Package: DesignLibrary (via r-universe)

September 1, 2024

Type Package

Title Library of Research Designs

Version 0.1.10

Description A simple interface to build designs using the package 'DeclareDesign'. In one line of code, users can specify the parameters of individual designs and diagnose their properties. The designers can also be used to compare performance of a given design across a range of combinations of parameters, such as effect size, sample size, and assignment probabilities.

URL `https://declaredesign.org/r/designlibrary/`, `https://github.com/DeclareDesign/DesignLibrary`

License MIT + file LICENSE

Encoding UTF-8

Depends DeclareDesign (>= 0.17.0), R (>= 3.4.0), randomizr (>= 0.16.1), fabricatr (>= 0.8.0), estimatr (>= 0.16.0)

Imports generics, rlang, glue

Suggests testthat, knitr, rmarkdown

RoxygenNote 7.2.3

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

RemoteUrl https://github.com/declaredesign/designlibrary

RemoteRef HEAD

RemoteSha b3676fba248e83d9192a4874dbf3ff5dbb07ba87

Contents

`assignment_string` *Generates string of assignment of value to argument*

Description

Generates string of assignment of value to argument

Usage

```
assignment_string(arg_name, arg_values)
```

Arguments

`arg_name`	A string. Label of assignment object.
`arg_values`	A list. Values to be assigned to the argument. Can be character, logical or numeric of any length.

`binary_iv_designer` *Create a binary instrumental variables design*

Description

Builds a design with one instrument, one binary explanatory variable, and one outcome.

Usage

```
binary_iv_designer(
  N = 100,
  type_probs = c(1/3, 1/3, 1/3, 0),
  assignment_probs = c(0.5, 0.5, 0.5, 0.5),
  a_Y = 1,
  b_Y = 0,
  d_Y = 0,
  outcome_sd = 1,
  a = c(1, 0, 0, 0) * a_Y,
  b = rep(b_Y, 4),
  d = rep(d_Y, 4),
  args_to_fix = NULL
)
```

Arguments

`N`	An integer. Sample size.
`type_probs`	A vector of four numbers in [0,1]. Probability of each complier type (always-taker, never-taker, complier, defier).
`assignment_probs`	A vector of four numbers in [0,1]. Probability of assignment to encouragement (Z) for each complier type (always-taker, never-taker, complier, defier). Under random assignment these are normally identical since complier status is not known to researchers in advance.
`a_Y`	A real number. Constant in Y equation. Assumed constant across types. Overridden by `a` if specified.
`b_Y`	A real number. Effect of X on Y equation. Assumed constant across types. Overridden by `b` if specified.
`d_Y`	A real number. Effect of Z on Y. Assumed constant across types. Overridden by `d` if specified.
`outcome_sd`	A real number. The standard deviation of the outcome.
`a`	A vector of four numbers. Constant in Y equation for each complier type (always-taker, never-taker, complier, defier).
`b`	A vector of four numbers. Slope on X in Y equation for each complier type (always-taker, never-taker, complier, defier).

`d`	A vector of four numbers. Slope on Z in Y equation for each complier type (non zero implies violation of exclusion restriction).
`args_to_fix`	A character vector. Names of arguments to be args_to_fix in design.

Details

A researcher is interested in the effect of binary X on outcome Y. The relationship is confounded because units that are more likely to be assigned to X=1 have higher Y outcomes. A potential instrument Z is examined, which plausibly causes X. The instrument can be used to assess the effect of X on Y for units whose value of X depends on Z if Z does not negatively affect X for some cases, affects X positively for some, and affects Y only through X.

See vignette online for more details on estimands.

Value

A simple instrumental variables design with binary instrument, treatment, and outcome variables.

Author(s)

DeclareDesign Team

Examples

```
# Generate a simple iv design: iv identifies late not ate
binary_iv_design_1 <- binary_iv_designer(N = 1000, b = c(.1, .2, .3, .4))
## Not run:
diagnose_design(binary_iv_design_1)

## End(Not run)

# Generates a simple iv design with violation of monotonicity
binary_iv_design_2 <- binary_iv_designer(type_probs = c(.1,.1,.6, .2), b_Y = .5)
## Not run:
diagnose_design(binary_iv_design_2)

## End(Not run)

# Generates a simple iv design with violation of exclusion restriction
binary_iv_design_3 <- binary_iv_designer(d_Y = .5, b_Y = .5)
## Not run:
diagnose_design(binary_iv_design_3)

## End(Not run)

# Generates a simple iv design with violation of randomization
binary_iv_design_4 <- binary_iv_designer(N = 1000, assignment_probs = c(.2, .3, .7, .5), b_Y = .5)
## Not run:
diagnose_design(binary_iv_design_4)

## End(Not run)
```

```
# Generates a simple iv design with violation of first stage
binary_iv_design_5 <- binary_iv_designer(type_probs = c(.5,.5, 0, 0), b_Y = .5)
## Not run:
diagnose_design(binary_iv_design_5)

## End(Not run)
```

`block_cluster_two_arm_designer`
Create a two-arm design with blocks and clusters

Description

Builds a two-arm design with blocks and clusters.

Usage

```
block_cluster_two_arm_designer(
  N = NULL,
  N_blocks = 1,
  N_clusters_in_block = ifelse(is.null(N), 100, round(N/N_blocks)),
 N_i_in_cluster = ifelse(is.null(N), 1, round(N/mean(N_blocks * N_clusters_in_block))),
  sd = 1,
  sd_block = 0.5773 * sd,
  sd_cluster = max(0, (sd^2 - sd_block^2)/2)^0.5,
  sd_i_0 = max(0, sd^2 - sd_block^2 - sd_cluster^2)^0.5,
  sd_i_1 = sd_i_0,
  rho = 1,
  assignment_probs = 0.5,
  control_mean = 0,
  ate = 0,
  treatment_mean = control_mean + ate,
  verbose = TRUE,
  args_to_fix = NULL
)
```

Arguments

`N`	An integer. Total number of units. Usually not specified as `N` is determined by `N_blocks`, `N_clusters_in_block`, and `N_i_in_cluster`. If `N_blocks`, and `N_clusters_in_block`, and `N_i_in_cluster` are specified then `N` is overridden. If these are not specified and `N` is specified then designer attempts to guess sizes of levels to approximate `N`, with preference for a design without blocks or clusters.
`N_blocks`	An integer. Number of blocks. Defaults to 1 for no blocks.

`N_clusters_in_block`	An integer or vector of integers of length `N_blocks`. Number of clusters in each block. This is the total `N` when `N_blocks` and `N_i_in_cluster` are at default values.
`N_i_in_cluster`	An integer or vector of integers of length `sum(N_clusters_in_block)`. Individuals per cluster. Defaults to 1 for no clusters.
`sd`	A nonnegative number. Overall standard deviation (combining individual level, cluster level, and block level shocks). Defaults to 1. Overridden if incompatible with other user-specified shocks.
`sd_block`	A nonnegative number. Standard deviation of block level shocks.
`sd_cluster`	A nonnegative number. Standard deviation of cluster level shock.
`sd_i_0`	A nonnegative number. Standard deviation of individual level shock in control. If not specified, and when possible given `sd_block` and `sd_cluster`, `sd_i_0` defaults to make total variance = sd.
`sd_i_1`	A nonnegative number. Standard deviation of individual level shock in treatment. Defaults to `sd_i_0`.
`rho`	A number in [-1,1]. Correlation in individual shock between potential outcomes for treatment and control.
`assignment_probs`	A number or vector of numbers in (0,1). Treatment assignment probability for each block (specified in order of `N_clusters_in_block`).
`control_mean`	A number. Average outcome in control.
`ate`	A number. Average treatment effect. Alternative to specifying `treatment_mean`. Note that `ate` is an argument for the designer but it does not appear as an argument in design code (design code uses `control_mean` and `treatment_mean` only).
`treatment_mean`	A number. Average outcome in treatment. If `treatment_mean` is not provided then it is calculated as `control_mean + ate`. If both `ate` and `treatment_mean` are provided then only `treatment_mean` is used.
`verbose`	Logical. If TRUE, prints intra-cluster correlation implied by design parameters.
`args_to_fix`	A character vector. Names of arguments to be args_to_fix in design.

Details

Units are assigned to treatment using complete block cluster random assignment. Treatment effects can be specified either by providing `control_mean` and `treatment_mean` or by specifying an `ate`. Estimation uses differences in means accounting for blocks and clusters.

In the usual case `N` is not provided by the user but is determined by `N_blocks`, `N_clusters_in_block`, `N_i_in_cluster` (when these are integers `N` is the product of these three numbers).

Normal shocks can be specified at the individual, cluster, and block levels. If individual level shocks are not specified and cluster and block level variances sum to less than 1, then individual level shocks are set such that total variance in outcomes equals 1.

Key limitations: The designer assumes covariance between potential outcomes at the individual level only.

See vignette online.

Value

A block cluster two-arm design.

Author(s)

DeclareDesign Team

Examples

```
# Generate a design using default arguments:
block_cluster_two_arm_design <- block_cluster_two_arm_designer()
block_cluster_uneven <- block_cluster_two_arm_designer(
       N_blocks = 3, N_clusters_in_block = 2:4, N_i_in_cluster = 1:9)
# A design in which number of clusters of cluster size is not specified
# but N and block size are:
block_cluster_guess <- block_cluster_two_arm_designer(N = 24, N_blocks = 3)
```

`cluster_sampling_designer`

Create a design for cluster random sampling

Description

Builds a cluster sampling design for an ordinal outcome variable for a population with `N_blocks` strata, each with `N_clusters_in_block` clusters, each of which contains `N_i_in_cluster` units. The sampling strategy involves sampling `n_clusters_in_block` clusters in each stratum, and then sampling `n_i_in_cluster` units in each cluster. Outcomes within clusters have intra-cluster correlation approximately equal to `ICC`.

Usage

```
cluster_sampling_designer(
  N_blocks = 1,
  N_clusters_in_block = 1000,
  N_i_in_cluster = 50,
  n_clusters_in_block = 100,
  n_i_in_cluster = 10,
  icc = 0.2,
  args_to_fix = NULL
)
```

Arguments

`N_blocks`	An integer. Number of blocks (strata). Defaults to 1 for no blocks.
`N_clusters_in_block`	An integer or vector of integers of length `N_blocks`. Number of clusters in each block in the population.

`N_i_in_cluster`	An integer or vector of integers of length `sum(N_clusters_in_block)`. Number of units per cluster sampled.
`n_clusters_in_block`	An integer. Number of clusters to sample in each block (stratum).
`n_i_in_cluster`	An integer. Number of units to sample in each cluster.
`icc`	A number in [0,1]. Intra-cluster Correlation Coefficient (ICC).
`args_to_fix`	A character vector. Names of arguments to be args_to_fix in design.

Details

Key limitations: The design assumes a args_to_fix number of clusters drawn in each stratum and a args_to_fix number of individuals drawn from each cluster.

See vignette online.

Value

A stratified cluster sampling design.

Author(s)

DeclareDesign Team

Examples

```
# To make a design using default arguments:
cluster_sampling_design <- cluster_sampling_designer()
# A design with varying block size and varying cluster size
cluster_sampling_design <- cluster_sampling_designer(
  N_blocks = 2, N_clusters_in_block = 6:7, N_i_in_cluster = 3:15,
  n_clusters_in_block = 5,  n_i_in_cluster = 2)
```

`code_fixer` *Substitute approach*

Description

Substitute approach

Usage

```
code_fixer(design_expr, list_fixed_str, eval_envir)
```

Arguments

`design_expr`	A string. The text of the expression in which you wish to substitute symbols for their set values.
`list_fixed_str`	A string. The string of code that generates a named list of arguments that will be substituted in the evaluated `design_expr`.
`eval_envir`	The evaluation environment. Defaults to environment in which design arguments are already evaluated.

`construct_design_code` *Generates clean code string that reproduces design*

Description

Generates clean code string that reproduces design

Usage

```
construct_design_code(
  designer,
  args,
  args_to_fix = NULL,
  arguments_as_values = FALSE,
  exclude_args = NULL
)
```

Arguments

`designer`	Designer function.
`args`	Named list of arguments to be passed to designer function.
`args_to_fix`	Vector of strings. Designer arguments to fix in design code.
`arguments_as_values`	Logical. Whether to replace argument names for value.
`exclude_args`	Vector of strings. Name of arguments to be excluded from argument definition at top of design code.

`DesignLibrary` *DesignLibrary: A package for creating designs*

Description

Library of Research Designs

`factorial_designer` *Create a factorial design*

Description

A `2^k` factorial designer with `k` factors assigned with independent probabilities. Results in `2^k` treatment combinations, each with independent, normally distributed shocks. Estimands are average effects and average interactions of given conditions, averaged over other conditions. Estimation uses regression of demeaned variables with propensity weights.

Usage

```
factorial_designer(
  N = 256,
  k = 3,
  outcome_means = rep(0, 2^k),
  sd = 1,
  outcome_sds = rep(sd, 2^k),
  assignment_probs = rep(0.5, k),
  outcome_name = "Y",
  treatment_names = NULL,
  args_to_fix = NULL
)
```

Arguments

`N`	An integer. Size of sample.
`k`	An integer. The number of factors in the design.
`outcome_means`	A numeric vector of length `2^k`. Means for each of the `2^k` treatment combinations. See 'Details' for the correct order of values.
`sd`	A nonnegative number. Standard deviation for outcomes when all outcomes have identical standard deviations. For outcome-specific standard deviations use `outcomes_sds`.
`outcome_sds`	A non negative numeric vector of length `2^k`. Standard deviations for each of the treatment combinations. See 'Details' for the correct order of values.
`assignment_probs`	A numeric vector of length `k`. Independent probability of assignment to each treatment.
`outcome_name`	A character. Name of outcome variable (defaults to "Y"). Must be provided without spacing inside the function `c()` as in `outcome_name = c("War")`.
`treatment_names`	A character vector of length `k`. Name of treatment factors variable (defaults to "T1", "T2", ..., "Tk"). Must be provided without spacing.
`args_to_fix`	A character vector. Names of arguments to be args_to_fix in design. By default `k`, `probs`, `outcome_name`, and `treatment_names` are always args_to_fix.

Details

`factorial_designer` creates a factorial design with `2^k` treatment combinations resulting from `k` factors, each with two conditions each (`c(0,1)`). The order of the scalar arguments `outcome_means` and `outcome_sds` must follow the one returned by `expand.grid(rep(list(c(0,1)), k))`, where each of the columns is a treatment.

Estimands are defined for each combination of treatment assignment as linear combinations of potential outcomes, typically weighted averages of differences. Note that the weighting for the estimand does not reflect treatment assignment probabilities but rather weights each possible condition equally.

For example, in a design with $k = 3$ factors, the treatment effect of A, (TE_A), averaged over conditions defined by B and C, is given by:

$$TE_A = 1/4 * (Y_{111} - Y_{011}) + 1/4 * (Y_{101} - Y_{001}) + 1/4 * (Y_{110} - Y_{010}) + 1/4 * (Y_{100} - Y_{000}).$$

The "average interaction of A and B" — that is the average effect (for a single unit) of A on the effect of B across conditions defined by C — is:

$$TE_{AB} = 1/2 * [(Y_{111} - Y_{011}) - (Y_{101} - Y_{001})] + 1/2 * [(Y_{110} - Y_{010}) - (Y_{100} - Y_{000})].$$

And the triple interaction—that is, the effect of C on the the effect of B on the effect of A is:

$$TE_{ABC} = [(Y_{111} - Y_{011}) - (Y_{101} - Y_{001})] - [(Y_{110} - Y_{010}) - (Y_{100} - Y_{000})],$$

where Y_{abc} is short for the potential outcome of Y when A is a, B is b, and C is c.

Estimates draw from a regression in which all treatments are demeaned and weighted by the inverse probability of being in the condition they are in. Note that in this demeaned regression the constant captures the average outcome across all conditions — not the outcome when all units are in the control condition. The coefficient on T1 captures the average effect of T1 across other conditions—not the effect of T1 when other conditions are at 0. And so on.

Value

A factorial design.

Author(s)

DeclareDesign Team

Examples

```
# A factorial design using default arguments
factorial_design <- factorial_designer()

# A 2 x 2 x 2 factorial design with unequal probabilities of assignment to
# each treatment condition. In this case the estimator weights up by the
# conditional probabilities of assignment.
factorial_design_2 <- factorial_designer(k = 3,
                                         assignment_probs = c(1/2, 1/4, 1/8),
                                         outcome_means = c(0,0,0,0,0,0,0,4))
## Not run:
```

```
diagnose_design(factorial_design_2)

## End(Not run)
# Mapping from outcomes to estimands
# The mapping between the potential outcomes schedule and the estimands of
# interest is not always easy. To help with intuition consider a 2^3
# factorial design. You might like to think of a data generating process as
# a collection of marginal effects and interaction effects mapping from
# treatments to outcomes.
# For instance: Y = -.25 + .75*X1 - .25*X2 -.25*X3 + X1*X2*X3
# The vector of implied potential outcome means as a function of conditions
# could then be generated like this:

X <- expand.grid(rep(list(c(0,1)), 3))
outcome_means =  -.25 + X[,1]*3/4 - X[,2]/4 - X[,3]/4 + X[,1]*X[,2]*X[,3]
outcomes <- cbind(X, outcome_means)
colnames(outcomes) <- c("X1", "X2", "X3", "mean")
outcomes

# Examination of the outcomes in this table reveals that there is an
# average outcome of 0 (over all conditions), an average effect of treatment
# X1 of 1,  an average effects for X2 and X3 of 0,  the two way interactions
# are .5 (averaged over conditions of the third treatment) and the triple
# interaction is 1.
# These are exactly the estimands calculated by the designer and returned in
# diagnosis.
factorial_design_3 <- factorial_designer(k = 3,
                                         outcome_means = outcome_means,
                                         outcome_sds = rep(.01, 8))
## Not run:
library(DeclareDesign)
diagnose_design(factorial_design_3, sims = 10)

## End(Not run)
```

get_design_code — *Get the code from a design*

Description

Get the code from a design

Usage

```
get_design_code(design)
```

Arguments

design	A design that has code as an attribute.

match.call.defaults *Argument matching with defaults*

Description

This is a version of `match.call` which also includes default arguments.

Usage

```
match.call.defaults(
  definition = sys.function(sys.parent()),
  call = sys.call(sys.parent()),
  expand.dots = TRUE,
  envir = parent.frame(2L)
)
```

Arguments

`definition`	a function, by default the function from which match.call is called. See details.
`call`	an unevaluated call to the function specified by definition, as generated by call.
`expand.dots`	ogical. Should arguments matching ... in the call be included or left as a ... argument?
`envir`	an environment, from which the ... in call are retrieved, if any.

Value

An object of class call.

Author(s)

Neal Fultz

Examples

```
foo <- function(x=NULL,y=NULL,z=4, dots=TRUE, ...) {
  match.call.defaults(expand.dots=dots)
}
```

mediation_analysis_designer

Create a design for mediation analysis

Description

A mediation analysis design that examines the effect of treatment (Z) on mediator (M) and the effect of mediator (M) on outcome (Y) (given Z=0) as well as direct effect of treatment (Z) on outcome (Y) (given M=0). Analysis is implemented using an interacted regression model. Note this model is not guaranteed to be unbiased despite randomization of Z because of possible violations of sequential ignorability.

Usage

```
mediation_analysis_designer(
  N = 200,
  a = 1,
  b = 0.4,
  c = 0,
  d = 0.5,
  rho = 0,
  args_to_fix = NULL
)
```

Arguments

`N`	An integer. Size of sample.
`a`	A number. Parameter governing effect of treatment (Z) on mediator (M).
`b`	A number. Effect of mediator (M) on outcome (Y) when Z = 0.
`c`	A number. Interaction between mediator (M) and (Z) for outcome (Y).
`d`	A number. Direct effect of treatment (Z) on outcome (Y), when M = 0.
`rho`	A number in [-1,1]. Correlation between mediator (M) and outcome (Y) error terms. Non zero correlation implies a violation of sequential ignorability.
`args_to_fix`	A character vector. Names of arguments to be args_to_fix in design.

Details

See vignette online.

Value

A mediation analysis design.

Author(s)

DeclareDesign Team

Examples

```
# Generate a mediation analysis design using default arguments:
mediation_1 <- mediation_analysis_designer()
draw_estimands(mediation_1)
## Not run:
diagnose_design(mediation_1, sims = 1000)

## End(Not run)

# A design with a violation of sequential ignorability and heterogeneous effects:
mediation_2 <- mediation_analysis_designer(a = 1, rho = .5, c = 1, d = .75)
draw_estimands(mediation_2)
## Not run:
diagnose_design(mediation_2, sims = 1000)

## End(Not run)
```

`multi_arm_designer` *Create a design with multiple experimental arms*

Description

Creates a design with `m_arms` experimental arms, each assigned with equal probability.

Usage

```
multi_arm_designer(
  N = 30,
  m_arms = 3,
  outcome_means = rep(0, m_arms),
  sd_i = 1,
  outcome_sds = rep(0, m_arms),
  conditions = 1:m_arms,
  args_to_fix = NULL
)
```

Arguments

`N`	An integer. Sample size.
`m_arms`	An integer. Number of arms.
`outcome_means`	A numeric vector of length `m_arms`. Average outcome in each arm.
`sd_i`	A nonnegative scalar. Standard deviation of individual-level shock (common across arms).
`outcome_sds`	A nonnegative numeric vector of length `m_arms`. Standard deviations for condition-level shocks.

`conditions`	A vector of length `m_arms`. The names of each arm. It can be given as numeric or character class (without blank spaces).
`args_to_fix`	A character vector. Names of arguments to be args_to_fix in design. By default, `m_arms` and `conditions` are always args_to_fix.

Details

See vignette online.

Value

A function that returns a design.

Author(s)

DeclareDesign Team

Examples

```
# To make a design using default arguments:
design <- multi_arm_designer()


# A design with different means and standard deviations in each arm
design <- multi_arm_designer(outcome_means = c(0, 0.5, 2), outcome_sds =  c(1, 0.1, 0.5))

design <- multi_arm_designer(N = 80, m_arms = 4, outcome_means = 1:4,
                             args_to_fix = c("outcome_means", "outcome_sds"))
```

`pretest_posttest_designer`

Create a pretest-posttest design

Description

Produces a design in which an outcome Y is observed pre- and post-treatment. The design allows for individual post-treatment outcomes to be correlated with pre-treatment outcomes and for at-random missingness in the observation of post-treatment outcomes.

Usage

```
pretest_posttest_designer(
  N = 100,
  ate = 0.25,
  sd_1 = 1,
  sd_2 = 1,
  rho = 0.5,
  attrition_rate = 0.1,
  args_to_fix = NULL
)
```

Arguments

`N`	An integer. Size of sample.
`ate`	A number. Average treatment effect.
`sd_1`	Nonnegative number. Standard deviation of period 1 shocks.
`sd_2`	Nonnegative number. Standard deviation of period 2 shocks.
`rho`	A number in [-1,1]. Correlation in outcomes between pre- and post-test.
`attrition_rate`	A number in [0,1]. Proportion of respondents in pre-test data that appear in post-test data.
`args_to_fix`	A character vector. Names of arguments to be args_to_fix in design.

Details

See vignette online.

Value

A pretest-posttest design.

Author(s)

DeclareDesign Team

Examples

```
# Generate a pre-test post-test design using default arguments:
pretest_posttest_design <- pretest_posttest_designer()
```

`process_tracing_designer`

Create a process-tracing design

Description

Builds a design in which two pieces of evidence are sought and used to update about whether X caused Y using Bayes' rule.

Usage

```
process_tracing_designer(
  N = 100,
  prob_X = 0.5,
  process_proportions = c(0.25, 0.25, 0.25, 0.25),
  prior_H = 0.5,
  p_E1_H = 0.8,
  p_E1_not_H = 0.2,
  p_E2_H = 0.3,
  p_E2_not_H = 0,
  cor_E1E2_H = 0,
  cor_E1E2_not_H = 0,
  label_E1 = "Straw in the Wind",
  label_E2 = "Smoking Gun",
  args_to_fix = NULL
)
```

Arguments

`N`	An integer. Size of population of cases from which a single case is selected.
`prob_X`	A number in [0,1]. Probability that X = 1 for a given case (equal throughout population of cases).
`process_proportions`	A vector of numbers in [0,1] that sums to 1. Simplex denoting the proportion of cases in the population in which, respectively: 1) X causes Y; 2) Y occurs regardless of X; 3) X causes the absence of Y; 4) Y is absent regardless of X.
`prior_H`	A number in [0,1]. Prior probability that X causes Y in a given case in which X and Y are both present.
`p_E1_H`	A number in [0,1]. Probability of observing first piece of evidence given hypothesis that X caused Y is true.
`p_E1_not_H`	A number in [0,1]. Probability of observing first piece of evidence given hypothesis that X caused Y is not true.
`p_E2_H`	A number in [0,1]. Probability of observing second piece of evidence given hypothesis that X caused Y is true.
`p_E2_not_H`	A number in [0,1]. Probability of observing second piece of evidence given hypothesis that X caused Y is not true.
`cor_E1E2_H`	A number in [-1,1]. Correlation between first and second pieces of evidence given hypothesis that X caused Y is true.
`cor_E1E2_not_H`	A number in [-1,1]. Correlation between first and second pieces of evidence given hypothesis that X caused Y is not true.
`label_E1`	A string. Label for the first piece of evidence (e.g., "Straw in the Wind").
`label_E2`	A string. Label for the second piece of evidence (e.g., "Smoking Gun").
`args_to_fix`	A character vector. Names of arguments to be args_to_fix in design.

Details

The model posits a population of `N` cases, each of which does or does not exhibit the presence of some outcome, Y. With probability `prob_X`, each case also exhibits the presence or absence of some potential cause, X. The outcome Y can be realized through four distinct causal relations, distributed through the population of cases according to `process_proportions`. First, the presence of X might cause Y. Second, the absence of X might cause Y. Third, Y might be present irrespective of X. Fourth, Y might be absent irrespective of X.

Our inquiry is a "cause of effects" question. We wish to know whether a specific case was one in which the presence (absence) of X caused the presence (absence) of Y.

Our data strategy consists of selecting one case at random in which both X and Y are present. As part of the data strategy we seek two pieces of evidence in favor of or against the hypothesized causal relationship, H, in which X causes Y.

The first (second) piece of evidence is observed with probability `p_E1_H` (`p_E2_H`) when H is true, and with probability `p_E1_not_H` (`p_E2_not_H`) when H is false.

Conditional on H being true (false), the correlation between the two pieces of evidence is given by `cor_E1E2_H` (`cor_E1E2_not_H`).

The researcher uses Bayes' rule to update about the probability that X caused Y given the evidence. In other words, they form a posterior inference, Pr(H|E). We specify four answer strategies for forming this inference. The first simply ignores the evidence and is equivalent to stating a prior belief without doing any causal process tracing. The second conditions inferences only on the first piece of evidence, and the third only on the second piece of evidence. The fourth strategy conditions posterior inferences on both pieces of evidence simultaneously.

We specify as diagnosands for this design the bias, RMSE, mean(estimand), mean(estimate) and sd(estimate).

Value

A process-tracing design.

Author(s)

DeclareDesign Team

Examples

```
# Generate a process-tracing design using default arguments:
pt_1 <- process_tracing_designer()
draw_estimands(pt_1)
draw_estimates(pt_1)
draw_data(pt_1)
## Not run:
diagnose_design(pt_1, sims = 1000)

## End(Not run)

# A design in which the smoking gun and straw-in-the-wind are correlated
pt_2 <- process_tracing_designer(cor_E1E2_H = .32)
## Not run:
```

```
diagnose_design(pt_2, sims = 1000)

## End(Not run)

# A design with two doubly-decisive tests pointing in opposite directions
pt_3 <- process_tracing_designer(p_E1_H = .80,p_E1_not_H = .05,
                                 label_E1 = "Doubly-Decisive: H",
                                 p_E2_H = .05,p_E2_not_H = .80,
                                 label_E2 = "Doubly-Decisive: Not H")
draw_estimates(pt_3)
## Not run:
diagnose_design(pt_3, sims = 1000)

## End(Not run)
```

---

randomized_response_designer

Create a randomized response design

Description

Produces a (forced) randomized response design that measures the share of individuals with a given trait `prevalence_trait` in a population of size `N`. Probability of forced response ("Yes") is given by `prob_forced_yes`, and rate at which individuals with trait lie is given by `withholding_rate`.

Usage

```
randomized_response_designer(
  N = 1000,
  prob_forced_yes = 0.6,
  prevalence_rate = 0.1,
  withholding_rate = 0.5,
  args_to_fix = NULL
)
```

Arguments

`N`	An integer. Size of sample.
`prob_forced_yes`	A number in [0,1]. Probability of a forced yes.
`prevalence_rate`	A number in [0,1]. Probability that individual has the sensitive trait.
`withholding_rate`	A number in [0,1]. Probability that an individual with the sensitive trait hides it.
`args_to_fix`	A character vector. Names of arguments to be args_to_fix in design.

Details

`randomized_response_designer` employs a specific variation of randomized response designs in which respondents are required to report a args_to_fix answer to the sensitive question with a given probability (see Blair, Imai, and Zhou (2015) for alternative applications and estimation strategies).

See vignette online.

Value

A randomized response design.

Author(s)

DeclareDesign Team

Examples

```
# Generate a randomized response design using default arguments:
randomized_response_design <- randomized_response_designer()
```

`regression_discontinuity_designer`
Create a regression discontinuity design

Description

Builds a design with sample from population of size `N`. The average treatment effect local to the cutpoint is equal to `tau`. It allows for specification of the order of the polynomial regression (`poly_reg_order`), cutoff value on the running variable (`cutoff`), and size of bandwidth around the cutoff (`bandwidth`). By providing a vector of numbers to `control_coefs` and `treatment_coefs`, users can also specify polynomial regression coefficients that generate the expected control and treatment potential outcomes given the running variable.

Usage

```
regression_discontinuity_designer(
  N = 1000,
  tau = 0.15,
  outcome_sd = 0.1,
  cutoff = 0.5,
  bandwidth = 0.5,
  control_coefs = c(0.5, 0.5),
  treatment_coefs = c(-5, 1),
  poly_reg_order = 4,
  args_to_fix = NULL
)
```

Arguments

`N`	An integer. Size of population to sample from.
`tau`	A number. Difference in potential outcomes functions at the threshold.
`outcome_sd`	A positive number. The standard deviation of the outcome.
`cutoff`	A number in (0,1). Threshold on running variable beyond which units are treated.
`bandwidth`	A number. The value of the bandwidth on both sides of the threshold from which to include units.
`control_coefs`	A vector of numbers. Coefficients for polynomial regression function that generates control potential outcomes. Order of polynomial is equal to length.
`treatment_coefs`	A vector of numbers. Coefficients for polynomial regression function that generates treatment potential outcomes. Order of polynomial is equal to length.
`poly_reg_order`	Integer greater than or equal to 1. Order of the polynomial regression used to estimate the jump at the cutoff.
`args_to_fix`	A character vector. Names of arguments to be args_to_fix in design.

Details

See vignette online.

Value

A regression discontinuity design.

Author(s)

DeclareDesign Team

Examples

```
# Generate a regression discontinuity design using default arguments:
regression_discontinuity_design <- regression_discontinuity_designer()
```

`return_args` *Generates character string for non-fixed arguments in a designer using substitution approach.*

Description

Generates character string for non-fixed arguments in a designer using substitution approach.

Usage

```
return_args(args, fixes)
```

Arguments

`args`	Function arguments.
`fixes`	Function arguments that are fixed (i.e., already evaluated in body of function)

`spillover_designer` *Create a design with spillovers*

Description

Builds a design with `N_groups` groups each containing `N_i_group` individuals. Potential outcomes exhibit spillovers: if any individual in a group receives treatment, the effect is spread equally among members of the group.

Usage

```
spillover_designer(
  N_groups = 80,
  N_i_group = 3,
  sd_i = 0.2,
  gamma = 2,
  args_to_fix = NULL
)
```

Arguments

`N_groups`	An integer. Number of groups.
`N_i_group`	Number of units in each group. Can be scalar or vector of length `N_groups`.
`sd_i`	A nonnegative number. Standard deviation of individual-level shock.
`gamma`	A number. Parameter that controls whether spillovers within groups substitute or complement each other. See 'Details'.
`args_to_fix`	A character vector. Names of arguments to be args_to_fix in design.

Details

Parameter `gamma` controls interactions between spillover effects.For `gamma`=1 for every $1 given to a member of a group, each member receives $1*`N_i_group` no matter how many others are already treated. For `gamma`>1 (<1) for every $1 given to a member of a group, each member receives an amount that depends negatively (positively) on the number already treated.

The default estimand is the average difference across subjects between no one treated and only that subject treated.

Value

A simple spillover design.

Author(s)

DeclareDesign Team

Examples

```
# Generate a simple spillover design using default arguments:
spillover_design <- spillover_designer()
```

---

str_within — *Takes substring between matched strings. Avoids dependency on stringr package.*

Description

Takes substring between matched strings. Avoids dependency on stringr package.

Usage

```
str_within(string, pattern = "^(structure\\()|(, \\.Names)")
```

Arguments

`string`	A string. String from which substring is extracted.
`pattern`	A regular expression that matches the beggining and end of a substring

---

sub_expr_text — *Substitute text from expressions in design code*

Description

Substitute text from expressions in design code

Usage

```
sub_expr_text(code, ...)
```

Arguments

`code`	List contaitining design code.
`...`	List of expressions to be substituted for their text.

Value

Code with expression text.

two_arm_attrition_designer

Create design with risk of attrition or post treatment conditioning

Description

Creates a two-arm design with application for when estimand of interest is conditional on a post-treatment outcome (the effect on Y given R) or data is conditionally observed (Y given R). See ‘Details‘ for more information on the data generating process.

Usage

```
two_arm_attrition_designer(
  N = 100,
  a_R = 0,
  b_R = 1,
  a_Y = 0,
  b_Y = 1,
  rho = 0,
  args_to_fix = NULL
)
```

Arguments

`N`	An integer. Size of sample.
`a_R`	A number. Constant in equation relating treatment to responses.
`b_R`	A number. Slope coefficient in equation relating treatment to responses.
`a_Y`	A number. Constant in equation relating treatment to outcome.
`b_Y`	A number. Slope coefficient in equation relating treatment to outcome.
`rho`	A number in [0,1]. Correlation between shocks in equations for R and Y.
`args_to_fix`	A character vector. Names of arguments to be args_to_fix in design.

Details

The data generating process is of the form:

`R ~ (a_R + b_R*Z > u_R)`

`Y ~ (a_Y + b_Y*Z > u_Y)`

where `u_R` and `u_Y` are joint normally distributed with correlation `rho`.

Value

A post-treatment design.

Author(s)

DeclareDesign Team

Examples

```
# To make a design using default argument (missing completely at random):
two_arm_attrition_design <- two_arm_attrition_designer()
## Not run:
diagnose_design(two_arm_attrition_design)

## End(Not run)
# Attrition can produce bias even for unconditional ATE even when not
# associated with treatment
## Not run:
diagnose_design(two_arm_attrition_designer(b_R = 0, rho = .3))

## End(Not run)
# Here the linear estimate using R=1 data is unbiased for
# "ATE on Y (Given R)" with b_R = 0 but not when  b_R = 1
## Not run:
diagnose_design(redesign(two_arm_attrition_design, b_R = 0:1, rho = .2))

## End(Not run)
```

two_arm_covariate_designer

Create a simple two arm design with a possibly prognostic covariate

Description

Builds a design with one treatment and one control arm. Treatment effects can be specified either by providing `control_mean` and `treatment_mean` or by specifying a `control_mean` and `ate`. Non random assignment is specified by a possible correlation, `rho_WZ`, between `W` and a latent variable that determines the probability of `Z`. Nonignorability is specified by a possible correlation, `rho_WY`, between `W` and outcome `Y`.

Usage

```
two_arm_covariate_designer(
  N = 100,
  prob = 0.5,
  control_mean = 0,
  sd = 1,
  ate = 1,
  h = 0,
  treatment_mean = control_mean + ate,
  rho_WY = 0,
  rho_WZ = 0,
  args_to_fix = NULL
)
```

Arguments

`N`	An integer. Sample size.
`prob`	A number in [0,1]. Probability of assignment to treatment.
`control_mean`	A number. Average outcome in control.
`sd`	A positive number. Standard deviation of shock on Y.
`ate`	A number. Average treatment effect.
`h`	A number. Controls heterogeneous treatment effects by W. Defaults to 0.
`treatment_mean`	A number. Average outcome in treatment. Overrides `ate` if both specified.
`rho_WY`	A number in [-1,1]. Correlation between W and Y.
`rho_WZ`	A number in [-1,1]. Correlation between W and Z.
`args_to_fix`	A character vector. Names of arguments to be args_to_fix in design.

Details

Units are assigned to treatment using complete random assignment. Potential outcomes are normally distributed according to the mean and sd arguments.

See vignette online.

Value

A simple two-arm design with covariate W.

Author(s)

DeclareDesign Team

Examples

```
#Generate a simple two-arm design using default arguments
two_arm_covariate_design <- two_arm_covariate_designer()
# Design with no confounding but a prognostic covariate
prognostic <- two_arm_covariate_designer(N = 40, ate = .2, rho_WY = .9, h = .5)
## Not run:
diagnose_design(prognostic)

## End(Not run)
# Design with confounding
confounding <- two_arm_covariate_designer(N = 40, ate = 0, rho_WZ = .9, rho_WY = .9, h = .5)
## Not run:
diagnose_design(confounding, sims = 2000)

## End(Not run)

# Curse of power: A biased design may be more likely to mislead the larger it is
```

```
curses <- expand_design(two_arm_covariate_designer,
                        N = c(50, 500, 5000), ate = 0, rho_WZ = .2, rho_WY = .2)
## Not run:
diagnoses <- diagnose_design(curses)
subset(diagnoses$diagnosands_df, estimator == "No controls")[,c("N", "power")]

## End(Not run)
```

`two_arm_designer` *Create a one-level two-arm design*

Description

Builds a design with one treatment and one control arm. Treatment effects can be specified either by providing `control_mean` and `treatment_mean` or by specifying a `control_mean` and `ate`.

Usage

```
two_arm_designer(
  N = 100,
  assignment_prob = 0.5,
  control_mean = 0,
  control_sd = 1,
  ate = 1,
  treatment_mean = control_mean + ate,
  treatment_sd = control_sd,
  rho = 1,
  args_to_fix = NULL
)
```

Arguments

`N`	An integer. Sample size.
`assignment_prob`	A number in [0,1]. Probability of assignment to treatment.
`control_mean`	A number. Average outcome in control.
`control_sd`	A positive number. Standard deviation in control.
`ate`	A number. Average treatment effect.
`treatment_mean`	A number. Average outcome in treatment. Overrides `ate` if both specified.
`treatment_sd`	A nonnegative number. Standard deviation in treatment. By default equals `control_sd`.
`rho`	A number in [-1,1]. Correlation between treatment and control outcomes.
`args_to_fix`	A character vector. Names of arguments to be args_to_fix in design.

Details

Units are assigned to treatment using complete random assignment. Potential outcomes are normally distributed according to the mean and sd arguments.

Value

A simple two-arm design.

Author(s)

DeclareDesign Team

Examples

```
# Generate a simple two-arm design using default arguments
two_arm_design <- two_arm_designer()
```

`two_by_two_designer` *Create a two-by-two factorial design*

Description

Builds a two-by-two factorial design in which assignments to each factor are independent of each other.

Usage

```
two_by_two_designer(
  N = 100,
  prob_A = 0.5,
  prob_B = 0.5,
  weight_A = 0.5,
  weight_B = 0.5,
  outcome_means = rep(0, 4),
  mean_A0B0 = outcome_means[1],
  mean_A0B1 = outcome_means[2],
  mean_A1B0 = outcome_means[3],
  mean_A1B1 = outcome_means[4],
  sd_i = 1,
  outcome_sds = rep(0, 4),
  args_to_fix = NULL
)
```

Arguments

`N`	An integer. Size of sample.
`prob_A`	A number in [0,1]. Probability of assignment to treatment A.
`prob_B`	A number in [0,1]. Probability of assignment to treatment B.
`weight_A`	A number. Weight placed on A=1 condition in definition of "average effect of B" estimand.
`weight_B`	A number. Weight placed on B=1 condition in definition of "average effect of A" estimand.
`outcome_means`	A vector of length 4. Average outcome in each A,B condition, in order AB = 00, 01, 10, 11. Values overridden by mean_A0B0, mean_A0B1, mean_A1B0, mean_A1B1, if provided.
`mean_A0B0`	A number. Mean outcome in A=0, B=0 condition.
`mean_A0B1`	A number. Mean outcome in A=0, B=1 condition.
`mean_A1B0`	A number. Mean outcome in A=1, B=0 condition.
`mean_A1B1`	A number. Mean outcome in A=1, B=1 condition.
`sd_i`	A nonnegative scalar. Standard deviation of individual-level shock (common across arms).
`outcome_sds`	A nonnegative vector of length 4. Standard deviation of (additional) unit level shock in each condition, in order AB = 00, 01, 10, 11.
`args_to_fix`	A character vector. Names of arguments to be args_to_fix in design.

Details

Three types of estimand are declared. First, weighted averages of the average treatment effects of each treatment, given the two conditions of the other treatments. Second and third, the difference in treatment effects of each treatment, given the conditions of the other treatment.

Units are assigned to treatment using complete random assignment. Potential outcomes follow a normal distribution.

Treatment A is assigned first and then Treatment B within blocks defined by treatment A. Thus, if there are 6 units 3 are guaranteed to receive treatment A but the number receiving treatment B is stochastic.

See `multi_arm_designer` for a factorial design with non independent assignments.

Value

A two-by-two factorial design.

Author(s)

DeclareDesign Team

Examples

```
design <- two_by_two_designer(outcome_means = c(0,0,0,1))

# A design biased for the specified estimands:
design <- two_by_two_designer(outcome_means = c(0,0,0,1), prob_A = .8, prob_B = .2)
## Not run:
diagnose_design(design)

## End(Not run)

# A design with estimands that "match" the assignment:
design <- two_by_two_designer(outcome_means = c(0,0,0,1),
                                    prob_A = .8, prob_B = .2,
                                    weight_A = .8, weight_B = .2)
## Not run:
diagnose_design(design)

## End(Not run)

# Compare power with and without interactions, given same average effects in each arm
designs <- redesign(two_by_two_designer(),
                    outcome_means = list(c(0,0,0,1), c(0,.5,.5,1)))
## Not run:
diagnose_design(designs)

## End(Not run)
```

Index