Package: spdesign (via r-universe)

October 30, 2026

Type Package

Title Designing Stated Preference Experiments

Version 0.0.8

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). The package features in Mariel et al. (2025) Environmental Valuation with Discrete Choice Experiments in R. (<doi:10.1007/978-3-031-89338-4>).

License CC BY-SA 4.0

Encoding UTF-8

URL https://spdesign.edsandorf.me, https://github.com/edsandorf/spdesign

Depends R (>= 4.1.0), stringr

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

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr, rmarkdown

Config/roxygen2/version 8.0.0

BugReports https://github.com/edsandorf/spdesign/issues

NeedsCompilation no

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

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

Date/Publication 2026-10-30 14:28:35 UTC

RemoteUrl https://github.com/cran/spdesign

RemoteRef HEAD

RemoteSha 4da00c1ce3e54c1c25618384522e13b96ea39799

Contents

.onAttach . . . 4
align_x_j . . . 5
all_priors_and_levels_specified . . . 5
any_duplicates . . . 6
as_whole_word . . . 6
attribute_level_balance . . . 7
attribute_levels . . . 7
attribute_names . . . 8
block . . . 8
build_balanced_design_candidate . . . 9
build_candidate_set . . . 10
calculate_a_error . . . 11
calculate_c_error . . . 11
calculate_d_error . . . 12
calculate_efficiency . . . 13
calculate_efficiency_criteria . . . 14
calculate_s_error . . . 15
clean_utility . . . 16
coef.spdesign . . . 16
combine_profiles . . . 17
contains_dummies . . . 17
cor . . . 18
cycle . . . 18
define_profiles . . . 19
define_x_j . . . 19
derive_vcov . . . 20
derive_vcov_mnl . . . 21
derive_vcov_rpl . . . 21
digitize . . . 21
draw_design_candidate . . . 22
dummy_names . . . 23
evaluate_design_candidate . . . 23
exclude . . . 24
expand_attribute_levels . . . 25
extract_all_names . . . 25
extract_attribute_names . . . 26
extract_distribution . . . 26
extract_level_occurrence . . . 27

extract_named_values . . . 27
extract_param_distribution . . . 28
extract_param_names . . . 28
extract_prior_distribution . . . 29
extract_specified . . . 29
extract_unparsed_values . . . 30
extract_values . . . 30
federov . . . 31
full_factorial . . . 33
generate_design . . . 33
has_bayesian_prior . . . 35
has_random_parameter . . . 36
invalid_dummy_coding . . . 36
is_balanced . . . 37
level_balance . . . 37
level_occurrences_specified . . . 38
lvl_occurrences . . . 38
lvl_violation . . . 39
make_draws . . . 39
make_mlhs . . . 40
make_pseudo_random . . . 41
make_scrambled_halton . . . 41
make_scrambled_sobol . . . 42
make_standard_halton . . . 42
make_standard_sobol . . . 43
min_lvl_occurrence . . . 43
nlvls . . . 44
normal . . . 44
occurrences . . . 45
pair_param_terms . . . 46
prepare_priors . . . 46
print.spdesign . . . 47
print_efficiency_criteria . . . 48
print_initial_header . . . 48
print_iteration_information . . . 49
priors . . . 50
probabilities . . . 50
probabilities_mnl . . . 51
radical_inverse . . . 51
random . . . 52
relabel . . . 53
remove_all_brackets . . . 54
remove_round_brackets . . . 54
remove_square_brackets . . . 55
remove_whitespace . . . 55
rep_cols . . . 56
rep_rows . . . 56
rsc . . . 57

set_default_level_occurrence 58
set_default_options 59
shuffle 59
summary.spdesign 60
swap 60
too_small 61
transform_distribution 61
transform_lognormal 62
transform_normal 62
transform_triangular 63
transform_uniform 63
unlisted_levels 64
update_utility 64
utility_formula 65
vcov.spdesign 66

Index **67**

`.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

align_x_j *Align x_j with the priors*

Description

Renames the columns of x_j from terms to parameters, so that a generic parameter is a single column even if its attribute has a different name in each alternative. Parameters that do not enter the utility function of an alternative are zero for that alternative. The columns are ordered as the priors, which ensures that the variance-covariance matrix derived by `derive_vcov` matches the priors.

Usage

```
align_x_j(x_j, terms, names_priors)
```

Arguments

`x_j`	A list of model matrices returned by `define_x_j`
`terms`	A list of terms and their parameters returned by `pair_param_terms`
`names_priors`	The names of the priors

Value

The list x_j with one column per prior in the order of the priors

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.

as_whole_word *Match names as whole words*

Description

Creates regular expressions that match names as whole words, so that e.g. x1 does not match x10, b_x1 or alt1_x1. Names in the utility functions only contain letters, digits and underscores.

Usage

```
as_whole_word(name)
```

Arguments

name	A character vector of names

Value

A character vector of regular expressions

`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_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_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.

`build_balanced_design_candidate`

Build a level balanced design candidate

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

```
build_balanced_design_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

`build_candidate_set` *Build the candidate set*

Description

Builds the candidate set one alternative at a time instead of from the full factorial of all alternatives. This uses far less memory and avoids choice tasks that are the same apart from the order of the alternatives.

Usage

```
build_candidate_set(utility, exclusions = list(), allow_reversed_pairs = 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.
`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
`allow_reversed_pairs`	If TRUE, include the profiles of exchangeable alternatives in every order, e.g. both A versus B and B versus A. This also applies to sets of three or more exchangeable alternatives. The default is FALSE.

Details

The profiles of each alternative are the full factorial of its attribute levels. Exclusions that only refer to a single alternative are applied to its profiles before the alternatives are combined. Alternatives are exchangeable if their utility functions and profiles are identical apart from the name of the alternative, e.g. two unlabelled alternatives. By default, each choice task contains a set of different profiles for the exchangeable alternatives only once, and the order of the exchangeable alternatives is randomised in each row to ensure that no alternative systematically gets the same profiles. Labelled alternatives are combined with all profiles of the other alternatives, as in the full factorial. Finally, exclusions that refer to more than one alternative are applied.

Because the order of exchangeable alternatives is random, exclusions across exchangeable alternatives must be specified in both directions, e.g. that alt1 dominates alt2 and that alt2 dominates alt1.

With 'allow_reversed_pairs = TRUE' and no exclusions, the candidate set is the full factorial without choice tasks where exchangeable alternatives have the same profile. To create a candidate set with attribute levels that are different from those in the utility functions, use `expand.grid`.

Value

A data frame with the candidate set

Examples

```
utility <- list(
  alt1 = "b_x1[0.1] * x1[1:3] + b_x2[-0.2] * x2[c(0, 1)]",
  alt2 = "b_x1      * x1      + b_x2       * x2"
)

build_candidate_set(utility, exclusions = list("alt1_x1 == 1 & alt1_x2 == 0"))
```

---

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,
  dudx,
  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

`combine_profiles` *Combine the profiles of a group of exchangeable alternatives*

Description

Combine the profiles of a group of exchangeable alternatives

Usage

```
combine_profiles(n, k, allow_reversed_pairs)
```

Arguments

`n`	The number of profiles of each alternative in the group
`k`	The number of alternatives in the group
`allow_reversed_pairs`	If TRUE, include the profiles of exchangeable alternatives in every order, e.g. both A versus B and B versus A. This also applies to sets of three or more exchangeable alternatives. The default is FALSE.

Value

A matrix with one row per combination and one column per alternative, giving the rows of the profiles to combine

---

`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

This function is deprecated and will be removed in a future version. Use `cor(design$design)` to calculate the correlation of a design.

Usage

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

Arguments

x	An object of class 'spdesign', or any input to cor
...	Additional arguments passed to cor

Details

For an object of class 'spdesign', the function calculates the correlation of the design. For any other object, it calls cor, so that it does not mask the function from stats.

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_profiles` *Define the profiles of each alternative*

Description

Defines the profiles of each alternative as the full factorial of its attribute levels, with the exclusions that only refer to that alternative applied, and finds the groups of exchangeable alternatives. See `build_candidate_set`.

Usage

```
define_profiles(utility, exclusions = list())
```

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.
`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 list with the profiles of each alternative, the groups of exchangeable alternatives and the exclusions across alternatives

---

`define_x_j` *Define x_j*

Description

Defines x_j, the list of attribute matrices with one matrix per alternative. Each matrix has one column per term in the utility function of that alternative: attributes, expanded dummy-coded attributes and, if requested, interaction terms. The columns are named after the terms and keep the order of `model.matrix`.

Usage

```
define_x_j(
  utility,
  design_candidate,
  terms = pair_param_terms(update_utility(utility)),
  interactions = TRUE
)
```

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
terms	A list of terms and their parameters returned by pair_param_terms. The default pairs the terms of the updated utility functions.
interactions	If TRUE, include interaction terms. The default is TRUE.

Value

A list of model matrices, one per alternative

---

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

---

draw_design_candidate *Draw a design candidate from the candidate set*

Description

Sample from the candidate set to create a random design_object. If level occurrences are specified, single rows are swapped with random rows from the candidate set until the design_object candidate satisfies them. A swap is kept if it does not move the design_object candidate further away from the level occurrences.

Usage

```
draw_design_candidate(
  utility,
  candidate_set,
  rows,
  print_counter = FALSE,
  max_attempts = 1e+05
)
```

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 it is built from the utility functions subject to specified exclusions with build_candidate_set. 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
print_counter	A boolean equal to TRUE if we print the number of attempts to find a design candidate every 1000th attempt. The default is FALSE
max_attempts	The maximum number of swaps to try before stopping with an error. The default is 100000.

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

Each exclusion is evaluated with the columns of the candidate set as variables, and the rows where it is TRUE are removed.

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 it is built from the utility functions subject to specified exclusions with build_candidate_set. 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

A name is a word that starts with a letter and is not followed by an opening round bracket. This avoids extracting numbers, e.g. the 2 in I(x1^2), and functions such as the interaction operator I() as attributes.

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. It is a first-improvement variant: a swap is kept as soon as it improves the design, instead of searching the entire candidate set for the best swap. A swap that does not improve the design is discarded. The algorithm has the following steps and restrictions.

Usage

```
federov(
  design_object,
  model,
  efficiency_criteria,
  utility,
  prior_values,
  dudx,
  candidate_set,
  rows,
  save_designs,
  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 it is built from the utility functions subject to specified exclusions with build_candidate_set. 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
`save_designs`	A boolean indicating whether to save up to 10 intermediate designs. The default value is FALSE.
`control`	A list of control options. Set 'allow_reversed_pairs = TRUE' to include the profiles of exchangeable alternatives in every order when the candidate set is built. See `build_candidate_set`.

Details

1) Create a random initial design and evaluate it. If level occurrences are specified, the initial design satisfies them. 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 next row of the candidate set. Keep trying new rows of the candidate set until an improvement is found. NOTE: A swap is skipped without being evaluated if it would include the same row multiple times or violate the level occurrences. 4) If a better candidate is found, then we try to swap out the next row in the design candidate, continuing with the next row of the candidate set. If no row of the candidate set improves the current row of the design candidate, move on to the next row. 5) When the end of the design candidate or the candidate set is reached, continue from the first row. This way every row of the candidate set is tried equally often. 6) If a full pass over all rows of the design candidate and the candidate set finds no improvement, the design candidate is a local optimum. Store it and restart from step 1 with a new random design candidate. 7) The algorithm terminates after a pre-determined number of iterations or when a pre-determined efficiency threshold has been found. Pressing Esc (or Ctrl + C) stops the search and returns the best design found so far.

The best design found across all runs is returned as the design. The best design of each run, including the one that is still running when the search stops, is stored in the list element 'runs' together with its efficiency criteria.

For very large candidate sets, the maximum number of iterations decides how deep into the candidate set the search goes. Every swap moves one row further through the candidate set, but only evaluated swaps count towards the maximum number of iterations. A search that stops after fewer iterations than there are rows in the candidate set may not have tried all of them.

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'

full_factorial *Generate the full factorial*

Description

This function is deprecated and will be removed in a future version. Use `build_candidate_set` to build a candidate set from the utility functions, or `expand.grid` to create the full factorial of a list of attributes with levels that are different from those in the utility functions.

Usage

```
full_factorial(attrs)
```

Arguments

`attrs`	A named list of attributes and their levels

Details

The function is a wrapper around `expand.grid` and generates the full factorial given the supplied attributes.

Value

A data frame containing the full factorial

Examples

```
attrs <- list(
  a1 = 1:5,
  a2 = c(0, 1)
)

# Instead of full_factorial(attrs)
expand.grid(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,
  save_designs = FALSE,
  control = list(cores = 1, max_iter = 10000, max_relabel = 500, max_swap = 500,
   max_no_improve = 1000, efficiency_threshold = 1e-06, allow_reversed_pairs = 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 it is built from the utility functions subject to specified exclusions with build_candidate_set. 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
save_designs	A boolean indicating whether to save up to 10 intermediate designs. The default value is FALSE.
control	A list of control options. Set 'allow_reversed_pairs = TRUE' to include the profiles of exchangeable alternatives in every order when the candidate set is built. See build_candidate_set.

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'. For the 'federov' and 'rsc' algorithms, the list element 'runs' holds the best design of each run of the search and its efficiency criteria. The best design across all runs is the one returned as the design.

---

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

invalid_dummy_coding *Find dummy-coded attributes that are not correctly specified*

Description

Dummy-coded attributes must have the levels 1, 2, ..., K, where 1 is the base level, and exactly K - 1 priors, one for each level except the base level. Only the number of levels matters for the design, and using 1, 2, ..., K ensures that the expanded dummy-coded attributes and priors are named and ordered consistently.

Usage

```
invalid_dummy_coding(x)
```

Arguments

x	An object of class utility

Value

A character vector with the names of dummy-coded attributes that are not correctly specified. Empty if there are none.

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

level_balance *Print level balance of your design*

Description

Prints a table of level balance for your design. If the design is blocked you will get both level balance per block and overall level balance

Usage

```
level_balance(design, block = FALSE)
```

Arguments

design	An spdesign object
block	A boolean equal to TRUE if you want frequency tables per block. The default value is FALSE

level_occurrences_specified *Test whether level occurrences are specified in the utility functions*

Description

Test whether level occurrences are specified in the utility functions

Usage

```
level_occurrences_specified(utility)
```

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.

Value

A boolean equal to TRUE if level occurrences are specified and FALSE otherwise

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.

lvl_violation *Measure how far a design candidate is from the level occurrence constraints*

Description

For each attribute level, count how often it occurs in the design candidate and find the distance to the nearest allowed number of occurrences. Levels that do not occur in the design candidate are counted as zero occurrences.

Usage

```
lvl_violation(
  utility,
  x,
  rows,
  ranges = occurrences(utility, rows),
  lvls = expand_attribute_levels(utility)
)
```

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	A design candidate with one column per attribute
rows	Number of rows in the design
ranges	Level occurrences as returned by occurrences. Pass these in when calling repeatedly to avoid parsing the utility functions each time
lvls	Attribute levels as returned by expand_attribute_levels

Value

The total distance summed over all attribute levels. A value of 0 means that the design candidate satisfies all level occurrence constraints.

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/bh
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

---

pair_param_terms *Pair the terms of the utility functions with their parameters*

Description

Splits each updated utility function into its additive terms and pairs each term with its parameter, e.g. "b_x1x2 * I(alt1_x1 * alt1_x2)". The function is used on the output of `update_utility`, where dummy-coded attributes are already expanded.

Usage

```
pair_param_terms(x)
```

Arguments

x	A list of updated utility functions returned by `update_utility`

Value

A list with one named character vector per utility function. The values are the parameter names and the names are the terms, both without whitespace.

---

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,
  save_designs,
  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 it is built from the utility functions subject to specified exclusions with build_candidate_set. 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

save_designs	A boolean indicating whether to save up to 10 intermediate designs. The default value is FALSE.
control	A list of control options. Set 'allow_reversed_pairs = TRUE' to include the profiles of exchangeable alternatives in every order when the candidate set is built. See build_candidate_set.

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'

---

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_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,
  save_designs,
  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 it is built from the utility functions subject to specified exclusions with build_candidate_set. 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
save_designs	A boolean indicating whether to save up to 10 intermediate designs. The default value is FALSE.
control	A list of control options. Set 'allow_reversed_pairs = TRUE' to include the profiles of exchangeable alternatives in every order when the candidate set is built. See build_candidate_set.

Details

Each iteration relabels or swaps the levels of a single column of the current design candidate, working through the columns with more than one level in turn. The change is kept if it improves the current design candidate and discarded otherwise.

If there is no improvement in 'max_no_improve' iterations, the design candidate is a local optimum. The search then restarts from a new base design candidate. The best design found across all runs is returned as the design. The best design of each run, including the one that is still running when the search stops, is stored in the list element 'runs' together with its efficiency criteria.

---

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 with different levels that are swapped, so that every swap changes the design candidate. A column with a single level is returned unchanged.

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

unlisted_levels *Find attributes with level occurrences and levels that are not listed*

Description

A supplied candidate set may contain attribute levels that are not listed in the utility functions. This is only a problem for attributes with level occurrences specified, because occurrences can only be counted for listed levels.

Usage

```
unlisted_levels(utility, candidate_set, 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.
candidate_set	A candidate set in wide format
rows	Number of rows in the design

Value

A character vector with the names of the attributes with level occurrences specified, where the candidate set contains levels that are not listed in the utility functions. Empty if there are none.

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. The formula of each utility function contains its terms, i.e. the utility function without the parameters.

Usage

```
utility_formula(x)
```

Arguments

`x`	An object of class utility

Details

Note that the formulas are created from the cleaned utility expression and **not** the 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

.onAttach, 4

align_x_j, 5
all_priors_and_levels_specified, 5
any_duplicates, 6
as_whole_word, 6
attribute_level_balance, 7
attribute_levels, 7
attribute_names, 8

block, 8
build_balanced_design_candidate, 9
build_candidate_set, 10, *19*, *22*, *24*, *31–35*, *52, 53*, *58*

calculate_a_error, 11, *14*
calculate_c_error, 11, *14*
calculate_d_error, 12, *14*
calculate_efficiency, 13
calculate_efficiency_criteria, 14, *48, 49*
calculate_s_error, *14*, 15
cat, *48*
clean_utility, 16
coef.spdesign, 16
combine_profiles, 17
contains_dummies, 17
cor, 18, *18*
cycle, 18, *57*

define_profiles, 19
define_x_j, *5*, 19
derive_vcov, *5*, *11, 12*, *14, 15*, 20
derive_vcov_mnl, *20*, 21
derive_vcov_rpl, *20*, 21
digitize, 21
draw_design_candidate, 22
dummy_names, 23

eval, *44*
evaluate_design_candidate, *13*, 23
exclude, 24
expand.grid, *10*, *33*
expand_attribute_levels, 25, *39*
extract_all_names, 25, *26*
extract_attribute_names, 26
extract_distribution, 26
extract_level_occurrence, 27
extract_named_values, 27
extract_param_distribution, 28
extract_param_names, *26*, 28
extract_prior_distribution, 29
extract_specified, 29
extract_unparsed_values, 30
extract_values, 30

federov, 31
full_factorial, 33

generate_design, *31*, 33, *49*, *52*, *57*, *59*

has_bayesian_prior, 35
has_random_parameter, 36

invalid_dummy_coding, 36
is_balanced, 37

level_balance, 37
level_occurrences_specified, 38
lognormal, *45*
lognormal *(*normal*)*, 44
lognormal_p *(*normal*)*, 44
lvl_occurrences, 38
lvl_violation, 39

make_draws, 39
make_mlhs, 40
make_pseudo_random, 41
make_scrambled_halton, 41
make_scrambled_sobol, 42
make_standard_halton, 42
make_standard_sobol, 43

min_lvl_occurrence, 43
model.matrix, *19*, *65*

nlvls, 44
normal, 44, *45*
normal_p *(*normal*)*, 44

occurrences, *39*, 45

pair_param_terms, *5*, *20*, 46
parse, *44*
prepare_priors, 46
print.spdesign, 47
print_efficiency_criteria, 48, *49*
print_initial_header, 48, *49*
print_iteration_information, 49
priors, 50
probabilities, 50
probabilities_mnl, 51

radical_inverse, 51
random, 52
relabel, 53, *57*
remove_all_brackets, *25*, 54
remove_round_brackets, 54, *54*
remove_square_brackets, *54*, 55
remove_whitespace, 55
rep_cols, 56
rep_rows, 56
rsc, 57

set_default_level_occurrence, 58
set_default_options, 59
shuffle, 59
str_extract_all, *25*
summary.spdesign, 60
swap, *57*, 60

too_small, 61
transform_distribution, 61
transform_lognormal, 62
transform_normal, 62
transform_triangular, 63
transform_uniform, 63
triangular, *45*
triangular *(*normal*)*, 44
triangular_p *(*normal*)*, 44

uniform, *45*
uniform *(*normal*)*, 44
uniform_p *(*normal*)*, 44
unlisted_levels, 64
update_utility, *16*, *46*, 64
utility_formula, 65

vcov.spdesign, 66