	A data frame containing the data to be plotted. Must contain columns specified by .time, .out, and optionally .phase, .cond, and .participant if used.
.out	The name of the column in .df that contains the outcome variable to be plotted on the y-axis.
.time	The name of the column in .df that contains the time variable to be plotted on the x-axis.
.phase	(Optional) The name of the column in .df that contains the phase variable used for annotating the plot with phase changes. If NULL, phase annotations are not added.
.cond	(Optional) The name of the column in .df that contains the condition variable. If not NULL, data points are colored based on condition.
.participant	(Optional) The name of the column in .df that contains participant identifiers. If not NULL, the plot is faceted by participant.
phase_levels	(Optional) A vector of values indicating the order of phase levels. This is used to set the factor levels of the phase variable.
phase_labels	(Optional) A vector of labels corresponding to the phase levels. These labels are used in annotations.

cond_levels	(Optional) A vector of values indicating the order of condition levels. This is used to set the factor levels of the condition variable.
cond_labels	(Optional) A vector of labels corresponding to the condition levels. These are used for the legend.
label_raise	A numeric value indicating how much to raise the phase labels on the y-axis. Defaults to 2.

Value

A ggplot object representing the raw data plot with optional annotations for phase and condition, and faceting by participant if specified.

Examples

```
rp <- raw_plot(.df = efficacy_of_CBT, .out = "Anxious", .time = "time",
               .phase = "phase", phase_levels = c(0, 1),
               phase_labels = c("Exposure", "Exposure + CT"))
```

---

replext_gls *Replications and Extension of Generalized Least Squares Simulation*

Description

This function performs multiple simulations of Generalized Least Squares (GLS) models across various conditions, extending the work of Maric et al. (2014). It allows for the exploration of different numbers of timepoints per phase and autocorrelation parameters, providing a comprehensive analysis of model performance across these conditions.

Usage

```
replext_gls(
  n_timepoints_list,
  rho_list,
  iterations,
  n_phases = 2,
  n_IDs = 1,
  betas,
  formula,
  covariate_specs = NULL,
  alpha_level = 0.05,
  verbose = FALSE
)
```

Arguments

n_timepoints_list	Numeric vector. The numbers of timepoints per phase to simulate.
rho_list	Numeric vector. The autocorrelation parameters to simulate.
iterations	Integer. The number of simulations to run for each combination of conditions.
n_phases	Integer. The number of phases in the design. Default is 2.
n_IDs	Integer. The number of subjects (IDs) to simulate. Default is 1.
betas	Named numeric vector. The true coefficient values for the model.
formula	Formula object. The model formula to be fitted.
covariate_specs	List of lists. Specifications for generating covariates. Each inner list should contain elements 'vars' (variable names), 'dist' (distribution function name), 'args' (list of distribution parameters), and optionally 'expr' (a function to generate data).
alpha_level	Numeric. The significance level for hypothesis tests. Default is 0.05.
verbose	Logical. If TRUE, print progress messages. Default is FALSE.

Value

A data frame of class 'replext_gls' containing simulation results, including:

term	Character. The name of the model term.
tppp	Integer. The number of timepoints per phase.
rho	Numeric. The autocorrelation parameter.
success_rate	Numeric. The proportion of successful model fits.
mean_estimates	Numeric. The mean of the estimated coefficients.
mean_bias	Numeric. The mean bias of the estimated coefficients.
se_estimates	Numeric. The standard error of the estimated coefficients.
rejection_rates	Numeric. The proportion of significant hypothesis tests.
se_rejection_rates	Numeric. The standard error of the rejection rates.
mean_rmse	Numeric. The mean root mean square error of the estimates.
alpha_level	Numeric. The significance level used for hypothesis tests.
iterations	Integer. The number of iterations performed.
n_phases	Integer. The number of phases in the design.
n_IDs	Integer. The number of subjects simulated.
formula	Character. The model formula used.
covariate_specs	Character. The covariate specifications used.

References

Maric, M., de Haan, E., Hogendoorn, S.M., Wolters, L.H. & Huizenga, H.M. (2014). Evaluating statistical and clinical significance of intervention effects in single-case experimental designs: An SPSS method to analyze univariate data. Behavior Therapy. doi: 10.1016/j.beth.2014.09.009

Examples

```
results <- replext_gls(
  n_timepoints_list = c(10),
  rho_list = c(0.2),
  iterations = 10,
  betas = c("(Intercept)" = 0, "phase1" = 1),
  formula = y ~ phase
)
```

---

replext_pgsql *Replication and Extension of Generalized Least Squares Simulation Shiny App*

Description

This function creates a Shiny app for running and visualizing simulations of Generalized Least Squares (GLS) models, extending the work of Maric et al. (2014). It allows users to interactively set simulation parameters, run simulations, view results, and generate plots.

Usage

```
replext_pgsql(dbname, datatable, host, port, user, password)
```

Arguments

dbname	Character string. The name of the PostgreSQL database to connect to.
datatable	Character string. The name of the table in the database where results will be stored.
host	Character string. The host name or IP address of the PostgreSQL server.
port	Integer. The port number on which the PostgreSQL server is listening.
user	Character string. The username for the PostgreSQL database connection.
password	Character string. The password for the PostgreSQL database connection.

Details

The app provides a user interface for:

- Setting simulation parameters
- Running simulations based on the specified parameters
- Viewing simulation results in a table format
- Generating plots of various metrics across different conditions
- Storing and retrieving results from a PostgreSQL database
- Displaying relevant citations

The app uses the `replext_gls()` function to perform the actual simulations.

Value

A Shiny app object which can be run to start the application.

References

Maric, M., de Haan, E., Hogendoorn, S.M., Wolters, L.H. & Huizenga, H.M. (2014). Evaluating statistical and clinical significance of intervention effects in single-case experimental designs: An SPSS method to analyze univariate data. Behavior Therapy. doi: 10.1016/j.beth.2014.09.009

See Also

`replext_gls` for the underlying simulation function.

Examples

```
if(interactive()){
replext_pgsql(
  dbname = "my_database",
  datatable = "simulation_results",
  host = "localhost",
  port = 5432,
  user = "myuser",
  password = "mypassword"
)
}
```

reversal_withdrawal *Reversal/withdrawal Design Example*

Description

This is the data for the Reversal/Withdrawal Design Example presented in Figure 17 of the What Works Clearinghouse Procedures and Standards Handbook, Version 5.0

Usage

```
reversal_withdrawal
```

Format

`reversal_withdrawal`**:**

A data frame with 21 rows and 3 columns:

phase Study phase

time Time of data collection

extbehavs Externalizing behaviors score

Source

https://ies.ed.gov/ncee/WWC/Docs/referenceresources/Final_WWC-HandbookVer5_0-0-508.pdf

scdtb *Single Case Design Toolbox Shiny Application*

Description

This Shiny application provides a toolbox for analyzing single case design data. It includes features for data upload, data type handling, raw data visualization, mixed effects analysis, cross-lagged correlation analysis, non-overlap of all pairs analysis, and randomization tests.

Usage

```
scdtb()
```

Value

A Shiny app object which can be run to start the application.

References

Maric, M., & van der Werff, V. (2020). Single-Case Experimental Designs in Clinical Intervention Research. In R. van de Schoot & M. Miocevic (Eds.), Small Sample Size Solutions: A Guide for Applied Researchers and Practitioners (1st ed., pp. 10). Routledge. doi:10.4324/9780429273872-9

What Works Clearinghouse. (2022). What Works Clearinghouse procedures and standards handbook, version 5.0. U.S. Department of Education, Institute of Education Sciences, National Center for Education Evaluation and Regional Assistance (NCEE). This report is available on the What Works Clearinghouse website at https://ies.ed.gov/ncee/wwc/Handbooks

Onghena, P. (2020). One by One: The design and analysis of replicated randomized single-case experiments.In R. van de Schoot & M. Mioecvic (Eds.), Small Sample Size Solutions: A Guide for Applied Researchers and Practitioners (1st ed., pp. 15). Routledge. doi:10.4324/9780429273872-8

Examples

```
# To run the Shiny app
if(interactive()){
  scdtb()
}
```

`simulate_gls_once` *Simulate and Analyze Generalized Least Squares*

Description

This function simulates data and fits a Generalized Least Squares (GLS) model to the simulated data. It can handle multiple phases, multiple subjects (IDs), and various covariate specifications.

Usage

```
simulate_gls_once(
  n_timepoints_per_phase,
  rho,
  n_phases = 2,
  n_IDs = 1,
  betas,
  formula,
  covariate_specs = NULL
)
```

Arguments

`n_timepoints_per_phase`	Integer. The number of timepoints in each phase.
`rho`	Numeric. The autocorrelation parameter for the AR(1) error structure.

n_phases	Integer. The number of phases in the design. Default is 2.
n_IDs	Integer. The number of subjects (IDs) to simulate. Default is 1.
betas	Named numeric vector. The true coefficient values for the model.
formula	Formula object. The model formula to be fitted.
covariate_specs	List of lists. Specifications for generating covariates. Each inner list should contain elements 'vars' (variable names), 'dist' (distribution function name), args' (list of distribution parameters), and optionally 'expr' (a function to generate data).

Value

A list containing:

success	Logical. TRUE if the model converged, FALSE otherwise.
estimates	Named numeric vector. Estimated coefficients from the fitted model.
std_errors	Named numeric vector. Standard errors of the estimated coefficients.
p_values	Named numeric vector. P-values for the estimated coefficients.
bias	Named numeric vector. Bias of the estimated coefficients (estimate - true value).
rmse	Numeric. Root Mean Square Error of the estimated coefficients.

Examples

```
# Simple example with 2 phases, 10 timepoints per phase, and no covariates
result <- simulate_gls_once(
  n_timepoints_per_phase = 10,
  rho = 0.2,
  betas = c("(Intercept)" = 0, "phase1" = 1),
  formula = y ~ phase
)
```

---

sleeping_pills *Sleeping Pills and Dizziness Example*

Description

This is the data set used for the example in Onghena (2020).

Usage

```
sleeping_pills
```

Format

`sleeping_pills`**:**

A data frame with 14 rows and 3 columns:

day day of study

treatment E, Experimental; C, Control

sever_compl Severity of complaints

Source

doi:10.4324/9780429273872-8

References

Onghena, P. (2020). One by one: The design and analysis of replicated randomized single-case experiments. In R. van de Schoot & M. Miocevic (Eds.), Small sample size solutions: A guide for applied researchers and practitioners (1st ed., pp. 15). Routledge. doi:10.4324/9780429273872-8

Index