Package 'idefix'

May 8, 2026

Type Package

Title Efficient Designs for Discrete Choice Experiments

Version 1.1.0

Maintainer Michel Meulders <michel.meulders@kuleuven.be>

Description Generates efficient designs for discrete choice experiments based on the multinomial logit model, and individually adapted designs for the mixed multinomial logit model. The generated designs can be presented on screen and choice data can be gathered using a shiny application. Traets F, Sanchez G, and Vandebroek M (2020) <doi:10.18637/jss.v096.i03>.

License GPL-3

Depends R (>= 3.1.1), shiny

LazyData TRUE

ByteCompile TRUE

Imports dplyr, MASS, parallel, Rcpp (>= 0.12.18), Rdpack, stats, scales, tmvtnorm, utils, dfidx, shinyjs, tableHTML

RoxygenNote 7.3.1

Encoding UTF-8

LinkingTo Rcpp, RcppArmadillo

RdMacros Rdpack

URL https://github.com/traets/idefix

Suggests RSGHB, bayesm, testthat, Rchoice, ChoiceModelR

NeedsCompilation yes

Author Frits Traets [aut],
Daniel Gil [ctb],
Qusai Iwidat [ctb],
Mouhannad Arabi [ctb],
Martina Vandebroek [ctb],
Michel Meulders [cre]

Repository CRAN

Date/Publication 2025-03-13 23:20:01 UTC

Contents

`idefix-package` *idefix: efficient designs for discrete choice experiments.*

Description

Generates efficient designs for discrete choice experiments based on the Multinomial Logit (MNL) model, and individually adapted designs for the Mixed Multinomial Logit model. The (adaptive) designs can be presented on screen and choice data can be gathered using a shiny application.

Author(s)

Maintainer: Michel Meulders <michel.meulders@kuleuven.be>

Other contributors:

- Frits Traets <frits.traets@gmail.com>[author]
- Daniel Gil <danielgils@gmail.com> [contributor]
- Qusai Iwidat <qusaiu@gmail.com> [contributor]
- Mouhannad Arabi <Mouhannadarabi@hotmail.com> [contributor]
- Martina Vandebroek <Martina.Vandebroek@kuleuven.be> [contributor]

References

Traets F, Sanchez G, Vandebroek M (2020). "Generating Optimal Designs for Discrete Choice Experiments in R: The idefix Package." *Journal of Statistical Software*, **96**(3).

- To generate efficient designs using the Modified Federov algorithm, please consult the Modfed documentation.
- To generate efficient designs using the Coordinate Exchange algorithm, please consult the CEA documentation.
- To generate adaptive designs using the Modified Fedorov algorithm, please consult the SeqMOD documentation.
- To generate adaptive designs using the Coordinate Exchange algorithm, please consult the SeqCEA documentation.
- To generate a discrete choice survey on screen, please consult the SurveyApp documentation.

See Also

Useful links:

- https://github.com/traets/idefix

`ABerr` *AB error*

Description

Function to calculate the AB-error given a design, and parameter values.

Usage

```
ABerr(par.draws, des, n.alts, weights = NULL, mean = TRUE)
```

Arguments

`par.draws`	Numeric matrix in which each row is a draw from a multivariate parameter distribution.
`des`	A design matrix in which each row is an alternative.
`n.alts`	Numeric value indicating the number of alternatives per choice set.
`weights`	A numeric vector containing weights of `par.draws`. The default is `NULL`.
`mean`	A logical value indicating whether the mean (AB) error should be returned or not. Default = TRUE.

Value

Numeric value indicating the AB-error of the design given the parameter draws.

Examples

```
des <- example_design
mu = c(-1, -1.5, -1, -1.5, 0.5, 1)
Sigma = diag(length(mu))
par.draws <- MASS::mvrnorm(100, mu = mu, Sigma = Sigma)
n.alts = 2
ABerr(par.draws = par.draws, des = des, n.alts = n.alts)

mu = c(-0.5, -1, -0.5, -1, 0.5, 1)
Sigma = diag(length(mu))
par.draws <- MASS::mvrnorm(100, mu = mu, Sigma = Sigma)
ABerr(par.draws = par.draws, des = des, n.alts = n.alts)
```

---

aggregate_design *Discrete choice aggregate design.*

Description

The dataset contains fictional data for seven participants, each responding to eight choice sets with two alternatives. Each alternative consists of three attributes, and each attribute contains three levels, which are dummy coded.

Usage

```
data(aggregate_design)
```

Format

A matrix with 112 rows and 9 variables

Blocks *Create blocks (sub-designs) from a given design.*

Description

This function breaks down a design output from `Modfed` or `CEA` into a specified number of blocks while aiming to maintain balance in levels frequency across the resulting blocks.

Usage

```
Blocks(
  des,
  n.blocks,
  n.alts,
  blocking.iter = 50,
  no.choice = FALSE,
  alt.cte = NULL
)
```

Arguments

`des`	The design to be distributed into blocks (sub-designs).
`n.blocks`	A numeric value indicating the desired number of blocks to create out of the provided design.
`n.alts`	The number of alternatives in each choice set.
`blocking.iter`	A numeric value indicating the maximum number of iterations for optimising the level balance in the blocks. The default value is 50.
`no.choice`	A logical value indicating whether a no choice alternative is added to each choice set in the provided design. The default is `FALSE`.
`alt.cte`	A binary vector indicating for each alternative whether an alternative specific constant is present in the design. The default is `NULL`.

Details

The argument `n.blocks` specifies the number of blocks to create. The algorithm strives to distribute the choice sets of the design evenly among the blocks, while maintaining level balance across them. The choice sets are assigned sequentially to the blocks, aiming to maintain the closest possible level balance among them up to that stage in the sequence. Hence, the algorithm runs different iterations, during each of which the choice sets in the design are shuffled randomly. The argument `blocking.iter` specifies the maximum number of these iterations.

If the design has a no.choice alternative then `no.choice` should be set to `TRUE`. Additionally, `asc.col` should indicate the number of alternative specific constants that are included in the design, if any.

This functionality is also available as an argument (`n.blocks`) when creating an efficient design using `Modfed` or `CEA`.

Note: To make sure the code works well, the names of the variables in the provided design should be aligned with variable names that the function `Profiles` produces. For example, if attribute 1 is a dummy variable of 3 levels then its corresponding columns should have numbered names such as: var11 and var12, or (if labelled) price1 and price2, for instance.

Value

A list of blocks from the original design is returned. Additionally, the frequency of every level in each block is returned.

Examples

```
# DB-efficient designs
# 3 Attributes with 3 levels, all dummy coded. 1 alternative specific constant = 7 parameters
cand.set <- Profiles(lvls = c(3, 3, 3), coding = c("D", "D", "D"))
mu <- c(0.5, 0.8, 0.2, -0.3, -1.2, 1.6, 2.2) # Prior parameter vector
v <- diag(length(mu)) # Prior variance.
set.seed(123)
pd <- MASS::mvrnorm(n = 10, mu = mu, Sigma = v) # 10 draws.
p.d <- list(matrix(pd[,1], ncol = 1), pd[,2:7])
design <- Modfed(cand.set = cand.set, n.sets = 8, n.alts = 2,
       alt.cte = c(1, 0), parallel = FALSE, par.draws = p.d)
```

```
Blocks(design$BestDesign$design, n.blocks = 2, n.alts = 2, alt.cte = c(1, 0))
```

CEA *Coordinate Exchange algorithm for MNL models.*

Description

The algorithm improves an initial start design by considering changes on an attribute-by-attribute basis. By doing this, it tries to minimize the chosen error (A(B) or D(B)-error) based on a multinomial logit model. This routine is repeated for multiple starting designs.

Usage

```
CEA(
  lvls,
  coding,
  c.lvls = NULL,
  n.sets,
  n.alts,
  par.draws,
  optim = "D",
  alt.cte = NULL,
  no.choice = FALSE,
  start.des = NULL,
  parallel = TRUE,
  max.iter = Inf,
  n.start = 12,
  overlap = NULL,
  n.blocks = 1,
  blocking.iter = 50,
  constraints = NULL
)
```

Arguments

lvls	A numeric vector which contains for each attribute the number of levels.
coding	Type of coding that needs to be used for each attribute.
c.lvls	A list containing numeric vectors with the attribute levels for each continuous attribute. The default is NULL.
n.sets	Numeric value indicating the number of choice sets.
n.alts	Numeric value indicating the number of alternatives per choice set.
par.draws	A matrix or a list, depending on alt.cte.
optim	A character value to choose between "D" and "A" optimality. The default is "D".

alt.cte	A binary vector indicating for each alternative whether an alternative specific constant is desired. The default is NULL.
no.choice	A logical value indicating whether a no choice alternative should be added to each choice set. The default is FALSE.
start.des	A list containing one or more matrices corresponding to initial start design(s). The default is NULL.
parallel	Logical value indicating whether computations should be done over multiple cores. The default is TRUE.
max.iter	A numeric value indicating the maximum number allowed iterations. The default is Inf.
n.start	A numeric value indicating the number of random start designs to use. The default is 12.
overlap	A numeric value indicating the minimum number of attributes to overlap in every choice sets to create partial profiles. The default is NULL.
n.blocks	A numeric value indicating the desired number of blocks to create out of the most efficient design.
blocking.iter	A numeric value indicating the maximum number of iterations for optimising the blocks. The default value is 50.
constraints	A list of constraints to enforce on the attributes and alternatives in every choice set. The default is NULL.

Details

Each iteration will loop through all profiles from the initial design, evaluating the change in A(B) or D(B)-error (as specified) for every level in each attribute. The algorithm stops when an iteration occurred without replacing a profile or when max.iter is reached.

By specifying a numeric vector in par.draws, the A- or D-error will be calculated and the design will be optimised locally. By specifying a matrix, in which each row is a draw from a multivariate distribution, the AB/DB-error will be calculated, and the design will be optimised globally. Whenever there are alternative specific constants, par.draws should be a list containing two matrices: The first matrix containing the parameter draws for the alternative specific constant parameters. The second matrix containing the draws for the rest of the parameters.

The AB/DB-error is calculated by taking the mean over A- / D-errors, respectively. It could be that for some draws the design results in an infinite error. The percentage of draws for which this was true for the final design can be found in the output inf.error.

Alternative specific constants can be specified in alt.cte. The length of this binary vector should equal n.alts, were 0 indicates the absence of an alternative specific constant and 1 the opposite.

start.des is a list with one or several matrices corresponding to initial start design(s). In each matrix, each row is a profile. The number of rows equals n.sets * n.alts, and the number of columns equals the number of columns of the design matrix + the number of non-zero elements in alt.cte. Consider that for a categorical attribute with *p* levels, there are *p - 1* columns in the design matrix, whereas for a continuous attribute there is only one column. If start.des = NULL, n.start random initial designs will be generated. If start designs are provided, n.start is ignored.

Note: To make sure the code works well, the names of the variables in the starting design should be aligned with variable names that the function Profiles produces. For example, if attribute 1 is a

dummy variable of 3 levels then its corresponding columns should have numbered names such as: var11 and var12, or (if labelled) price1 and price2, for instance.

If `no.choice` is `TRUE`, in each choice set an alternative with one alternative specific constant is added. The return value of the A(B) or D(B)-error is however based on the design without the no choice option.

When `parallel` is `TRUE`, `detectCores` will be used to decide upon the number of available cores. That number minus 1 cores will be used to search for efficient designs. The computation time will decrease significantly when `parallel = TRUE`.

Partial profiles/overlapping attributes

If `overlap` is set to 1 or more, then partial profiles will be used in the resulting efficient designs. The value of `overlap` determines the minimum number of attributes to overlap in each choice set. The optimising algorithm will enforce this constraint across all choice sets. Note that the running time may increase significantly, as the algorithm searches through all possible (combinations of) attributes to achieve optimisation.

Blocking

If the value of `n.blocks` is more than 1, a new list with the specified number of blocks of the best design (one with the least A(B)- or D(B)-error) will be added to the output. The algorithm strives to distribute the choice sets of the best design evenly among the blocks, while maintaining level balance across them. The choice sets are assigned sequentially to the blocks, aiming to maintain the closest possible balance among them up to that stage in the sequence. Hence, the algorithm runs different iterations, during each of which the choice sets in the design are shuffled randomly. The argument `blocking.iter` specifies the maximum number of these iterations. This functionality is also available as a separate function in `Blocks` that works with a given design.

Adding constraints to the design

The argument `constraints` can be used to determine a list of constraints to be enforced on the resulting efficient design. The package offers flexibility in the possible constraints. The basic syntax for the constraint should determine an attribute Y within an alternative X (`AltX.AttY`) and an operator to be applied on that attribute followed by a list of values or another attribute. In addition to this basic syntax, conditional If statements can be included in the conditions as will be shown in the examples below. The following operators can be used:

- =
- `!=`
- < or <=
- > or >=
- `AND`
- `OR`
- +, -, *, / operations for continuous attributes.

For example, if attributes 1, 2 and 3 are continuous attributes, then possible constraints include:

- `"Alt2.Att1 = list(100, 200)"`: restrict values of attribute 1 in alternative 2 to 100 and 200.
- `"Alt1.Att1 > Alt2.Att1"`: enforce that attribute 1 in alternative 1 to be higher than the attribute's value in alternative 2.

- "Alt1.Att1 + Alt1.Att2 < Alt1.Att3": enforce that the sum of attributes 1 and 2 to be less than the value of attribute 3 in alternative 1.
- "Alt1.Att1 > Alt1.Att3 OR Alt1.Att2 > Alt1.Att3": either attribute 1 or attribute 2 should be higher than attribute 3 in alternative 1.

For dummy and effect coded attributes, the levels are indicated with the number of the attribute followed by a letter from the alphabet. For example 1A is the first level of attribute 1 and 3D is the fourth level of attribute 3. Examples on constraints with dummy/effect coded variables:

- "Alt2.Att1 = list(1A,1B)": restrict attribute 1 in alternative 2 to the reference level (A) and the second level (B).
- "Alt1.Att1 = list(1B,1C) AND Alt2.Att2 != list(2A, 2E)": restrict attribute 1 in alternative 1 to the second and third levels, and at the same time, attribute 2 in alternative 2 cannot be the first and fifth levels of the attribute.

Additionally, and as aforementioned, conditional If statements can be included in the conditions. Examples:

- "if Alt1.Att1 != Alt2.Att1 then Alt2.Att2 = list(100,200)"
- "if Alt1.Att1 = Alt2.Att1 OR Alt1.Att1 = 0 then Alt2.Att1 > 3"

Lastly, more than one constraint can be specified at the same time. For example: constraints = list("if Alt1.Att1 != Alt2.Att1 then Alt2.Att2 = list(100,200)", "Alt1.Att3 = list (3A, 3C)").

To ensure the best use of constraints in optimising designs, please keep in mind the following:

- Proper spacing should be respected between the terms, to make sure the syntax translates properly into an R code. To clarify, spaces should be placed before and after the operators listed above. Otherwise, the console will return an error.
- Lists should be used for constrained values as shown in the examples above.
- Constraints should not be imposed on the no.choice alternative because it is fixed with zeros for all attributes. The no.choice alternative, if included, will be the last alternative in every choice set in the design. Therefore, if no.choice is TRUE and the no.choice alternative number (= n.alts) is included in the constraints, the console will return an Error.
- Attention should be given when a starting design that does not satisfy the constraint is provided. It is possible that the algorithm might not find a design that is more efficient and, at the same time, that satisfies the constraints.
- With tight constraints, the algorithm might fail to find a design that satisfies all the specified constraints.

Value

Two lists of designs and statistics are returned: First, the list BestDesign contains the design with the lowest A(B)- or D(B)- error. The method print can be used to return this list. Second, the list AllDesigns contains the results of all (provided) start designs. The method summary can be used to return this list.

design A numeric matrix wich contains an efficient design.

optimality	"A" or "D", depending on the chosen optimality criteria.
inf.error	Numeric value indicating the percentage of draws for which the D-error was Inf.
probs	Numeric matrix containing the probabilities of each alternative in each choice set. If a sample matrix was provided in par.draws, this is the average over all draws.
AB.error	Numeric value indicating the A(B)-error of the design.
DB.error	Numeric value indicating the D(B)-error of the design.
SD	The standard deviations of the parameters. Calculated by taking the diagonal of the varcov matrix, averaged over all draws if a sample matrix was provided in par.draws.
level.count	The count of all levels of each attribute in the design.
level.overlap	The count of overlapping levels accross alternatives in every choice set in the design.
Orthogonality	Numeric value indicating the degree of orthogonality of the design. The closer the value to 1, the more orthogonal the design is.
Blocks	A list showing the created blocks of the best design, along with the level counts in each block. For more details, see function Blocks.

Examples

```
# DB-efficient designs
# 3 Attributes, all dummy coded. 1 alternative specific constant = 7 parameters
mu <- c(1.2, 0.8, 0.2, -0.3, -1.2, 1.6, 2.2) # Prior parameter vector
v <- diag(length(mu)) # Prior variance.
set.seed(123)
pd <- MASS::mvrnorm(n = 10, mu = mu, Sigma = v) # 10 draws.
p.d <- list(matrix(pd[,1], ncol = 1), pd[,2:7])
CEA(lvls = c(3, 3, 3), coding = c("D", "D", "D"), par.draws = p.d,
n.alts = 2, n.sets = 8, parallel = FALSE, alt.cte = c(0, 1))
# Or AB-efficient design
set.seed(123)
CEA(lvls = c(3, 3, 3), coding = c("D", "D", "D"), par.draws = p.d,
n.alts = 2, n.sets = 8, parallel = FALSE, alt.cte = c(0, 1), optim = "A")

# DB-efficient design with categorical and continuous factors
# 2 categorical attributes with 4 and 2 levels (effect coded) and 1
# continuous attribute (= 5 parameters)
mu <- c(0.5, 0.8, 0.2, 0.4, 0.3)
v <- diag(length(mu)) # Prior variance.
set.seed(123)
pd <- MASS::mvrnorm(n = 3, mu = mu, Sigma = v) # 10 draws.
CEA(lvls = c(4, 2, 3), coding = c("E", "E", "C"), par.draws = pd,
c.lvls = list(c(2, 4, 6)), n.alts = 2, n.sets = 6, parallel = FALSE)
# The same can be done if A-optimality is chosen
set.seed(123)
CEA(lvls = c(4, 2, 3), coding = c("E", "E", "C"), par.draws = pd,
c.lvls = list(c(2, 4, 6)), n.alts = 2, n.sets = 6, parallel = FALSE, optim = "A")
```

```
# DB-efficient design with start design provided.
# 3 Attributes with 3 levels, all dummy coded (= 6 parameters).
mu <- c(0.8, 0.2, -0.3, -0.2, 0.7, 0.4)
v <- diag(length(mu)) # Prior variance.
sd <- list(example_design)
set.seed(123)
ps <- MASS::mvrnorm(n = 10, mu = mu, Sigma = v) # 10 draws.
CEA(lvls = c(3, 3, 3), coding = c("D", "D", "D"), par.draws = ps,
n.alts = 2, n.sets = 8, parallel = FALSE, start.des = sd)

# DB-efficient design with partial profiles
# 3 Attributes, all dummy coded. = 6 parameters
mu <- c(1.2, 0.8, 0.2, -0.3, -1.2, 1.6) # Prior parameter vector
v <- diag(length(mu)) # Prior variance.
set.seed(123)
pd <- MASS::mvrnorm(n = 10, mu = mu, Sigma = v) # 10 draws.
CEA(lvls = c(3, 3, 3), coding = c("D", "D", "D"), par.draws = pd,
n.alts = 2, n.sets = 8, parallel = FALSE, alt.cte = c(0, 0), overlap = 1)
# The same function but asking for blocks (and no overlap)
set.seed(123)
CEA(lvls = c(3, 3, 3), coding = c("D", "D", "D"), par.draws = pd,
n.alts = 2, n.sets = 8, parallel = FALSE, alt.cte = c(0, 0), n.blocks = 2)

# AB-efficient design with constraints
# 2 dummy coded attributes, 1 continuous attribute and 1 effect coded
# attribute (with 4 levels). = 8 parameters
mu <- c(1.2, 0.8, 0.2, 0.5, -0.3, -1.2, 1, 1.6) # Prior parameter vector
v <- diag(length(mu)) # Prior variance.
set.seed(123)
pd <- MASS::mvrnorm(n = 10, mu = mu, Sigma = v) # 10 draws.
constraints <- list("Alt2.Att1 = list(1A,1B)",
                    "if Alt1.Att3 = list(4) then Alt2.Att4 = list(4C, 4D)")
CEA(lvls = c(3, 3, 2, 4), coding = c("D", "D", "C", "E"), c.lvls = list(c(4,7)), par.draws = pd,
n.alts = 2, n.sets = 8, parallel = FALSE, alt.cte = c(0, 0), optim = "A", constraints = constraints)
```

`Datatrans` *Data transformation.*

Description

Transforms the data into the desired data format required by different estimation packages.

Usage

```
Datatrans(
  pkg,
```

```
  des,
  y,
  n.alts,
  n.sets,
  n.resp,
  bin,
  alt.names = NULL,
  coding = NULL,
  lvls = NULL
)
```

Arguments

pkg	Indicates the desired estimation package. Options are RSGHB = doHB, bayesm = rhierMnlRwMixture, bayesm::rbprobitGibbs = rbprobitGibbs (previously, in package Mixed.Probit), mlogit = mlogit, logitr = logitr, Rchoice = Rchoice, gmnl = gmnl, ChoiceModelR = choicemodelr.
des	A design matrix in which each row is a profile.
y	A numeric vector containing binary or discrete responses. See bin argument.
n.alts	Numeric value indicating the number of alternatives per choice set.
n.sets	Numeric value indicating the number of choice sets.
n.resp	Numeric value indicating the number of respondents.
bin	Logical value indicating whether the response vector contains binary data (TRUE) or discrete data (FALSE). See y argument.
alt.names	A character vector containing the names of the alternatives. The default = NULL.
coding	Type of coding that needs to be used for each attribute. To be used only for package "ChoiceModelR". The default = NULL.
lvls	A numeric vector which contains for each attribute the number of levels. To be used only for package "ChoiceModelR". The default = NULL.

Details

The design (des) specified should be the full aggregated design. Thus, if all participants responded to the same design, des will be a repetition of that design matrix.

The responses in y should be successive when there are multiple respondents. There can be n.sets elements for each respondent with discrete values indicating the chosen alternative for each set. Or there can be n.sets * n.alts elements for each respondent with binary values indicating for each alternative whether it was chosen or not. In the latter case the bin argument should be TRUE.

n.sets indicates the number of sets each respondent responded to. It is assumed that every responded responded to the same number of choice sets.

Value

The data ready to be used by the specified package.

Examples

```
idefix.data <- aggregate_design
des <- as.matrix(idefix.data[, 3:8], ncol = 6)
y <- idefix.data[, 9]
bayesm.data <- Datatrans(pkg = "bayesm", des = des, y = y,
n.alts = 2, n.sets = 8, n.resp = 7, bin = TRUE)
rbprobit.data <- Datatrans(pkg = "bayesm::rbprobitGibbs", des = des, y = y,
 n.alts = 2, n.sets = 8, n.resp = 7, bin = TRUE)
mlogit.data <- Datatrans(pkg = "mlogit", des = des, y = y,
 n.alts = 2, n.sets = 8, n.resp = 7, bin = TRUE)
ChoiceM.data <- Datatrans(pkg = "ChoiceModelR", des = des, y = y,
 n.alts = 2, n.sets = 8, n.resp = 7, bin = TRUE, coding = c("D", "D", "D"), lvls = c(3, 3, 3))
```

DBerr *DB error*

Description

Function to calculate the DB-error given a design, and parameter values.

Usage

```
DBerr(par.draws, des, n.alts, weights = NULL, mean = TRUE)
```

Arguments

par.draws	Numeric matrix in which each row is a draw from a multivariate parameter distribution.
des	A design matrix in which each row is an alternative.
n.alts	Numeric value indicating the number of alternatives per choice set.
weights	A numeric vector containing weights of par.draws. The default is NULL.
mean	A logical value indicating whether the mean (DB) error should be returned or not. Default = TRUE.

Value

Numeric value indicating the DB-error of the design given the parameter draws.

Examples

```
des <- example_design
mu = c(-1, -1.5, -1, -1.5, 0.5, 1)
Sigma = diag(length(mu))
par.draws <- MASS::mvrnorm(100, mu = mu, Sigma = Sigma)
n.alts = 2
DBerr(par.draws = par.draws, des = des, n.alts = n.alts)
```

```
mu = c(-0.5, -1, -0.5, -1, 0.5, 1)
Sigma = diag(length(mu))
par.draws <- MASS::mvrnorm(100, mu = mu, Sigma = Sigma)
DBerr(par.draws = par.draws, des = des, n.alts = n.alts)
```

`Decode` *Coded design to readable design.*

Description

Transforms a coded design matrix into a design containing character attribute levels, ready to be used in a survey. The frequency of each attribute level in the design is also included in the output.

Usage

```
Decode(
  des,
  n.alts,
  lvl.names,
  coding,
  alt.cte = NULL,
  c.lvls = NULL,
  no.choice = NULL
)
```

Arguments

`des`	A numeric matrix which represents the design matrix. Each row is a profile.
`n.alts`	Numeric value indicating the number of alternatives per choice set.
`lvl.names`	A list containing character vectors with the values of each level of each attribute.
`coding`	A character vector denoting the type of coding used for each attribute. See also `Profiles`.
`alt.cte`	A binary vector indicating for each alternative if an alternative specific constant is present. The default is `NULL`.
`c.lvls`	A list containing numeric vectors with the attribute levels for each continuous attribute. The default is `NULL`.
`no.choice`	An integer indicating the no choice alternative. The default is `NULL`.

Details

`des` A design matrix, this can also be a single choice set. See for example the output of Modfed or CEA.

In `lvl.names`, the number of character vectors in the list should equal the number of attributes in de choice set. The number of elements in each character vector should equal the number of levels for that attribute.

Valid arguments for coding are C, D and E. When using C the attribute will be treated as continuous and no coding will be applied. All possible levels of that attribute should then be specified in c.lvls. If D (dummy coding) is used contr.treatment will be applied to that attribute. The first attribute wil be used as reference level. For E (effect coding) contr.sum is applied, in this case the last attribute level is used as reference level.

If des contains columns for alternative specific constants, alt.cte should be specified. In this case, the first column(s) (equal to the number of nonzero elements in alt.cte) will be removed from des before decoding the alternatives.

Value

design	A character matrix which represents the design.
level.count	A list containing the frequency of appearance of each attribute level in the design.

Examples

```
## Not run:
# Example without continuous attributes.
design <- example_design
coded <- c("D", "D", "D") # Coding.
# Levels as they should appear in survey.
al <- list(
  c("$50", "$75", "$100"), # Levels attribute 1.
  c("2 min", "15 min", "30 min"), # Levels attribute 2.
  c("bad", "moderate", "good") # Levels attribute 3.
)
# Decode
Decode(des = design, n.alts = 2, lvl.names = al, coding = coded)

# Example with alternative specific constants
design <- example_design2
coded <- c("D", "D", "D") # Coding.
# Levels as they should appear in survey.
al <- list(
  c("$50", "$75", "$100"), # Levels attribute 1.
  c("2 min", "15 min", "30 min"), # Levels attribute 2.
  c("bad", "moderate", "good") # Levels attribute 3.
)
# Decode
Decode(des = design, n.alts = 3, lvl.names = al, coding = coded, alt.cte = c(1, 1, 0))

## End(Not run)
```

EvaluateDesign *Calculate efficiency measures for a given design*

Description

This function calculates the following measures for a given design: AB-error, DB-error, standard deviations of the parameters, level frequency, level overlaps, and orthogonality.

Usage

```
EvaluateDesign(des, par.draws, n.alts, alt.cte = NULL, no.choice = FALSE)
```

Arguments

`des`	A design matrix in which each row is an alternative.
`par.draws`	A matrix or a list, depending on `alt.cte`.
`n.alts`	Numeric value indicating the number of alternatives per choice set.
`alt.cte`	A binary vector indicating for each alternative whether an alternative specific constant is present in the design. The default is `NULL`.
`no.choice`	A logical value indicating whether the design has a no choice alternative. The default is `FALSE`.

Details

The rules for specifying the function arguments are the same as in `Modfed` or `CEA`.

Alternative specific constants can be specified in `alt.cte`, if the design has any. The length of this binary vector should equal `n.alts`, were `0` indicates the absence of an alternative specific constant and `1` the opposite.

`par.draws` should be a matrix in which each row is a draw from a multivariate distribution. However, if there are alternative specific constants in the specified design, then `par.draws` should be a list containing two matrices: The first matrix containing the parameter draws for the alternative specific constant parameters. The second matrix containing the draws for the rest of the parameters.

If the design has a no.choice alternative, then `no.choice` should be set to `TRUE`.

Value

`AB.error`	Numeric value indicating the A(B)-error of the design.
`DB.error`	Numeric value indicating the D(B)-error of the design.
`SD`	The standard deviations of the parameters. Calculated by taking the diagonal of the varcov matrix, averaged over all draws if a sample matrix was provided in `par.draws`.
`level.count`	The count of all levels of each attribute in the design.
`level.overlap`	The count of overlapping levels accross alternatives in every choice set in the design.
`Orthogonality`	Numeric value indicating the degree of orthogonality of the design. The closer the value to 1, the more orthogonal the design is.

Examples

```
des <- example_design
mu = c(-1, -1.5, -1, -1.5, 0.5, 1)
Sigma = diag(length(mu))
par.draws <- MASS::mvrnorm(100, mu = mu, Sigma = Sigma)
n.alts = 2
EvaluateDesign(des = des, par.draws = par.draws, n.alts = n.alts)

#Example with a no.choice alternative
des.nc <- nochoice_design
mu = c(0.2, -0.5, -1, -0.5, -1, 0.5, 1)
Sigma = diag(length(mu))
par.draws <- MASS::mvrnorm(100, mu = mu, Sigma = Sigma)
par.draws <- list(par.draws[,1], par.draws[,-1])
n.alts = 3
EvaluateDesign(des = des.nc, par.draws = par.draws, n.alts = n.alts,
   alt.cte = c(0,0,1), no.choice = TRUE)
```

---

example_design *Discrete choice design.*

Description

This discrete choice design is generated using the Modfed function. There are 8 choice sets, each containig 2 alternatives (rows). The alternatives consist of 3 attributes (time, price, comfort) with 3 levels each, all of which are dummy coded (columns).

Usage

```
data(example_design)
```

Format

A matrix with 16 rows and 6 columns.

---

example_design2 *Discrete choice design.*

Description

This discrete choice design is generated using the Modfed function. There are 8 choice sets, each containig 3 alternatives (rows). The alternatives consist of 3 attributes (time, price, comfort) with 3 levels each, all of which are dummy coded (columns). The first two colums are alternative specific constants for alternative 1 and 2.

Usage

```
data(example_design2)
```

Format

A matrix with 24 rows and 8 columns.

`ImpsampMNL` *Importance sampling MNL*

Description

This function samples from the posterior distribution using importance sampling, assuming a multivariate (truncated) normal prior distribution and a MNL likelihood.

Usage

```
ImpsampMNL(
  n.draws,
  prior.mean,
  prior.covar,
  des,
  n.alts,
  y,
  alt.cte = NULL,
  lower = NULL,
  upper = NULL
)
```

Arguments

`n.draws`	Numeric value indicating the number of draws.
`prior.mean`	Numeric vector indicating the mean of the multivariate normal distribution (prior).
`prior.covar`	Covariance matrix of the prior distribution.
`des`	A design matrix in which each row is a profile. If alternative specific constants are present, those should be included as the first column(s) of the design. Can be generated with `Modfed` or `CEA`.
`n.alts`	Numeric value indicating the number of alternatives per choice set.
`y`	A binary response vector. `RespondMNL` can be used to simulate response data.
`alt.cte`	A binary vector indicating for each alternative whether an alternative specific constant is desired. The default is `NULL`.
`lower`	Numeric vector of lower truncation points, the default is `NULL`.
`upper`	Numeric vector of upper truncation points, the default is `NULL`.

Details

For the proposal distribution a t-distribution with degrees of freedom equal to the number of parameters is used. The posterior mode is estimated using optim, and the covariance matrix is calculated as the negative inverse of the generalized Fisher information matrix. See reference for more information.

From this distribution a lattice grid of draws is generated.

If truncation is present, incorrect draws are rejected and new ones are generated untill `n.draws` is reached. The covariance matrix is in this case still calculated as if no truncation was present.

Value

`sample`	Numeric vector with the (unweigthted) draws from the posterior distribution.
`weights`	Numeric vector with the associated weights of the draws.
`max`	Numeric vector with the estimated mode of the posterior distribution.
`covar`	Matrix representing the estimated variance covariance matrix.

References

Yu J, Goos P, Vandebroek M (2011). "Individually adapted sequential Bayesian conjoint-choice designs in the presence of consumer heterogeneity."

Examples

```
## Example 1: sample from posterior, no constraints, no alternative specific constants
# choice design
design <- example_design
# Respons.
truePar <- c(0.7, 0.6, 0.5, -0.5, -0.7, 1.7) # some values
set.seed(123)
resp <- RespondMNL(par = truePar, des = design, n.alts = 2)
#prior
pm <- c(1, 1, 1, -1, -1, 1) # mean vector
pc <- diag(1, ncol(design)) # covariance matrix
# draws from posterior.
ImpsampMNL(n.draws = 100, prior.mean =  pm, prior.covar = pc,
           des = design, n.alts = 2, y = resp)

## example 2:  sample from posterior with constraints
# and alternative specific constants
# choice design.
design <- example_design2
# Respons.
truePar <- c(0.2, 0.8, 0.7, 0.6, 0.5, -0.5, -0.7, 1.7) # some values
set.seed(123)
resp <- RespondMNL(par = truePar, des = design, n.alts = 3)
# prior
pm <- c(1, 1, 1, 1, 1, -1, -1, 1) # mean vector
pc <- diag(1, ncol(design)) # covariance matrix
low = c(-Inf, -Inf, 0, 0, 0, -Inf, -Inf, 0)
```

```
up = c(Inf, Inf, Inf, Inf, Inf, 0, 0, Inf)
# draws from posterior.
ImpsampMNL(n.draws = 100, prior.mean =  pm, prior.covar = pc, des = design,
           n.alts = 3, y = resp, lower = low, upper = up, alt.cte = c(1, 1, 0))
```

`LoadData` *Load numeric choice data from directory*

Description

Reads all individual choice data files, created by `SurveyApp` function, from a directory and concatenates those files into a single data file. Files containing either "num" or "char" will be read, with num indicating numeric data and char indicating character data. For more information, see output of `SurveyApp`.

Usage

```
LoadData(data.dir, type)
```

Arguments

`data.dir`	A character string containing the directory to read from.
`type`	Character vector containing either num or char.

Value

A data frame containg the full design and all the responses of the combined data files that were found. Different files are indicated by an ID variable.

`Modfed` *Modified Fedorov algorithm for MNL models.*

Description

The algorithm swaps every profile of an initial start design with candidate profiles. By doing this, it tries to minimize the D(B)-error, based on a multinomial logit model. This routine is repeated for multiple starting designs.

Usage

```
Modfed(
  cand.set,
  n.sets,
  n.alts,
  par.draws,
  optim = "D",
  alt.cte = NULL,
  no.choice = FALSE,
  start.des = NULL,
  parallel = TRUE,
  max.iter = Inf,
  n.start = 12,
  overlap = NULL,
  n.blocks = 1,
  blocking.iter = 50,
  constraints = NULL
)
```

Arguments

cand.set	A numeric matrix in which each row is a possible profile. The Profiles function can be used to generate this matrix.
n.sets	Numeric value indicating the number of choice sets.
n.alts	Numeric value indicating the number of alternatives per choice set.
par.draws	A matrix or a list, depending on alt.cte.
optim	A character value to choose between "D" and "A" optimality. The default is "D".
alt.cte	A binary vector indicating for each alternative whether an alternative specific constant is desired. The default is NULL.
no.choice	A logical value indicating whether a no choice alternative should be added to each choice set. The default is FALSE.
start.des	A list containing one or more matrices corresponding to initial start design(s). The default is NULL.
parallel	Logical value indicating whether computations should be done over multiple cores. The default is TRUE.
max.iter	A numeric value indicating the maximum number allowed iterations. The default is Inf.
n.start	A numeric value indicating the number of random start designs to use. The default is 12.
overlap	A numeric value indicating the minimum number of attributes to overlap in every choice sets to create partial profiles. The default is NULL.
n.blocks	A numeric value indicating the desired number of blocks to create out of the most efficient design.
blocking.iter	A numeric value indicating the maximum number of iterations for optimising the blocks. The default value is 50.

`constraints` A list of constraints to enforce on the attributes and alternatives in every choice set. The default is `NULL`.

Details

Each iteration will loop through all profiles from the initial design, evaluating the change in A(B) or D(B)-error (as specified) for every profile from `cand.set`. The algorithm stops when an iteration occurred without replacing a profile or when `max.iter` is reached.

By specifying a numeric vector in `par.draws`, the A- or D-error will be calculated and the design will be optimised locally. By specifying a matrix, in which each row is a draw from a multivariate distribution, the AB/DB-error will be calculated, and the design will be optimised globally. Whenever there are alternative specific constants, `par.draws` should be a list containing two matrices: The first matrix containing the parameter draws for the alternative specific constant parameters. The second matrix containing the draws for the rest of the parameters.

The AB/DB-error is calculated by taking the mean over A/D-errors, respectively. It could be that for some draws the design results in an infinite error. The percentage of draws for which this was true for the final design can be found in the output `inf.error`.

Alternative specific constants can be specified in `alt.cte`. The length of this binary vector should equal `n.alts`, were `0` indicates the absence of an alternative specific constant and `1` the opposite.

`start.des` is a list with one or several matrices corresponding to initial start design(s). In each matrix, each row is a profile. The number of rows equals `n.sets * n.alts`, and the number of columns equals the number of columns of `cand.set` + the number of non-zero elements in `alt.cte`. If `start.des = NULL`, `n.start` random initial designs will be generated. If start designs are provided, `n.start` is ignored.

Note: To make sure the code works well, the names of the variables in the starting design should be aligned with variable names that the function `Profiles` produces. For example, if attribute 1 is a dummy variable of 3 levels then its corresponding columns should have numbered names such as: var11 and var12, or (if labelled) price1 and price2, for instance.

If `no.choice` is `TRUE`, in each choice set an alternative with one alternative specific constant is added. The return value of the A(B) or D(B)-error is however based on the design without the no choice option.

When `parallel` is `TRUE`, `detectCores` will be used to decide upon the number of available cores. That number minus 1 cores will be used to search for efficient designs. The computation time will decrease significantly when `parallel = TRUE`.

Partial profiles/overlapping attributes

If `overlap` is set to 1 or more, then partial profiles will be used in the resulting efficient designs. The value of `overlap` determines the minimum number of attributes to overlap in each choice set. The optimising algorithm will enforce this constraint across all choice sets. Note that the running time may increase significantly, as the algorithm searches through all possible (combinations of) attributes to achieve optimisation.

Blocking

If the value of `n.blocks` is more than 1, a new list with the specified number of blocks of the best design (one with the least A(B)- or D(B)-error) will be added to the output. The algorithm strives to distribute the choice sets of the best design evenly among the blocks, while maintaining level balance across them. The choice sets are assigned sequentially to the blocks, aiming to maintain the

closest possible balance among them up to that stage in the sequence. Hence, the algorithm runs different iterations, during each of which the choice sets in the design are shuffled randomly. The argument `blocking.iter` specifies the maximum number of these iterations. This functionality is also available as a separate function in `Blocks` that works with a given design.

Adding constraints to the design

The argument `constraints` can be used to determine a list of constraints to be enforced on the resulting efficient design. The package offers flexibility in the possible constraints. The basic syntax for the constraint should determine an attribute Y within an alternative X (`AltX.AttY`) and an operator to be applied on that attribute followed by a list of values or another attribute. In addition to this basic syntax, conditional If statements can be included in the conditions as will be shown in the examples below. The following operators can be used:

- =
- `!=`
- < or <=
- > or >=
- `AND`
- `OR`
- +, -, *, / operations for continuous attributes.

For example, if attributes 1, 2 and 3 are continuous attributes, then possible constraints include:

- `"Alt2.Att1 = list(100, 200)"`: restrict values of attribute 1 in alternative 2 to 100 and 200.
- `"Alt1.Att1 > Alt2.Att1"`: enforce that attribute 1 in alternative 1 to be higher than the attribute's value in alternative 2.
- `"Alt1.Att1 + Alt1.Att2 < Alt1.Att3"`: enforce that the sum of attributes 1 and 2 to be less than the value of attribute 3 in alternative 1.
- `"Alt1.Att1 > Alt1.Att3 OR Alt1.Att2 > Alt1.Att3"`: either attribute 1 or attribute 2 should be higher than attribute 3 in alternative 1.

For dummy and effect coded attributes, the levels are indicated with the number of the attribute followed by a letter from the alphabet. For example `1A` is the first level of attribute 1 and `3D` is the fourth level of attribute 3. Examples on constraints with dummy/effect coded variables:

- `"Alt2.Att1 = list(1A,1B)"`: restrict attribute 1 in alternative 2 to the reference level (A) and the second level (B).
- `"Alt1.Att1 = list(1B,1C) AND Alt2.Att2 != list(2A, 2E)"`: restrict attribute 1 in alternative 1 to the second and third levels, and at the same time, attribute 2 in alternative 2 cannot be the first and fifth levels of the attribute.

Additionally, and as aforementioned, conditional If statements can be included in the conditions. Examples:

- `"if Alt1.Att1 != Alt2.Att1 then Alt2.Att2 = list(100,200)"`
- `"if Alt1.Att1 = Alt2.Att1 OR Alt1.Att1 = 0 then Alt2.Att1 > 3"`

Lastly, more than one constraint can be specified at the same time. For example: `constraints = list("if Alt1.Att1 != Alt2.Att1 then Alt2.Att2 = list(100,200)", "Alt1.Att3 = list (3A, 3C)")`.

To ensure the best use of constraints in optimising designs, please keep in mind the following:

- Proper spacing should be respected between the terms, to make sure the syntax translates properly into an R code. To clarify, spaces should be placed before and after the operators listed above. Otherwise, the console will return an error.
- Lists should be used for constrained values as shown in the examples above.
- Constraints should not be imposed on the `no.choice` alternative because it is fixed with zeros for all attributes. The `no.choice` alternative, if included, will be the last alternative in every choice set in the design. Therefore, if `no.choice` is `TRUE` and the no.choice alternative number (= `n.alts`) is included in the constraints, the console will return an Error.
- Attention should be given when a starting design that does not satisfy the constraint is provided. It is possible that the algorithm might not find a design that is more efficient and, at the same time, that satisfies the constraints.
- With tight constraints, the algorithm might fail to find a design that satisfies all the specified constraints.

Value

Two lists of designs and statistics are returned: First, the list `BestDesign` contains the design with the lowest A(B)- or D(B)- error. The method `print` can be used to return this list. Second, the list `AllDesigns` contains the results of all (provided) start designs. The method `summary` can be used to return this list.

`design`	A numeric matrix wich contains an efficient design.
`optimality`	`"A"` or `"D"`, depending on the chosen optimality criteria.
`inf.error`	Numeric value indicating the percentage of draws for which the D-error was `Inf`.
`probs`	Numeric matrix containing the probabilities of each alternative in each choice set. If a sample matrix was provided in `par.draws`, this is the average over all draws.
`AB.error`	Numeric value indicating the A(B)-error of the design.
`DB.error`	Numeric value indicating the D(B)-error of the design.
`SD`	The standrad deviation of the parameters. Calculated by taking the diagonal of the varcov matrix, averaged over all draws if a sample matrix was provided in `par.draws`.
`level.count`	The count of all levels of each attribute in the design.
`level.overlap`	The count of overlapping levels accross alternatives in every choice set in the design.
`Orthogonality`	Numeric value indicating the degree of orthogonality of the design. The closer the value to 1, the more orthogonal the design is.
`Blocks`	A list showing the created blocks of the best design, along with the level counts in each block. For more details, see function `Blocks`.

References

Traets F, Sanchez G, Vandebroek M (2020). "Generating Optimal Designs for Discrete Choice Experiments in R: The idefix Package." *Journal of Statistical Software*, **96**(3).

Examples

```
## Not run:
# DB-efficient designs
# 3 Attributes, all dummy coded. 1 alternative specific constant = 7 parameters
cand.set <- Profiles(lvls = c(3, 3, 3), coding = c("D", "D", "D"))
mu <- c(0.5, 0.8, 0.2, -0.3, -1.2, 1.6, 2.2) # Prior parameter vector
v <- diag(length(mu)) # Prior variance.
set.seed(123)
pd <- MASS::mvrnorm(n = 10, mu = mu, Sigma = v) # 10 draws.
p.d <- list(matrix(pd[,1], ncol = 1), pd[,2:7])
Modfed(cand.set = cand.set, n.sets = 8, n.alts = 2, alt.cte = c(1, 0),
       parallel = FALSE, par.draws = p.d)
# Or AB-efficient design
set.seed(123)
Modfed(cand.set = cand.set, n.sets = 8, n.alts = 2, alt.cte = c(1, 0),
       parallel = FALSE, par.draws = p.d, optim = "A")

# DB-efficient design with start design provided.
# 3 Attributes with 3 levels, all dummy coded (= 6 parameters).
cand.set <- Profiles(lvls = c(3, 3, 3), coding = c("D", "D", "D"))
mu <- c(0.8, 0.2, -0.3, -0.2, 0.7, 0.4) # Prior mean (total = 5 parameters).
v <- diag(length(mu)) # Prior variance.
sd <- list(example_design)
set.seed(123)
ps <- MASS::mvrnorm(n = 10, mu = mu, Sigma = v) # 10 draws.
Modfed(cand.set = cand.set, n.sets = 8, n.alts = 2,
       alt.cte = c(0, 0), parallel = FALSE, par.draws = ps, start.des = sd)

# DB-efficient design with partial profiles
# 3 Attributes, all dummy coded. = 5 parameters
cand.set <- Profiles(lvls = c(3, 3, 2), coding = c("D", "D", "D"))
mu <- c(1.2, 0.8, 0.2, -0.3, -1.2) # Prior parameter vector
v <- diag(length(mu)) # Prior variance.
set.seed(123)
pd <- MASS::mvrnorm(n = 10, mu = mu, Sigma = v) # 10 draws.
Modfed(cand.set = cand.set, par.draws = pd, n.alts = 2,
n.sets = 8, parallel = TRUE, alt.cte = c(0, 0), overlap = 1)
# The same function but asking for blocks (and no overlap)
set.seed(123)
Modfed(cand.set = cand.set, par.draws = pd, n.alts = 2,
n.sets = 8, parallel = TRUE, alt.cte = c(0, 0), n.blocks = 2)

# AB-efficient design with constraints
# 2 dummy coded attributes, 1 continuous attribute and 1 effect coded
# attribute (with 4 levels). = 8 parameters
cand.set <- Profiles(lvls = c(3, 3, 2, 4), coding = c("D", "D", "C", "E"),
     c.lvls = list(c(4,7)))
```

```
mu <- c(1.2, 0.8, 0.2, 0.5, -0.3, -1.2, 1, 1.6) # Prior parameter vector
v <- diag(length(mu)) # Prior variance.
set.seed(123)
pd <- MASS::mvrnorm(n = 10, mu = mu, Sigma = v) # 10 draws.
constraints <- list("Alt2.Att1 = list(1A,1B)",
                    "if Alt1.Att3 = list(4) then Alt2.Att4 = list(4C, 4D)")
Modfed(cand.set = cand.set, par.draws = pd, n.alts = 2, n.sets = 8,
parallel = TRUE, alt.cte = c(0, 0), optim = "A", constraints = constraints)

## End(Not run)
```

`nochoice_design` *Discrete choice design with no choice option.*

Description

This discrete choice design is generated using the `Modfed` function. There are 8 choice sets, each containig 3 alternatives (rows), of which one is a no choice option. The no choice option consist of an alternative specific constant and zero's for all other attribute levels. There are three attributes (time, price, comfort) with 3 levels each, all of which are dummy coded (columns).

Usage

```
data(nochoice_design)
```

Format

A matrix with 24 rows and 7 variables

`Profiles` *Profiles generation.*

Description

Function to generate all possible combinations of attribute levels (i.e. all possible profiles).

Usage

```
Profiles(lvls, coding, c.lvls = NULL)
```

Arguments

`lvls`	A numeric vector which contains for each attribute the number of levels.
`coding`	Type of coding that needs to be used for each attribute.
`c.lvls`	A list containing numeric vectors with the attribute levels for each continuous attribute. The default is `NULL`.

Details

Valid arguments for `coding` are `C`, `D` and `E`. When using `C` the attribute will be treated as continuous and no coding will be applied. All possible levels should then be specified in `c.lvls`. If `D` (dummy coding) is used `contr.treatment` will be applied to that attribute. For `E` (effect coding) `contr.sum` will be applied.

Value

A numeric matrix which contains all possible profiles.

Examples

```
# Without continuous attributes
at.lvls <- c(3, 4, 2) # 3 Attributes with respectively 3, 4 and 2 levels.
c.type <- c("E", "E", "E") # All Effect coded.
Profiles(lvls = at.lvls, coding = c.type) # Generate profiles.

# With continuous attributes
at.lvls <- c(3, 4, 2) # 3 attributes with respectively 3, 4 and 2 levels.
# First attribute is dummy coded, second and third are continuous.
c.type <- c("D", "C", "C")
# Levels for continuous attributes, in the same order.
con.lvls <- list(c(4, 6, 8, 10), c(7, 9))
Profiles(lvls = at.lvls, coding = c.type, c.lvls = con.lvls)
```

RespondMNL *Response generation*

Description

Function to generate random responses given parameter values and a design matrix, assuming a MNL model.

Usage

```
RespondMNL(par, des, n.alts, bin = TRUE)
```

Arguments

`par`	Numeric vector containing parameter values.
`des`	A design matrix in which each row is a profile. If alternative specific constants are present, those should be included as the first column(s) of the design. Can be generated with `Modfed` or `CEA`.
`n.alts`	Numeric value indicating the number of alternatives per choice set.
`bin`	A logical value indicating whether the returned value should be a binary vector or a discrete value which denotes the chosen alternative.

Value

Numeric vector indicating the chosen alternatives.

Examples

```
# design: 3 dummy coded attributes, each 3 levels. There are 8 choice sets.
des <- example_design
set.seed(123)
true_par <- rnorm(6)
RespondMNL(par = true_par, des = des, n.alts = 2)
```

SeqCEA *Sequential Coordinate Exchange algorithm for MNL model.*

Description

Selects the choice set that minimizes the DB-error when added to an initial design, given (updated) parameter values.

Usage

```
SeqCEA(
  des = NULL,
  lvls,
  coding,
  c.lvls = NULL,
  n.alts,
  par.draws,
  prior.covar,
  alt.cte = NULL,
  no.choice = NULL,
  weights = NULL,
  parallel = TRUE,
  reduce = TRUE,
  n.cs = NULL
)
```

Arguments

des	A design matrix in which each row is a profile. If alternative specific constants are present, those should be included as the first column(s) of the design. Can be generated with Modfed or CEA
lvls	A numeric vector which contains for each attribute the number of levels.
coding	Type of coding that needs to be used for each attribute.
c.lvls	A list containing numeric vectors with the attribute levels for each continuous attribute. The default is NULL.

n.alts	Numeric value indicating the number of alternatives per choice set.
par.draws	A matrix or a list, depending on alt.cte.
prior.covar	Covariance matrix of the prior distribution.
alt.cte	A binary vector indicating for each alternative whether an alternative specific constant is desired. The default is NULL.
no.choice	An integer indicating the no choice alternative. The default is NULL.
weights	A vector containing the weights of the draws. Default is NULL. See also ImpsampMNL.
parallel	Logical value indicating whether computations should be done over multiple cores.
reduce	Logical value indicating whether the candidate set should be reduced or not.
n.cs	An integer indicating the number of possible random choice sets to consider in the search for the next best choice set possible. The default is NULL.

Details

This algorithm is ideally used in an adaptive context. The algorithm will select the next DB-efficient choice set given parameter values and possible previously generated choice sets. In an adaptive context these parameter values are updated after each observed response.

Previously generated choice sets, which together form an initial design, can be provided in des. When no design is provided, the algorithm will select the most efficient choice set based on the fisher information of the prior covariance matrix prior.covar.

If alt.cte = NULL, par.draws should be a matrix in which each row is a sample from the multivariate parameter distribution. In case that alt.cte is not NULL, a list containing two matrices should be provided to par.draws. The first matrix containing the parameter draws for the alternative specific parameters. The second matrix containing the draws for the rest of the parameters.

The list of potential choice sets is created by selecting randomly a level for each attribute in an alternative/profile. n.cs controls the number of potential choice sets to consider. The default is NULL, which means that the number of possible choice sets is the product of attribute levels considered in the experiment. For instance, an experiment with 3 attribute and 3 levels each will consider 3^3 = 27 possible choice sets.

The weights argument can be used when the par.draws have weights. This is for example the case when parameter values are updated using ImpsampMNL.

When parallel is TRUE, detectCores will be used to decide upon the number of available cores. That number minus 1 cores will be used to search for the optimal choice set. For small problems (6 parameters), parallel = TRUE can be slower. For larger problems the computation time will decrease significantly.

Note: this function is faster than SeqMOD, but the output is not as stable. This happens because this function makes a random search to get the choice set, whereas SeqMOD makes an exhaustive search.

Value

set	A matrix representing a DB efficient choice set.
error	A numeric value indicating the DB-error of the whole design.

References

Traets F, Sanchez G, Vandebroek M (2020). "Generating Optimal Designs for Discrete Choice Experiments in R: The idefix Package." *Journal of Statistical Software*, **96**(3).

Yu J, Goos P, Vandebroek M (2011). "Individually adapted sequential Bayesian conjoint-choice designs in the presence of consumer heterogeneity."

Meyer RK, Nachtsheim CJ (1995). "The Coordinate-Exchange Algorithm for Constructing Exact Optimal Experimental Designs." *Technometrics*, **37**(1), 60–69. ISSN 00401706, https://www.jstor.org/stable/1269153.

Kessels R, Jones B, Goos P, Vandebroek M (2009). "An Efficient Algorithm for Constructing Bayesian Optimal Choice Designs." *Journal of Business & Economic Statistics*, **27**(2), 279–291. ISSN 07350015.

Examples

```
# DB efficient choice set, given a design and parameter draws.
# 3 attributes with 3 levels each
m <- c(0.3, 0.2, -0.3, -0.2, 1.1, 2.4) # mean (total = 6 parameters).
pc <- diag(length(m)) # covariance matrix
set.seed(123)
sample <- MASS::mvrnorm(n = 10, mu = m, Sigma = pc)
# Initial design.
des <- example_design
# Efficient choice set to add.
SeqCEA(des = des, lvls = c(3, 3, 3), coding = c("D", "D", "D"), n.alts = 2,
       par.draws = sample, prior.covar = pc, parallel = FALSE)

# DB efficient choice set, given parameter draws.
# with alternative specific constants
des <- example_design2
ac <- c(1, 1, 0) # Alternative specific constants.
m <- c(0.3, 0.2, -0.3, -0.2, 1.1, 2.4, 1.8, 1.2) # mean
pc <- diag(length(m)) # covariance matrix
pos <- MASS::mvrnorm(n = 10, mu = m, Sigma = pc)
sample <- list(pos[ , 1:2], pos[ , 3:8])
# Efficient choice set.
SeqCEA(des = des, lvls = c(3, 3, 3), coding = c("D", "D", "D"), n.alts = 3,
      par.draws = sample, alt.cte = ac, prior.covar = pc, parallel = FALSE)
```

SeqKL *Sequential Kullback-Leibler based algorithm for the MNL model.*

Description

Selects the choice set that maximizes the Kullback-Leibler divergence between the prior parameter values and the expected posterior, assuming a MNL model.

Usage

```
SeqKL(
  des = NULL,
  cand.set,
  n.alts,
  par.draws,
  alt.cte = NULL,
  no.choice = NULL,
  weights = NULL,
  allow.rep = FALSE
)
```

Arguments

des	A design matrix in which each row is a profile. If alternative specific constants are present, those should be included as the first column(s) of the design. Can be generated with Modfed or CEA.
cand.set	A numeric matrix in which each row is a possible profile. The Profiles function can be used to generate this matrix.
n.alts	Numeric value indicating the number of alternatives per choice set.
par.draws	A matrix or a list, depending on alt.cte.
alt.cte	A binary vector indicating for each alternative if an alternative specific constant is desired.
no.choice	An integer indicating the no choice alternative. The default is NULL.
weights	A vector containing the weights of the draws. Default is NULL, See also ImpsampMNL.
allow.rep	Logical value indicating whether repeated choice sets are allowed in the design.

Details

This algorithm is ideally used in an adaptive context. The algorithm selects the choice set that maximizes the Kullback-Leibler divergence between prior and expected posterior. Otherwisely framed the algorithm selects the choice set that maximizes the expected information gain.

If alt.cte = NULL, par.draws should be a matrix in which each row is a sample from the multivariate parameter distribution. In case that alt.cte is not NULL, a list containing two matrices should be provided to par.draws. The first matrix containing the parameter draws for the alternative specific parameters. The second matrix containing the draws for the rest of the parameters.

The list of potential choice sets are created using combn. The weights argument can be used when the par.draws have weights. This is for example the case when parameter values are updated using ImpsampMNL.

Value

set	Numeric matrix containing the choice set that maximizes the expected KL divergence.
kl	Numeric value which is the Kullback leibler divergence.

References

Crabbe M, Akinc D, Vandebroek M (2014). "Fast algorithms to generate individualized designs for the mixed logit choice model."

Examples

```
# KL efficient choice set, given parameter draws.
# Candidate profiles
cs <- Profiles(lvls = c(3, 3), coding = c("E", "E"))
m <- c(0.3, 0.2, -0.3, -0.2) # Prior mean (4 parameters).
pc <- diag(length(m)) # Prior variance
set.seed(123)
ps <- MASS::mvrnorm(n = 10, mu = m, Sigma = pc) # 10 draws.
# Efficient choice set to add.
SeqKL(cand.set = cs, n.alts = 2, alt.cte = NULL, par.draws = ps, weights = NULL)

# KL efficient choice set, given parameter draws.
# Candidate profiles
cs <- Profiles(lvls = c(3, 3), coding = c("C", "E"), c.lvls = list(c(5,3,1)))
m <- c(0.7, 0.3, -0.3, -0.2) # Prior mean (4 parameters).
pc <- diag(length(m)) # Prior variance
set.seed(123)
ps <- MASS::mvrnorm(n = 10, mu = m, Sigma = pc) # 10 draws.
sample <- list(ps[ , 1], ps[ , 2:4])
ac <- c(1, 0) # Alternative specific constant.
# Efficient choice set to add.
SeqKL(cand.set = cs, n.alts = 2, alt.cte = ac, par.draws = sample, weights = NULL)
```

SeqMOD *Sequential modified federov algorithm for MNL model.*

Description

Selects the choice set that minimizes the DB-error when added to an initial design, given (updated) parameter values.

Usage

```
SeqMOD(
  des = NULL,
  cand.set,
  n.alts,
  par.draws,
  prior.covar,
  alt.cte = NULL,
  no.choice = NULL,
  weights = NULL,
  parallel = TRUE,
```

```
  reduce = TRUE,
  allow.rep = FALSE
)
```

Arguments

des	A design matrix in which each row is a profile. If alternative specific constants are present, those should be included as the first column(s) of the design. Can be generated with Modfed or CEA.
cand.set	A numeric matrix in which each row is a possible profile. The Profiles function can be used to generate this matrix.
n.alts	Numeric value indicating the number of alternatives per choice set.
par.draws	A matrix or a list, depending on alt.cte.
prior.covar	Covariance matrix of the prior distribution.
alt.cte	A binary vector indicating for each alternative whether an alternative specific constant is desired. The default is NULL.
no.choice	An integer indicating the no choice alternative. The default is NULL.
weights	A vector containing the weights of the draws. Default is NULL, See also ImpsampMNL.
parallel	Logical value indicating whether computations should be done over multiple cores.
reduce	Logical value indicating whether the candidate set should be reduced or not.
allow.rep	Logical value indicating whether repeated choice sets are allowed in the design.

Details

This algorithm is ideally used in an adaptive context. The algorithm will select the next DB-efficient choice set given parameter values and possible previously generated choice sets. In an adaptive context these parameter values are updated after each observed response.

Previously generated choice sets, which together form an initial design, can be provided in des. When no design is provided, the algorithm will select te most efficient choice set based on the fisher information of the prior covariance matrix prior.covar.

If alt.cte = NULL, par.draws should be a matrix in which each row is a sample from the multivariate parameter distribution. In case that alt.cte is not NULL, a list containing two matrices should be provided to par.draws. The first matrix containing the parameter draws for the alternative specific parameters. The second matrix containing the draws for the rest of the parameters.

The list of potential choice sets are created using combn. If reduce is TRUE, allow.rep = FALSE and vice versa. Furthermore, the list of potential choice sets will be screaned in order to select only those choice sets with a unique information matrix. If no alternative specific constants are used, reduce should always be TRUE. When alternative specific constants are used reduce can be TRUE so that the algorithm will be faster, but the combinations of constants and profiles will not be evaluated exhaustively.

The weights argument can be used when the par.draws have weights. This is for example the case when parameter values are updated using ImpsampMNL.

When parallel is TRUE, detectCores will be used to decide upon the number of available cores. That number minus 1 cores will be used to search for the optimal choice set. For small problems

(6 parameters), `parallel = TRUE` can be slower. For larger problems the computation time will decrease significantly.

Note: this function is more stable than SeqCEA, but it takes more time to get the output. This happens because this function makes an exhaustive search to get the choice set, whereas SeqCEA makes a random search.

Value

`set`	A matrix representing a DB efficient choice set.
`error`	A numeric value indicating the DB-error of the whole design.

References

Traets F, Sanchez G, Vandebroek M (2020). "Generating Optimal Designs for Discrete Choice Experiments in R: The idefix Package." *Journal of Statistical Software*, **96**(3).

Yu J, Goos P, Vandebroek M (2011). "Individually adapted sequential Bayesian conjoint-choice designs in the presence of consumer heterogeneity."

Examples

```
# DB efficient choice set, given a design and parameter draws.
# Candidate profiles
cs <- Profiles(lvls = c(3, 3, 3), coding = c("D", "D", "D"))
m <- c(0.3, 0.2, -0.3, -0.2, 1.1, 2.4) # mean (total = 6 parameters).
pc <- diag(length(m)) # covariance matrix
set.seed(123)
sample <- MASS::mvrnorm(n = 10, mu = m, Sigma = pc)
# Initial design.
des <- example_design
# Efficient choice set to add.
SeqMOD(des = des, cand.set = cs, n.alts = 2, par.draws = sample,
           prior.covar = pc, parallel = FALSE)

# DB efficient choice set, given parameter draws.
# with alternative specific constants
des <- example_design2
cs <- Profiles(lvls = c(3, 3, 3), coding = c("D", "D", "D"))
ac <- c(1, 1, 0) # Alternative specific constants.
m <- c(0.3, 0.2, -0.3, -0.2, 1.1, 2.4, 1.8, 1.2) # mean
pc <- diag(length(m)) # covariance matrix
pos <- MASS::mvrnorm(n = 10, mu = m, Sigma = pc)
sample <- list(pos[ , 1:2], pos[ , 3:8])
# Efficient choice set.
SeqMOD(des = des, cand.set = cs, n.alts = 3, par.draws = sample, alt.cte = ac,
           prior.covar = pc, parallel = FALSE)
```

SurveyApp *Shiny application to generate a discrete choice survey.*

Description

This function starts a shiny application which puts choice sets on screen and saves the responses. The complete choice design can be provided in advance, or can be generated sequentially adaptively, or can be a combination of both.

Usage

```
SurveyApp(
  des = NULL,
  n.total,
  alts,
  atts,
  lvl.names,
  coding,
  alt.cte = NULL,
  no.choice = NULL,
  buttons.text,
  intro.text,
  end.text,
  data.dir = NULL,
  c.lvls = NULL,
  prior.mean = NULL,
  prior.covar = NULL,
  cand.set = NULL,
  n.draws = NULL,
  lower = NULL,
  upper = NULL,
  parallel = TRUE,
  reduce = TRUE
)
```

Arguments

des	A numeric matrix which represents the design matrix. Each row is a profile.
n.total	A numeric value indicating the total number of choice sets.
alts	A character vector containing the names of the alternatives.
atts	A character vector containing the names of the attributes.
lvl.names	A list containing character vectors with the values of each level of each attribute.
coding	A character vector denoting the type of coding used for each attribute. See also Profiles.
alt.cte	A binary vector indicating for each alternative if an alternative specific constant is present. The default is NULL.

no.choice	An integer indicating which alternative should be a no choice alternative. The default is NULL.
buttons.text	A string containing the text presented together with the option buttons.
intro.text	A string containing the text presented before the choice survey.
end.text	A string containing the text presented after the choice survey.
data.dir	A character string with the directory denoting where the data needs to be written. The default is NULL
c.lvls	A list containing numeric vectors with the attribute levels for each continuous attribute. The default is NULL.
prior.mean	Numeric vector indicating the mean of the multivariate normal distribution (prior).
prior.covar	Covariance matrix of the prior distribution.
cand.set	A numeric matrix in which each row is a possible profile. The Profiles function can be used to generate this matrix.
n.draws	Numeric value indicating the number of draws.
lower	Numeric vector of lower truncation points, the default is NULL.
upper	Numeric vector of upper truncation points, the default is NULL.
parallel	Logical value indicating whether computations should be done over multiple cores. The default is TRUE.
reduce	Logical value indicating whether the candidate set should be reduced or not.

Details

A pregenerated design can be specified in des. This should be a matrix in which each row is a profile. This can be generated with Modfed or CEA, but it is not necessary.

If n.total = nrow(des) / length(alts), the specified design will be put on screen, one set after the other, and the responses will be saved. If n.total > (nrow(des) / length(alts)), first the specified design will be shown and afterwards the remaining sets will be generated adaptively. If des = NULL, n.total sets will be generated adaptively. See SeqMOD for more information on adaptive choice sets.

Whenever adaptive sets will be generated, prior.mean, prior.covar, cand.set and n.draws, should be specified. These arguments are necessary for the underlying importance sampling algorithm to update the prior preference distribution. lower and upper can be used to specify lower and upper truncation points. See ImpsampMNL for more details.

The names specified in alts will be used to label the choice alternatives. The names specified in atts will be used to name the attributes in the choice sets. The values of lvl.names will be used to create the values in the choice sets. See Decode for more details.

The text specified in buttons.text will be displayed above the buttons to indicate the preferred choice (for example: "indicate your preferred choice"). The text specified in intro.text will be displayed before the choice sets. This will generally be a description of the survey and some instructions. The text specified in end.text will be displayed after the survey. This will generally be a thanking note and some further instructions.

A no choice alternative is coded as an alternative with 1 alternative specific constant and zero's for all other attribute levels. If a no choice alternative is present in des, or is desired when generating

adaptive choice sets, no.choice should be specified. This should be done with an integer, indicating which alternative is the no choice option. This alternative will not be presented on screen, but the option to select "no choice" will be. The alt.cte argument should be specified accordingly, namely with a 1 on the location of the no.choice option. See examples for illustration.

When parallel is TRUE, detectCores will be used to decide upon the number of available cores. That number minus 1 cores will be used to search for the optimal adaptive choice set. For small problems (6 parameters), parallel = TRUE can be slower. For larger problems the computation time will decrease significantly.

When reduce = TRUE, the set of all potential choice sets will be reduced to choice sets that have a unique information matrix. If no alternative specific constants are used, reduce should always be TRUE. When alternative specific constants are used reduce can be TRUE so that the algorithm will be faster, but the combinations of constants and profiles will not be evaluated exhaustively.

Value

After completing the survey, two text files can be found in data.dir. The file with "num" in the filename is a matrix with the numeric choice data. The coded design matrix ("par"), presented during the survey, together with the observed responses ("resp") can be found here. Rownames indicate the setnumbers. The file with "char" in the filename is a matrix with character choice data. The labeled design matrix ("par"), presented during the survey, together with the observed responses ("resp") can be found here. See LoadData to load the data.

References

Yu J, Goos P, Vandebroek M (2011). "Individually adapted sequential Bayesian conjoint-choice designs in the presence of consumer heterogeneity."

Examples

```
## Not run:
#### Present choice design without adaptive sets (n.total = sets in des)
# example design
data("example_design") # pregenerated design
xdes <- example_design
### settings of the design
code <- c("D", "D", "D")
n.sets <- 8
# settings of the survey
alternatives <- c("Alternative A", "Alternative B")
attributes <- c("Price", "Time", "Comfort")
labels <- vector(mode="list", length(attributes))
labels[[1]] <- c("$10", "$5", "$1")
labels[[2]] <- c("20 min", "12 min", "3 min")
labels[[3]] <- c("bad", "average", "good")
i.text <- "Welcome, here are some instructions ... good luck!"
b.text <- "Please choose the alternative you prefer"
e.text <- "Thanks for taking the survey"
dataDir <- getwd()
# Display the survey
SurveyApp (des = xdes, n.total = n.sets, alts = alternatives,
```

```
          atts = attributes, lvl.names = labels, coding = code,
          buttons.text = b.text, intro.text = i.text, end.text = e.text)

#### Present choice design with partly adaptive sets (n.total > sets in des)
# example design
data("example_design") # pregenerated design
xdes <- example_design
### settings of the design
code <- c("D", "D", "D")
n.sets <- 12
# settings of the survey
alternatives <- c("Alternative A", "Alternative B")
attributes <- c("Price", "Time", "Comfort")
labels <- vector(mode="list", length(attributes))
labels[[1]] <- c("$10", "$5", "$1")
labels[[2]] <- c("20 min", "12 min", "3 min")
labels[[3]] <- c("bad", "average", "good")
i.text <- "Welcome, here are some instructions ... good luck!"
b.text <- "Please choose the alternative you prefer"
e.text <- "Thanks for taking the survey"
# setting for adaptive sets
levels <- c(3, 3, 3)
cand <- Profiles(lvls = levels, coding = code)
p.mean <- c(0.3, 0.7, 0.3, 0.7, 0.3, 0.7)
p.var <- diag(length(p.mean))
dataDir <- getwd()
# Display the survey
SurveyApp(des = xdes, n.total = n.sets, alts = alternatives,
          atts = attributes, lvl.names = labels, coding = code,
          buttons.text = b.text, intro.text = i.text, end.text = e.text,
          prior.mean = p.mean, prior.covar = p.var, cand.set = cand,
          n.draws = 50)

#### Choice design with only adaptive sets (des=NULL)
# setting for adaptive sets
levels <- c(3, 3, 3)
p.mean <- c(0.3, 0.7, 0.3, 0.7, 0.3, 0.7)
low = c(-Inf, -Inf, -Inf, 0, 0, -Inf)
up = rep(Inf, length(p.mean))
p.var <- diag(length(p.mean))
code <- c("D", "D", "D")
cand <- Profiles(lvls = levels, coding = code)
n.sets <- 12
# settings of the survey
alternatives <- c("Alternative A", "Alternative B")
attributes <- c("Price", "Time", "Comfort")
labels <- vector(mode="list", length(attributes))
labels[[1]] <- c("$10", "$5", "$1")
labels[[2]] <- c("20 min", "12 min", "3 min")
labels[[3]] <- c("bad", "average", "good")
i.text <- "Welcome, here are some instructions ... good luck!"
b.text <- "Please choose the alternative you prefer"
e.text <- "Thanks for taking the survey"
```

```
dataDir <- getwd()
# Display the survey
SurveyApp(des = NULL, n.total = n.sets, alts = alternatives,
          atts = attributes, lvl.names = labels, coding = code,
          buttons.text = b.text, intro.text = i.text, end.text = e.text,
          prior.mean = p.mean, prior.covar = p.var, cand.set = cand,
          lower = low, upper = up, n.draws = 50)
# If CEA algorithm is desired, cand.set argument is not needed
SurveyApp(des = NULL, n.total = n.sets, alts = alternatives,
         atts = attributes, lvl.names = labels, coding = code,
         buttons.text = b.text, intro.text = i.text, end.text = e.text,
         prior.mean = p.mean, prior.covar = p.var,
         lower = low, upper = up, n.draws = 50)

#### Present choice design with a no choice alternative.
# example design
data("nochoice_design") # pregenerated design
xdes <- nochoice_design
### settings of the design
code <- c("D", "D", "D")
n.sets <- 8
# settings of the survey
alternatives <- c("Alternative A", "Alternative B", "None")
attributes <- c("Price", "Time", "Comfort")
labels <- vector(mode = "list", length(attributes))
labels[[1]] <- c("$10", "$5", "$1")
labels[[2]] <- c("20 min", "12 min", "3 min")
labels[[3]] <- c("bad", "average", "good")
i.text <- "Welcome, here are some instructions ... good luck!"
b.text <- "Please choose the alternative you prefer"
e.text <- "Thanks for taking the survey"

# Display the survey
SurveyApp(des = xdes, n.total = n.sets, alts = alternatives,
          atts = attributes, lvl.names = labels, coding = code,
          buttons.text = b.text, intro.text = i.text, end.text = e.text,
          no.choice = 3, alt.cte = c(0, 0, 1))

## End(Not run)
```

Index