Package 'spdesign'

January 16, 2024

Type Package

Title Designing Stated Preference Experiments

Version 0.0.3

Maintainer Erlend Dancke Sandorf <erlend.dancke.sandorf@nmbu.no>

Description Contemporary software commonly used to design stated preference experiments are expensive and the code is closed source. This is a free software package with an easy to use interface to make flexible stated preference experimental designs using state-of-the-art methods. For an overview of stated choice experimental design theory, see e.g., Rose, J. M. & Bliemer, M. C. J. (2014) in Hess S. & Daly. A. <doi:10.4337/9781781003152>. The package website can be accessed at <https://spdesign.edsandorf.me>. We acknowledge funding from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant INSPiRE (Grant agreement ID: 793163).

License CC BY-SA 4.0

Encoding UTF-8

URL https://spdesign.edsandorf.me,

https://github.com/edsandorf/spdesign

Depends R (>= 4.0.0), stringr

Imports cli, future, randtoolbox, matrixStats, dplyr, tibble

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

RoxygenNote 7.2.3

NeedsCompilation no

Author Erlend Dancke Sandorf [aut, cre],
Danny Campbell [aut]

Repository CRAN

Date/Publication 2024-01-16 10:20:02 UTC

R topics documented:

.onAttach *Print package startup message*

Description

The function is called when the package is loaded through library or require.

Usage

```
.onAttach(libname, pkgname)
```

Arguments

libname	Library name
pkgname	Package name

Value

Nothing

all_priors_and_levels_specified *Check whether all priors and attributes have specified levels*

Description

Check whether all priors and attributes have specified levels

Usage

```
all_priors_and_levels_specified(x)
```

Arguments

x	A list of utility expressions

Value

A boolean equal to ‘TRUE‘ if all are specified and ‘FALSE‘ if not

any_duplicates *Check whether any priors or attributes are specified with a value more than once*

Description

Check whether any priors or attributes are specified with a value more than once

Usage

```
any_duplicates(x)
```

Arguments

x	A list of utility expressions

Value

A boolean equal to 'TRUE' if specified more than once.

attribute_levels *Generic for getting the attributes and levels from the utility function*

Description

Generic for getting the attributes and levels from the utility function

Usage

```
attribute_levels(x)
```

Arguments

x	An object of class utility

Value

A named list of attribute levels

attribute_level_balance *Check whether we can achieve attribute level balance*

Description

Check whether we can achieve attribute level balance

Usage

```
attribute_level_balance(x, rows)
```

Arguments

x	A list of utility expressions
rows	The number of rows in the design

Value

A boolean equal to ‘TRUE‘ if attribute level balance can be achieved and ‘FALSE‘ otherwise

attribute_names *Generic for getting the attribute names*

Description

Generic for getting the attribute names

Usage

```
attribute_names(x)
```

Arguments

x	An object of class utility

Value

A character vector of attribute names

`block` *Block the design*

Description

The function will take an object of class 'spdesign' and add a blocking column to the design matrix. The function will use random permutations of the blocking column to find the column that minimizes correlation between the blocking column and the design columns. Specifically the target for the minimization procedure is the mean squared correlation.

Usage

```
block(x, blocks, target = 5e-04, max_iter = 1e+06)
```

Arguments

`x`	An object of class 'spdesign'
`blocks`	An integer giving the number of blocks. The number of blocks must be a multiple of the number of rows to ensure equal number of choices within a block.
`target`	A target value for the mean squared correlation. The default value is 0.0005. Setting the target to 0 forces the function to search all 'max_iter' blocking candidates
`max_iter`	The maximum number of candidates to consider before returning the best blocking candidate. The default value is 1000000.

Details

The function uses a random permutation so every time you run the function you will get a slightly different blocking column. You can set a seed prior to calling the function to always return the same blocking vector.

If you pass in a design that already contains a blocking column, then this blocking column will be replaced without warning.

Value

A modified 'spdesign' object where the design is replaced with the same design and a blocking column. In addition a correlation vector, number of iterations and the target value are returned as part of the modified 'spdesign' object.

calculate_a_error *A-error*

Description

Computes the A-error of the design, which is equal to the trace of the variance-covariance matrix over the number of parameters to be estimated

Usage

```
calculate_a_error(design_vcov)
```

Arguments

design_vcov	A variance-covariance matrix returned by derive_vcov or returned by an estimation routine. The matrix should be symmetrical and K-by-K

Value

A single error measure

calculate_c_error *C-error*

Description

Seeks to minimize the variance of the ratio of two parameters, for example, willingness-to-pay.

Usage

```
calculate_c_error(design_vcov, p, dudx, return_all)
```

Arguments

design_vcov	A variance-covariance matrix returned by derive_vcov or returned by an estimation routine. The matrix should be symmetrical and K-by-K
p	Prior values
dudx	A character string giving the name of the prior in the denominator. Must be specified when optimizing for 'c-error'
return_all	If 'TRUE' return a K or K-1 vector with parameter specific error measures. Default is 'FALSE'.

Value

A vector giving the variance of the ratio for each K-1 parameter or a single number with the sum of the variances used for optimization

calculate_d_error *D-error*

Description

Computes the D-error of the design, which is equal to the K-root of the determinant of the variance-covariance matrix.

Usage

```
calculate_d_error(design_vcov)
```

Arguments

design_vcov	A variance-covariance matrix returned by derive_vcov or returned by an estimation routine. The matrix should be symmetrical and K-by-K

Value

A single number

calculate_efficiency *Calculate efficiency*

Description

The function is called inside evaluate_design_candidate

Usage

```
calculate_efficiency(
  prior_values,
  design_env,
  model,
  dudx,
  return_all = FALSE,
  significance = 1.96
)
```

Arguments

prior_values	a list or vector of assumed priors
design_env	A design environment in which to evaluate the the function to derive the variance-covariance matrix.
model	A character string indicating the model to optimize the design for. Currently the only model programmed is the 'mnl' model and this is also set as the default.
dudx	A character string giving the name of the prior in the denominator. Must be specified when optimizing for 'c-error'
return_all	If 'TRUE' return a K or K-1 vector with parameter specific error measures. Default is 'FALSE'.
significance	A t-value corresponding to the desired level of significance. The default is significance at the 5 t-value of 1.96.

Value

A list with a named vector of efficiency criteria and the variance-covariance matrix

---

calculate_efficiency_criteria

Calculate efficiency criteria

Description

The function is a wrapper around calculate_a_error, calculate_c_error, calculate_d_error and calculate_s_error to provide a unified interface for calling and calculating efficiency criteria.

Usage

```
calculate_efficiency_criteria(
  design_vcov,
  p = NULL,
  dudx = NULL,
  return_all = FALSE,
  significance = 1.96,
  type
)
```

Arguments

design_vcov	A variance-covariance matrix returned by derive_vcov or returned by an estimation routine. The matrix should be symmetrical and K-by-K
p	Prior values
dudx	A character string giving the name of the prior in the denominator. Must be specified when optimizing for 'c-error'

return_all	If 'TRUE' return a K or K-1 vector with parameter specific error measures. Default is 'FALSE'.
significance	A t-value corresponding to the desired level of significance. The default is significance at the 5 t-value of 1.96.
type	A string indicating the type of efficiency criteria to calculate can be either: "a-error", "c-error", "d-error" or "s-error"

Details

The function is mainly used internally to evaluate and report on designs, but is exported to allow the user to use the function to calculate the efficiency criteria of the model once it has been run on their data.

Value

See individual efficiency criteria

References

Bliemer and Rose, 2009, Efficiency and sample size requirements for state choice experiments, Transportation Research Board Annual Meeting, Washington DC Scarpa and Rose, 2008, Designs efficiency for non-market valuation with choice modelling: How to measure it, what to report and why, Australian Journal of Agricultural and Resource Economics, 52(3):253-282 Bliemer and Rose, 2005a, Efficiency and sample size requirements for stated choice experiments, Report ITLS-WP-05-08, Institute for Transport and Logistics Studies, University of Sydney Kessels, R., Goos, P. and Vandebroek, M., 2006, A comparison of criteria to design efficient choice experiments, Journal of Marketing Research, 43(3):409-419

---

calculate_s_error *S-error*

Description

Calculates a "lower bound" sample size to obtain theoretically significant parameter estimates under the assumption that the priors are correct.

Usage

```
calculate_s_error(design_vcov, p, return_all, significance)
```

Arguments

design_vcov	A variance-covariance matrix returned by derive_vcov or returned by an estimation routine. The matrix should be symmetrical and K-by-K
p	Prior values
return_all	If 'TRUE' return a K or K-1 vector with parameter specific error measures. Default is 'FALSE'.
significance	A t-value corresponding to the desired level of significance. The default is significance at the 5 t-value of 1.96.

Value

A vector giving the "minimum" sample size for each parameter or a single number with the smallest sample size needed for all parameters to be theoretically significant.

---

clean_utility *Cleans the utility expression*

Description

The function cleans the utility expression by removing extra white spaces, removes brackets and other information to return a clean, easy-to-read expression.

Usage

```
clean_utility(x)
```

Arguments

x	An object of class utility

Details

We can also use the side-effect of the function on a list of utility expressions that do not contain brackets to return a an updated utility expression with alternative specific attribute names.

Warning: The function does not check if the utility expression *is* clean, which means that running the function multiple times will result in duplicate alternative names for the attributes. You need to pay particular attention to this fact when using the formula `update_utility` because this function calls `clean_utility`.

Value

A cleaned utility function as a list

---

coef.spdesign *Generic for extracting the vector of priors*

Description

Generic for extracting the vector of priors

Usage

```
## S3 method for class 'spdesign'
coef(object, ...)
```

Arguments

`object`	A model object of class 'spdesign'
`...`	Additional arguments passed to the function

Value

A vector of named priors used in the optimization

`contains_dummies` *Check whether the utility function contains dummy coded variables*

Description

We are splitting on all separators first before detecting whether we have dummy coded attributes to allow for people reusing the _dummy name for the attribute.

Usage

```
contains_dummies(string)
```

Arguments

`string`	A string or list of strings

Value

A boolean equal to 'TRUE' if the utility function contains dummy coded attributes and 'FALSE' otherwise

---

`cor` *Correlation*

Description

Calculate the correlation of the design. The function gets the design from the design object before passing it to `cor` from stats. This is a wrapper around `cor`.

Usage

```
cor(x, ...)
```

Arguments

`x`	A model object of class 'spdesign'
`...`	Additional parameters passed to the function

Details

Note that when your design includes constants, the function will print a warning because the standard deviation of a constant is 0.

Value

A matrix with correlations

`cycle` *Cycling of attribute levels*

Description

Cycles the attribute levels to create a new design candidate. "Cycling replaces all attribute levels in each choice situation at the time by replacing the first level with the second level, second level with the third etc. Since this change affects all columns, cycling can only be performed if all attributes have exactly the same sets of feasible levels, (e.g., where all variables are dummy coded)." (p. 253).

Usage

```
cycle(x)
```

Arguments

`x`	A vector of attribute levels

Details

This part of the RSC algorithm is rarely invoked.

Value

A cycled design candidate

References

Hensher, D. A., Rose, J. M. & Greene, W., 2005, Applied Choice Analysis, 2nd ed., Cambridge University Press

define_base_x_j *Define base x_j*

Description

Defines the base of the x_j list using the parsed utility expression, design_candidate and the base model matrix

Usage

```
define_base_x_j(utility, design_candidate)
```

Arguments

utility	A named list of utility functions. See the examples and the vignette for examples of how to define these correctly for different types of experimental designs.
design_candidate	The current design candidate under consideration

Value

A base list x_j with model matrices the lenght of J

define_x_j *Define x_j*

Description

Define x_j to use for the analytic derivatives of the variance-covariance matrix. x_j is derived based on the provided utility functions and design candidate using base model.matrix to automatically handle alternative specific attributes and interaction terms

Usage

```
define_x_j(utility, design_candidate)
```

Arguments

utility	A named list of utility functions. See the examples and the vignette for examples of how to define these correctly for different types of experimental designs.
design_candidate	The current design candidate under consideration

Details

We can extract the attribute names for each utility function to allow us to place the correct restrictions on the design candidate. Specifically, we restrict all levels of unavailable attributes to zero for alternatives where they do not feature. This is to ensure that we do not give weight when deriving the variance-covariance matrix using derive_vcov. Furthermore, the Xs are "sorted" using the order of the candidate set, which ensures that when we calculate the sum of the probabilities times X, the correct columns are added together. See derive_vcov.

Value

The list x_j

---

derive_vcov *Derive the variance covariance matrix of the design*

Description

The function is a wrapper around derive_vcov_mnl and derive_vcov_rpl and calculates the variance-covariance matrix of the specified model and design given the priors.

Usage

```
derive_vcov(design_env, model)
```

Arguments

design_env	An environment containing all the elements necessary to derive the variance-covariance matrix
model	A string indicating the model for which you wish to derive the variance covariance matrix. Can be either "mnl" or "rpl"

Value

The variance covariance matrix. If the Fisher information matrix is singular, then return NULL

derive_vcov_mnl *Derive the variance covariance matrix for the MNL model*

Description

The function takes no arguments and is evaluated in context!

Usage

```
derive_vcov_mnl()
```

Value

The variance co-variance matrix

derive_vcov_rpl *Derive the variance covariance matrix for the RPL model*

Description

The function takes no arguments and is evaluated in context!

Usage

```
derive_vcov_rpl()
```

Value

The variance co-variance matrix

digitize *Expand the sequence of integers*

Description

Equation 1 in Bhat (2003)

Usage

```
digitize(n_dim, primes, count, digit)
```

Arguments

n_dim	Number of dimensions
primes	A vector of prime numbers
count	A matrix
digit	A vector

References

Bhat, C. n_draws., 2003, Simulation Estimation of Mixed Discrete Choice Models Using Randomized and Scrambled Halton Sequences, Transportation Research Part B, 9, pp. 837-855

---

dummy_names *Find the position of the dummy coded attributes*

Description

The function will find the position of the dummy coded attributes in the candidate set (in the case of the Modified Federov or Random algorithms) or the design candidate (in the case of the RSC algorithm). This will let us know which columns to coerce to factors prior to defining x_j.

Usage

```
dummy_names(x)
```

Arguments

x	An object of class utility

Value

A boolean vector matching the expanded utility expression

---

evaluate_design_candidate *Evaluate the design candidate*

Description

The evaluation of the design candidate is independent of the optimization algorithm used.

Usage

```
evaluate_design_candidate(
  utility,
  design_candidate,
  prior_values,
  design_env,
  model,
  dudx,
  return_all,
  significance
)
```

Arguments

utility	A utility function
design_candidate	The current design candidate
prior_values	a list or vector of assumed priors
design_env	A design environment in which to evaluate the the function to derive the variance-covariance matrix.
model	A character string indicating the model to optimize the design for. Currently the only model programmed is the 'mnl' model and this is also set as the default.
dudx	A character string giving the name of the prior in the denominator. Must be specified when optimizing for 'c-error'
return_all	If 'TRUE' return a K or K-1 vector with parameter specific error measures. Default is 'FALSE'.
significance	A t-value corresponding to the desired level of significance. The default is significance at the 5 t-value of 1.96.

Value

A named vector with efficiency criteria of the current design candidate. If Bayesian prior_values are used, then it returns the average error.

---

exclude *Exclude rows from the candidate set*

Description

The function takes the list of exclusions and transforms them into an expression that is then parsed and evaluated to apply the exclusions to the supplied candidate set using standard subsetting routines.

Usage

```
exclude(candidate_set, exclusions)
```

Arguments

candidate_set	A matrix or data frame in the "wide" format containing all permitted combinations of attributes. The default is NULL. If no candidate set is provided, then the full factorial subject to specified exclusions will be used. This is passed in as an object and not a character string. The candidate set will be expanded to include zero columns to consider alternative specific attributes.
exclusions	A list of exclusions Often this list will be pulled directly from the list of options or it is a modified list of exclusions

Value

A restricted candidate set

---

expand_attribute_levels *Expand the list of attributes and levels to the "wide" format*

Description

Expands the attributes and levels to the wide format. The nested list is padded with zeros where alternative specific attributes are present to ensure that we can work with square matrices.

Usage

```
expand_attribute_levels(x)
```

Arguments

x	An object of class utility

Value

A named vector

---

extract_all_names *Extract all names*

Description

Extracts all parameter and attribute names from the utility function. This is a wrapper around `str_extract_all` with a specified boundary. The function also calls `remove_all_brackets` to ensure that if a word is used inside a square bracket, e.g. seq, it is not extracted.

Usage

```
extract_all_names(string, simplify = FALSE)
```

Arguments

string	A character string
simplify	If TRUE return as a vector. Default is FALSE.

Details

Note that we are not matching spaces nor the interaction operator I(). This is to avoid I being identified as its own (unspecified) attribute.

Value

A list or vector with all names

---

extract_attribute_names *Extract attribute names*

Description

Extracts attribute names. It is a wrapper around `extract_all_names` and `extract_param_names`.

Usage

```
extract_attribute_names(string, simplify = FALSE)
```

Arguments

string	A character string
simplify	If TRUE return as a vector. Default is FALSE.

Value

A Vector or string wtih attribute names

extract_distribution *Extract distributions*

Description

This function will locate and extract the the distributions for Bayesian priors and random parameters as specified in the design. The output is used to create the matrix of correct draws for priors and parameters.

Usage

```
extract_distribution(string, type)
```

Arguments

string	A single character string or list of character strings with a single or multiple utility functions
type	A string indicating the type: prior or param

Details

IMPORTANT: The function will silently drop duplicates.

Value

A named vector of priors or parameters where the type of distribution is given by a character letter: "normal", "lognormal", "uniform" or "triangular"

extract_level_occurrence *Extract the frequency of levels*

Description

The function extracts how many times each level of an attribute should occur within the design when attribute level balance is not enforced. Note that it extracts the parentheses AFTER the end of the square brackets. Specifying round brackets without the square brackets are syntactically invalid and therefore we want the code to fail in this case.

Usage

```
extract_level_occurrence(string, simplify = FALSE)
```

Arguments

string	A character string
simplify	If TRUE return as a vector. Default is FALSE.

extract_named_values *Extracts the named values of the utility function*

Description

The function extracts the named values of the supplied utility function.

Usage

```
extract_named_values(string)
```

Arguments

string	A character string

Value

A named list of parameter and attribute values. Each list element is named and can contain a single prior, a list with a mean and sd, or a vector with attribute levels

extract_param_distribution *Extract the parameter distribution*

Description

Extract the parameter distribution

Usage

```
extract_param_distribution(string)
```

Arguments

string	A single character string or list of character strings with a single or multiple utility functions

extract_param_names *Extract parameter names*

Description

Extracts all words starting with "b_". Leverages the fact that all parameters has to start with "b_".

Usage

```
extract_param_names(string, simplify = FALSE)
```

Arguments

string	A character string
simplify	If TRUE return as a vector. Default is FALSE.

Value

A list or vector with the parameter names.

extract_prior_distribution *Extract the prior distribution*

Description

Extract the prior distribution

Usage

```
extract_prior_distribution(string)
```

Arguments

string	A single character string or list of character strings with a single or multiple utility functions

extract_specified *Extract specified*

Description

Only extract parameters and attributes with specified priors and levels. This is very useful to test whether parameters or attributes are specified multiple times

Usage

```
extract_specified(string, simplify = FALSE)
```

Arguments

string	A character string
simplify	If TRUE return as a vector. Default is FALSE.

extract_unparsed_values *Extract unparsed named values of the utilitiy function*

Description

If the utility function contains parameters that are dummy coded, the dummy coding is handled here. By expanding the dummy coding prior to parsing we can directly consider Bayesian priors for each level.

Usage

```
extract_unparsed_values(string)
```

Arguments

string	A character string

Value

A named list of parameter and attribute values. Each list element is named and contains a numeric value or expression to be parsed

extract_values *Extract the value argument(s)*

Description

Extracts the value argument(s) of the supplied string. The value argument is defined as the characters between [] string.

Usage

```
extract_values(string, simplify = FALSE)
```

Arguments

string	A character string
simplify	If TRUE return as a vector. Default is FALSE.

Value

A vector or list with the extracted value arguments

federov *Find a design using a modified Federov algorithm*

Description

The modified Federov algorithm implemented here starts with a random design candidate and systematically swaps out rows of the design candidate to iteratively find better designs. The algorithm has the following steps and restrictions.

Usage

```
federov(
  design_object,
  model,
  efficiency_criteria,
  utility,
  prior_values,
  dudx,
  candidate_set,
  rows,
  control
)
```

Arguments

design_object	A list of class 'spdesign' created within the generate_design function
model	A character string indicating the model to optimize the design for. Currently the only model programmed is the 'mnl' model and this is also set as the default.
efficiency_criteria	A character string giving the efficiency criteria to optimize for. One of 'a-error', 'c-error', 'd-error' or 's-error'. No default is set and argument must be specified. Optimizing for multiple criteria is not yet implemented and will result in an error.
utility	A named list of utility functions. See the examples and the vignette for examples of how to define these correctly for different types of experimental designs.
prior_values	A list of priors
dudx	A character string giving the name of the prior in the denominator. Must be specified when optimizing for 'c-error'
candidate_set	A matrix or data frame in the "wide" format containing all permitted combinations of attributes. The default is NULL. If no candidate set is provided, then the full factorial subject to specified exclusions will be used. This is passed in as an object and not a character string. The candidate set will be expanded to include zero columns to consider alternative specific attributes.
rows	An integer giving the number of rows in the final design
control	A list of control options

Details

1) Create a random initial design and evaluate it. 2) Swap the first row of the design candidate with the first row of the candidate set. 3) If no better candidate is found, try the second row of the candidate set. Keep trying new rows of the candidate set until an improvement is found. 4) If a better candidate is found, then we try to swap out the next row in the design candidate with the first row of the candidate set. Keep repeating the previous step. 5) When all rows of the design candidate has been swapped once, reset the counter and work through the design candidate and candidate set again. 6) The algorithm terminates after a pre-determined number of iterations or when a pre-determined efficiency threshold has been found.

NOTE: I have not yet implemented a duplicate check! That is, I do not check whether the "same" choice rows are included but with the order of alternatives swapped. This can be achieved by further restricting the candidate set prior to searching for designs. That said, "identical" choice rows will not provide much additional information and should be excluded by default in the search process.

Value

A list of class 'spdesign'

fits_lvl_occurrences *Test whether a design candidate fits the constraints imposed by the level occurrences*

Description

Test whether a design candidate fits the constraints imposed by the level occurrences

Usage

```
fits_lvl_occurrences(utility, x, rows)
```

Arguments

utility	A named list of utility functions. See the examples and the vignette for examples of how to define these correctly for different types of experimental designs.
x	An object of class 'utility' or 'spdesign'
rows	Number of rows in the design

Value

A boolean equal to TRUE if attribute level balanced

full_factorial *Generate the full factorial*

Description

The function is a wrapper around `expand.grid` and generates the full factorial given the supplied attributes. The attributes can either be specified directly by the user or extracted from the list of utility functions using.

Usage

```
full_factorial(attrs)
```

Arguments

attrs	A named list of attributes and their levels

Details

The full factorial is often used as the starting point to generate a candidate set. Note that the full factorial will include unrealistic and completely dominated alternatives. It is therefore advised to use a subset of the full factorial as a candidate set. The user can call `full_factorial` and create a subset that is passed to `generate_design` using the ‘candidate_set‘ parameter, or supply a set of restrictions through the ‘restrictions‘ argument.

Value

A matrix containing the full factorial

Examples

```
opts <- list(
  level_balance = FALSE,
  tasks = 10
)
attrs <- list(
  a1 = 1:5,
  a2 = c(0, 1)
)

full_factorial(attrs)

V <- list(
  alt1 = "b_a1[0.1] * a1[1:5] + b_a2[-2] * a2[c(0, 1)]",
  alt2 = "b_a1      * a1      + b_a2     * a2"
)

attrs <- expand_attribute_levels(V)
full_factorial(attrs)
```

---

generate_design *Generate an efficient experimental design*

Description

The function generates efficient experimental designs. The function takes a set of indirect utility functions and generates efficient experimental designs assuming that people are maximizing utility.

Usage

```
generate_design(
  utility,
  rows,
  model = "mnl",
  efficiency_criteria = c("a-error", "c-error", "d-error", "s-error"),
  algorithm = c("federov", "rsc", "random"),
  draws = c("pseudo-random", "mlhs", "standard-halton", "scrambled-halton",
    "standard-sobol", "scrambled-sobol"),
  R = 100,
  dudx = NULL,
  candidate_set = NULL,
  exclusions = NULL,
 control = list(cores = 1, max_iter = 10000, max_relabel = 10000, max_no_improve =
```

```
    1e+05, efficiency_threshold = 0.1, sample_with_replacement = FALSE)
)
```

Arguments

utility	A named list of utility functions. See the examples and the vignette for examples of how to define these correctly for different types of experimental designs.
rows	An integer giving the number of rows in the final design
model	A character string indicating the model to optimize the design for. Currently the only model programmed is the 'mnl' model and this is also set as the default.
efficiency_criteria	A character string giving the efficiency criteria to optimize for. One of 'a-error', 'c-error', 'd-error' or 's-error'. No default is set and argument must be specified. Optimizing for multiple criteria is not yet implemented and will result in an error.
algorithm	A character string giving the optimization algorithm to use. No default is set and the argument must be specified to be one of 'rsc', 'federov' or 'random'.
draws	The type of draws to use with Bayesian priors. No default is set and must be specified even if you are not creating a Bayesian design. Can be one of "pseudo-random", "mlhs", "standard-halton", "scrambled-halton", "standard-sobol","scrambled-sobol".
R	An integer giving the number of draws to use. The default is 100.
dudx	A character string giving the name of the prior in the denominator. Must be specified when optimizing for 'c-error'
candidate_set	A matrix or data frame in the "wide" format containing all permitted combinations of attributes. The default is NULL. If no candidate set is provided, then the full factorial subject to specified exclusions will be used. This is passed in as an object and not a character string. The candidate set will be expanded to include zero columns to consider alternative specific attributes.
exclusions	A list of exclusions Often this list will be pulled directly from the list of options or it is a modified list of exclusions
control	A list of control options

Details

No assumptions are made with respect to default values and it is up to the user to specify optimization criteria, optmization routines, draws to use for Bayesian priors and more.

Value

An object of class 'spdesign'

generate_rsc_candidate

Generates a candidate for the RSC algorithm

Description

Creates a design candidate by assuming attribute level balance. Will work out the minimum level of times an attribute must occur for level balance. If level balance cannot be achieved the function will systematically add level occurrences to get as close as possible to attribute level balance.

Usage

```
generate_rsc_candidate(utility, rows)
```

Arguments

utility	A named list of utility functions. See the examples and the vignette for examples of how to define these correctly for different types of experimental designs.
rows	An integer giving the number of rows in the final design

Value

A data.frame with rows equal to the number of choice tasks and columns equal to the number of attributes in the 'wide' format

has_bayesian_prior

Tests whether the utility expression contains Bayesian priors

Description

This is particularly useful for flow-control

Usage

```
has_bayesian_prior(string)
```

Arguments

string	A string or list of strings

Value

A boolean equal to 'TRUE' if we have Bayesian priors

has_random_parameter *Tests whether the utility expression contains random parameters*

Description

This is particularly useful for flow-control

Usage

```
has_random_parameter(string)
```

Arguments

string	A string or list of strings

Value

A boolean equal to 'TRUE' if we have random parameters

is_balanced *Tests whether a utility function is balanced*

Description

Tests whether there is an equal number of opening and closing brackets in the utility functions.

Usage

```
is_balanced(string, open, close)
```

Arguments

string	A character string
open	An opening bracket ([or <
close	A closing bracket)] or >

Value

A boolean equal to 'TRUE' if the utility expression is balanced

lvl_occurrences *Attribute level occurrence lookup tables*

Description

Creates a list of lookup tables for attribute level occurrence.

Usage

```
lvl_occurrences(utility, rows, level_balance)
```

Arguments

utility	A named list of utility functions. See the examples and the vignette for examples of how to define these correctly for different types of experimental designs.
rows	An integer giving the number of rows in the final design
level_balance	Boolean equal to TRUE if level balance. This is not used

Value

A list the length of the expanced attribute levels. Each list element is a lookup table where the names of the table is the attribute level and the element the number of times the minimum number of times the level occurs.

make_draws *Make random draws*

Description

A common interface to creating a variety of random draws used to simulate the log likelihood function

Usage

```
make_draws(n_ind, n_draws, n_dim, seed, type)
```

Arguments

n_ind	Number of individuals in your sample
n_draws	Number of draws per respondent
n_dim	Number of dimensions
seed	A seed to change the scrambling of the sobol sequence.
type	A character string

Value

A matrix of dimensions n_ind*n_draws x n_dim of standard uniform draws

Examples

```
n_ind <- 10
n_draws <- 5
n_dim <- 3

draws <- make_draws(n_ind, n_draws, n_dim, seed = 10, "scrambled-sobol")
head(draws)

draws <- make_draws(n_ind, n_draws, n_dim, seed = 10, "scrambled-halton")
head(draws)
```

---

make_mlhs *Make Modified Latin Hypercube Draws*

Description

Make Modified Latin Hypercube Draws

Usage

```
make_mlhs(n_ind, n_draws, n_dim)
```

Arguments

n_ind	Number of individuals in your sample
n_draws	Number of draws per respondent
n_dim	Number of dimensions

References

Hess, S., Train, K. E. & Polak, J. W., 2006, On the use of a Modified Latin Hypercube Sampling (MLHS) method in the estimation of a Mixed Logit Model for vehicle choice, Transportation Research Part B, 40, pp. 147-163

make_pseudo_random *Make pseudo random draws*

Description

Wrapper for runif to create a common interface

Usage

```
make_pseudo_random(n_ind, n_draws, n_dim)
```

Arguments

n_ind	Number of individuals in your sample
n_draws	Number of draws per respondent
n_dim	Number of dimensions

make_scrambled_halton *Make scrambled Halton draws*

Description

A function for creating scrambled Halton draws. The code is a translation of the [GAUSS](http://www.caee.utexas.edu/prof/bl codes written by Professor Chandra Bhat. Note that the maximum number of dimensions for the scrambled Halton draws is limited to 16. This is because only permutations up to prime 16 are included in the permutation matrix. Extending to more than 16 dimensions can be achieved by including a different permutation matrix.

Usage

```
make_scrambled_halton(n_ind, n_draws, n_dim)
```

Arguments

n_ind	Number of individuals in your sample
n_draws	Number of draws per respondent
n_dim	Number of dimensions

Details

The permutations are based on the Braaten-Weller algorithm.

References

Bhat, C. n_draws., 2003, Simulation Estimation of Mixed Descrete Choice Models Using Randomized and Scrambled Halton Sequences, Transportation Research Part B, 9, pp. 837-855

make_scrambled_sobol *Make scrambled sobol draws*

Description

Wrapper function for sobol() from randtoolbox to create a common interface. Owen + Fazure_Tezuka Scrambling

Usage

```
make_scrambled_sobol(n_ind, n_draws, n_dim, seed = seed)
```

Arguments

n_ind	Number of individuals in your sample
n_draws	Number of draws per respondent
n_dim	Number of dimensions
seed	A seed to change the scrambling of the sobol sequence.

make_standard_halton *Wrapper for halton()*

Description

Wrapper function for halton() from randtoolbox to create a common interface

Usage

```
make_standard_halton(n_ind, n_draws, n_dim)
```

Arguments

n_ind	Number of individuals in your sample
n_draws	Number of draws per respondent
n_dim	Number of dimensions

make_standard_sobol *Make sobol draws*

Description

Wrapper function for sobol() from randtoolbox to create a common interface

Usage

```
make_standard_sobol(n_ind, n_draws, n_dim, seed = seed)
```

Arguments

n_ind	Number of individuals in your sample
n_draws	Number of draws per respondent
n_dim	Number of dimensions
seed	A seed to change the scrambling of the sobol sequence.

min_lvl_occurrence *Find minimum level occurrences*

Description

Find minimium level occurrences. This is useful to ensure/approximate attribute level balance in designs using the Modified Federov Algorithm or the Random design algorithms.

Usage

```
min_lvl_occurrence(x, rows)
```

Arguments

x	An object of class 'utility' or 'spdesign'
rows	Number of rows in the design

Value

A list of minimum level occurrences for the attribute levels

nlvls *Find the number of levels*

Description

Find the number of levels for each attribute

Usage

```
nlvls(x)
```

Arguments

x	An object of class 'utility' or 'spdesign'

Value

A list with the number of levels for each attribute

normal *Evaluating a distribution*

Description

The function returns its arguments as a named list. The function is used inside the utility functions. It is transformed to an expression using `parse` and evaluated using `eval`. This ensures that in the case of an RPL with Bayesian priors, recursion is handled automatically. This significantly simplifies translating the utility function to lists of parameters to use when optimizing the designs. It is also less error prone.

Usage

```
normal(mu, sigma)

normal_p(mu, sigma)

lognormal(mu, sigma)

lognormal_p(mu, sigma)

triangular(mu, sigma)

triangular_p(mu, sigma)

uniform(mu, sigma)

uniform_p(mu, sigma)
```

Arguments

mu	A parameter indicating the mean or location of the distribution depending on whether it is a normal, log-normal, triangular or uniform, or it can be another call to normal, lognormal, uniform or triangular if the model is an RPL with a Bayesian prior.
sigma	A parameter indicating the SD or spread of the distribution or it can be another call to normal, lognormal, uniform or triangular.

Value

A list of parameters

Functions

- normal(): The normal distribution
- normal_p(): The normal distribution when applied to a prior
- lognormal(): The log normal distribution
- lognormal_p(): The log-normal distribution when applied to a prior
- triangular(): The triangular distribution
- triangular_p(): The triangular distribution when applied to a prior
- uniform(): The uniform distribution
- uniform_p(): The uniform distribution when applied to a prior

---

occurrences *Extract or set attribute level occurrences*

Description

This function will set the range of attribute level occurrences equal to to the size of the design. This is equivalent to fully letting go of attribute level balance. Letting go of attribute level balance is the default behavior for the Modified Federov algorithm and the Random algorithm.

Usage

```
occurrences(x, rows)
```

Arguments

x	An object of class 'utility' or 'spdesign'
rows	Number of rows in the design

Details

If restrictions are placed on attribute level occurrence in the utility function, then this function will extract these and add them to the output.

Notice that specifying restrictions in the utility function only matters for the Modified Federov and Random algorithms and will in general result in a less efficient design.

Value

A named list of lists where the outer list is for the attributes and the inner list, the levels of each attribute and the number or range of times they can occur

---

prepare_priors *Prepare the list of priors*

Description

Prepare the list of priors

Usage

```
prepare_priors(utility, draws, R)
```

Arguments

utility	A named list of utility functions. See the examples and the vignette for examples of how to define these correctly for different types of experimental designs.
draws	The type of draws to use with Bayesian priors. No default is set and must be specified even if you are not creating a Bayesian design. Can be one of "pseudo-random", "mlhs", "standard-halton", "scrambled-halton", "standard-sobol","scrambled-sobol".
R	An integer giving the number of draws to use. The default is 100.

Value

A list of priors

`print.spdesign` *A generic function for printing an 'spdesign' object*

Description

A generic function for printing an 'spdesign' object

Usage

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

Arguments

`x`	A model object of class 'spdesign'
`...`	Additional parameters passed to the function

Value

No return value. Prints the 'spdesign' object.

`print_efficiency_criteria` *Creates a printable version of the efficiency criteria*

Description

The function prints a string of efficiency criteria to the console and highlights the color of the considered efficiency criteria. Effectively it is a wrapper around multiple calls to `cat`.

Usage

```
print_efficiency_criteria(
  iter,
  values,
  criteria,
  digits = 4,
  padding = 10,
  efficiency_criteria
)
```

Arguments

iter	An integer giving the iteration of the loop
values	The value of the efficiency criteria obtained by calculate_efficiency_criteria
criteria	A character string with the name of the efficiency criteria. See manual for valid values
digits	The nubmer of digits to round the printed value to. The default is 4.
padding	An integer specifying the padding of each column element. Default value is 10.
efficiency_criteria	The criteria that we optimize over

Value

A character string.

---

print_initial_header *Prints the initial header for the table of results*

Description

The function prints the initial header for the console output and colors in the criteria used for optimization. Effectively, the function makes multiple calls to cat.

Usage

```
print_initial_header(efficiency_criteria, padding = 10, width = 80)
```

Arguments

efficiency_criteria	The criteria that we optimize over
padding	An integer specifying the padding of each column element. Default value is 10.
width	An integer giving the width of the horizontal rules. Default value is 80

Value

Noting

print_iteration_information *Prints iteration information*

Description

Prints iteration information every time a better design is found. The function wraps around `print_initial_header` and `print_efficiency_criteria`. This reduces the number of if-statements and function calls within `generate_design` in an attempt simplify code maintenance.

Usage

```
print_iteration_information(
  iter,
  values,
  criteria,
  digits = 4,
  padding = 10,
  width = 80,
  efficiency_criteria
)
```

Arguments

`iter`	An integer giving the iteration of the loop
`values`	The value of the efficiency criteria obtained by `calculate_efficiency_criteria`
`criteria`	A character string with the name of the efficiency criteria. See manual for valid values
`digits`	The nubmer of digits to round the printed value to. The default is 4.
`padding`	An integer specifying the padding of each column element. Default value is 10.
`width`	An integer giving the width of the horizontal rules. Default value is 80
`efficiency_criteria`	The criteria that we optimize over

Value

Nothing

priors *Generic for extracting the vector of priors*

Description

Generic for extracting the vector of priors

Usage

```
priors(x)
```

Arguments

x	An object of class 'utility' or 'spdesign'

Value

A list of named priors used in the optimization

probabilities *Calculate the probabilities of the design*

Description

Will take the design object and calculate the probabilities of each alternative and choice tasks.

Usage

```
probabilities(x)
```

Arguments

x	An 'spdesign' object.

Details

Using Bayesian priors the average across the prior distribution will be used.

Using the specific type of model, either the MNL or RPL probs will be returned.

Value

A matrix of probabilities for each alternative and choice task.

probabilities_mnl *Calculate the MNL probabilities*

Description

Calculate the MNL probabilities

Usage

```
probabilities_mnl(x)
```

Arguments

x	An 'spdesign' object.

Value

A matrix of probabilities for each alternative and choice task. With Bayesian priors the return is the average probabilites over the prior distribution

radical_inverse *Compute the radical inverse*

Description

Equation 2 in Bhat (2003)

Usage

```
radical_inverse(n_dim, primes, count, digit, perms)
```

Arguments

n_dim	Number of dimensions
primes	A vector of prime numbers
count	A matrix
digit	A vector
perms	A matrix of the permutations. Defaults to a set of Braaten-Weller permutations.

References

Bhat, C. n_draws., 2003, Simulation Estimation of Mixed Descrete Choice Models Using Randomized and Scrambled Halton Sequences, Transportation Research Part B, 9, pp. 837-855

`random` *Make a random design*

Description

Generates a random design by sampling from the candidate set each update of the algorithm.

Usage

```
random(
  design_object,
  model,
  efficiency_criteria,
  utility,
  prior_values,
  dudx,
  candidate_set,
  rows,
  control
)
```

Arguments

`design_object`	A list of class 'spdesign' created within the `generate_design` function
`model`	A character string indicating the model to optimize the design for. Currently the only model programmed is the 'mnl' model and this is also set as the default.
`efficiency_criteria`	A character string giving the efficiency criteria to optimize for. One of 'a-error', 'c-error', 'd-error' or 's-error'. No default is set and argument must be specified. Optimizing for multiple criteria is not yet implemented and will result in an error.
`utility`	A named list of utility functions. See the examples and the vignette for examples of how to define these correctly for different types of experimental designs.
`prior_values`	A list of priors
`dudx`	A character string giving the name of the prior in the denominator. Must be specified when optimizing for 'c-error'
`candidate_set`	A matrix or data frame in the "wide" format containing all permitted combinations of attributes. The default is NULL. If no candidate set is provided, then the full factorial subject to specified exclusions will be used. This is passed in as an object and not a character string. The candidate set will be expanded to include zero columns to consider alternative specific attributes.
`rows`	An integer giving the number of rows in the final design
`control`	A list of control options

Details

With no restrictions placed, this type of design will only consider efficiency. There is no guarantee that you will achieve attribute level balance, nor that all attribute levels will be present. More efficient designs tend to have more extreme trade-offs.

Value

A list of class 'spdesign'

`random_design_candidate`

Create a random design_object candidate

Description

Sample from the candidate set to create a random design_object.

Usage

```
random_design_candidate(utility, candidate_set, rows, sample_with_replacement)
```

Arguments

`utility`	A named list of utility functions. See the examples and the vignette for examples of how to define these correctly for different types of experimental designs.
`candidate_set`	A matrix or data frame in the "wide" format containing all permitted combinations of attributes. The default is NULL. If no candidate set is provided, then the full factorial subject to specified exclusions will be used. This is passed in as an object and not a character string. The candidate set will be expanded to include zero columns to consider alternative specific attributes.
`rows`	An integer giving the number of rows in the final design
`sample_with_replacement`	A boolean equal to TRUE if we sample from the candidate set with replacement. The default is FALSE

relabel *Relabeling of attribute levels*

Description

Relabels the attribute levels to create a new design candidate. For example, if the column contains the levels (1, 2, 1, 3, 2, 3) and 1 and 3 are relabeled, then the column becomes (3, 2, 3, 1, 2, 1), i.e. 1 becomes 3 and 3 becomes 1.

Usage

```
relabel(x)
```

Arguments

x	A vector of attribute levels

Details

Will randomly sample 2 attribute levels that will be relabeled and the relabeling is done independently for each column, which implies that the same attribute will be relabeled differently depending on which alternative it belongs to.

References

Hensher, D. A., Rose, J. M. & Greene, W., 2005, Applied Choice Analysis, 2nd ed., Cambridge University Press

remove_all_brackets *Removes all brackets*

Description

Takes a string as input and removes everything between square and round brackets. The function wraps around remove_square_brackets and remove_round_brackets. To avoid problems, we first remove square brackets.

Usage

```
remove_all_brackets(string)
```

Arguments

string	A character string

Value

A string without brackets

remove_prior *Removes the parameter from the utility string*

Description

Removes the parameter from the utility string

Usage

```
remove_prior(prior, string)
```

Arguments

prior	A string with the parameter name
string	A string to remove param from

remove_round_brackets *Remove round bracket*

Description

Removes everything between (and including) round brackets. We negating matches with I(), since this is R's interaction operator.

Usage

```
remove_round_brackets(string)
```

Arguments

string	A character string

Details

(?<!I) - A negative lookbehind for I

`remove_square_brackets` *Remove square bracket*

Description

Removes everything between (and including) square brackets

Usage

```
remove_square_brackets(string)
```

Arguments

`string`	A character string

`remove_whitespace` *Remove all white spaces*

Description

Takes a string as an input and removes all whitespaces in the string

Usage

```
remove_whitespace(string)
```

Arguments

`string`	A character string

Value

A character vector with no white spaces

rep_cols *Repeat columns*

Description

Repeats each column of the matrix or data frame 'x' a number of times equal to 'times'.

Usage

```
rep_cols(x, times)
```

Arguments

x	A matrix or data frame
times	An integer indicating the number of times to repeat the row/column

Value

A matrix or data.frame depending on the type of the input

Examples

```
test_matrix <- matrix(runif(12), 4)
rep_cols(test_matrix, 2)
```

rep_rows *Repeat rows*

Description

Repeats each row in the matrix or data frame 'x' a number of times equal to 'times'.

Usage

```
rep_rows(x, times)
```

Arguments

x	A matrix or data frame
times	An integer indicating the number of times to repeat the row/column

Value

A matrix or data.frame depending on type of the input

Examples

```
test_matrix <- matrix(runif(12), 4)
rep_rows(test_matrix, 2)
```

`rsc` *Make a design candidate based on the rsc algorithm*

Description

Depending on the setting the function calls a combination of `relabel`, `swap` and `cycle` to create new design candidates. The code is intentionally written modular to allow for all special cases of the algorithm.

Usage

```
rsc(
  design_object,
  model,
  efficiency_criteria,
  utility,
  prior_values,
  dudx,
  candidate_set,
  rows,
  control
)
```

Arguments

`design_object`	A list of class 'spdesign' created within the `generate_design` function
`model`	A character string indicating the model to optimize the design for. Currently the only model programmed is the 'mnl' model and this is also set as the default.
`efficiency_criteria`	A character string giving the efficiency criteria to optimize for. One of 'a-error', 'c-error', 'd-error' or 's-error'. No default is set and argument must be specified. Optimizing for multiple criteria is not yet implemented and will result in an error.
`utility`	A named list of utility functions. See the examples and the vignette for examples of how to define these correctly for different types of experimental designs.
`prior_values`	A list of priors
`dudx`	A character string giving the name of the prior in the denominator. Must be specified when optimizing for 'c-error'

candidate_set	A matrix or data frame in the "wide" format containing all permitted combinations of attributes. The default is NULL. If no candidate set is provided, then the full factorial subject to specified exclusions will be used. This is passed in as an object and not a character string. The candidate set will be expanded to include zero columns to consider alternative specific attributes.
rows	An integer giving the number of rows in the final design
control	A list of control options

`set_default_level_occurrence` *Sets the default level occurrence in an attribute level balanced design*

Description

The function sets the default level occurrence of an attribute when a design is restricted to be attribute level balanced. If the design cannot be attribute level balanced, then the restriction will be relaxed for each attribute failing to meet this criteria. Specifically, the code will impose a minimum range of how often an attribute level can occur. This will secure that the design is near attribute level balanced. In this case a warning is issued.

Usage

```
set_default_level_occurrence(n_lvls, rows)
```

Arguments

n_lvls	An integer giving the number of levels for the considered attribute
rows	Number of rows in the design

Value

A named list of lists where the top level gives the attribute and the lower level gives the times or range each attribute level should occur in the design

---

`set_default_options` *Validate design opt*

Description

The function takes the list of design options and adds default values where none are specified. This function is exported, but is not intended to be called by the user of the package. The function is called from within `generate_design` to populate the list with sensible defaults

Usage

```
set_default_options(opts_input)
```

Arguments

opts_input	A list of user supplied design options

Value

A list of design options populated by sensible default values

---

shuffle *Shuffle the order of points in the unit interval.*

Description

Shuffle the order of points in the unit interval.

Usage

```
shuffle(x)
```

Arguments

x	A vector

---

summary.spdesign *Create a summary of the experimental design*

Description

Create a summary of the experimental design

Usage

```
## S3 method for class 'spdesign'
summary(object, ...)
```

Arguments

object	A model object of class 'spdesign'
...	Additional arguments passed to the function

Value

No return value. Prints a summary of the 'spdesign' object to the console

swap *Swapping of attribute*

Description

Swaps the order of the attributes to create a new design candidate. For example, if the attributes in the first and fourth choice situation (row) are swapped, then (1, 2, 1, 3, 2, 3) becomes(3, 2, 1, 1, 2, 3).

Usage

```
swap(x)
```

Arguments

x	A vector of attribute levels

Details

The algorithm randomly samples 2 row positions that are swapped and the swaps are independent across attributes and alternatives

References

Hensher, D. A., Rose, J. M. & Greene, W., 2005, Applied Choice Analysis, 2nd ed., Cambridge University Press

too_small *Check if the design is too small*

Description

Uses the formula of T * (J - 1) to check if the design is large enough to identify the parameters of the utility function.

Usage

```
too_small(x, rows)
```

Arguments

x	A list of utility expressions
rows	The number of rows in the design

Value

A boolean equal to ‘TRUE‘ if the design is too small

transform_distribution *Transform distribution*

Description

Transform distribution

Usage

```
transform_distribution(mu, sigma, eta, type)
```

Arguments

mu	A value for the mean of the distribution
sigma	A value for the standard deviation of the distribution
eta	A numeric standard uniform vector
type	The type of distribution

Value

A vector with the transformed distribution given the parameters

transform_lognormal *Transform to the lognormal distribution*

Description

Transform to the lognormal distribution

Usage

```
transform_lognormal(mu, sigma, eta)
```

Arguments

mu	A value for the mean of the distribution
sigma	A value for the standard deviation of the distribution
eta	A numeric standard uniform vector

transform_normal *Transform to the normal distribution*

Description

Transform to the normal distribution

Usage

```
transform_normal(mu, sigma, eta)
```

Arguments

mu	A value for the mean of the distribution
sigma	A value for the standard deviation of the distribution
eta	A numeric standard uniform vector

transform_triangular *Transform to the triangular distribution*

Description

Transform to the triangular distribution

Usage

```
transform_triangular(mu, sigma, eta)
```

Arguments

mu	A value for the mean of the distribution
sigma	A value for the standard deviation of the distribution
eta	A numeric standard uniform vector

transform_uniform *Transform to the uniform distribution*

Description

Transform to the uniform distribution

Usage

```
transform_uniform(mu, sigma, eta)
```

Arguments

mu	A value for the mean of the distribution
sigma	A value for the standard deviation of the distribution
eta	A numeric standard uniform vector

update_utility *Update the utility function*

Description

Updates the utility function to consider dummy coded attributes. It will expand the dummy-coding to K-1 dropping the lowest level. This is consistent with standard practice.

Usage

```
update_utility(x)
```

Arguments

x	An object of class utility

Details

The function is called prior to evaluating designs if dummy-coded attributes are present in the utility function. This is because the utility function is evaluated in the context of the design environment and must be added there

Important to note about the naming of the expanded priors and attributes: The names for the attributes will be attached with the level of the factor, whereas the prior will be named corresponding to the level, e.g., 2, 3, 4. This is simply the result of the difference between how it's extracted from the utility functions and how model.matrix creates names.

Value

An updated cleaned utility expression

utility_formula *Create formulas from the utility functions*

Description

Create formulas from the utility functions such that we can create correct model matrices.

Usage

```
utility_formula(x)
```

Arguments

x	An object of class utility

Details

Note that this function should be used on a cleaned utility expression and **not** an updated utility expression. This is because we are converting dummy coded attributes to factors prior to calling `model.matrix`. This ensures that dummy coded attributes are correctly returned with the model matrix.

Value

A list of formula expressions for the utility functions

vcov.spdesign *Extract the variance co-variance matrix*

Description

A generic method for extracting the variance covariance matrix from a design object

Usage

```
## S3 method for class 'spdesign'
vcov(object, ...)
```

Arguments

object	A model object of class 'spdesign'
...	Additional arguments passed to the function

Value

A matrix with row- and column names equal to the parameter names

Index