# Package: cbcTools (via r-universe)

September 3, 2024

**Title** Choice-Based Conjoint Experiment Design Generation and Power Evaluation in R

**Version** 0.5.2

**Maintainer** John Helveston <john.helveston@gmail.com>

**Description** Design and evaluate choice-based conjoint survey experiments. Generate a variety of survey designs, including random designs, full factorial designs, orthogonal designs, D-optimal designs, and Bayesian D-efficient designs as well as designs with ``no choice'' options and ``labeled'' (also known as ``alternative specific'') designs. Conveniently inspect the design balance and overlap, and simulate choice data for a survey design either randomly or according to a multinomial or mixed logit utility model defined by user-provided prior parameters. Conduct a power analysis for a given survey design by estimating the same model on different subsets of the data to simulate different sample sizes. Full factorial and orthogonal designs are obtained using the 'DoE.base' package (Grömping, 2018) <doi:10.18637/jss.v085.i05>. D-optimal designs are obtained using the 'AlgDesign' package (Wheeler, 2022) <https://CRAN.R-project.org/package=AlgDesign>. Bayesian D-efficient designs are obtained using the 'idefix' package (Traets et al, 2020) <doi:10.18637/jss.v096.i03>. Choice simulation and model estimation in power analyses are handled using the 'logitr' package (Helveston, 2023) <doi:10.18637/jss.v105.i10>.

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.2.3

**VignetteBuilder** knitr

**Depends** R (>= 3.5.0)

**Suggests** here, knitr, testthat, tibble

**Imports** AlgDesign, DoE.base, fastDummies, ggplot2, idefix, logitr (>= 1.0.1), MASS, parallel, randtoolbox, rlang, stats, utils

**URL** https://github.com/jhelvy/cbcTools, https://jhelvy.github.io/cbcTools/

**BugReports** https://github.com/jhelvy/cbcTools/issues

**Repository** https://jhelvy.r-universe.dev

**RemoteUrl** https://github.com/jhelvy/cbctools

**RemoteRef** HEAD

**RemoteSha** fb81dc4fd2a0dfdb0d9fc9de493764b91c3e4618

# Contents



---

`cbc_balance` *Counts of attribute balance*

---

## Description

This function prints out a summary of the individual and pairwise counts of each level for each attribute across all choice questions in the design.

## Usage

```
cbc_balance(design)
```

## Arguments

| | |
|---|---|
| `design` | A data frame of a survey design. |

## Value

Prints the individual and pairwise counts of the number of times each attribute levels in shown in the design.

## Examples

```
library(cbcTools)

# A simple conjoint experiment about apples

# Generate all possible profiles
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  type      = c("Fuji", "Gala", "Honeycrisp"),
  freshness = c('Poor', 'Average', 'Excellent')
)

# Make a survey design from all possible profiles
# (This is the default setting where method = 'full' for "full factorial")
design <- cbc_design(
  profiles = profiles,
  n_resp   = 300, # Number of respondents
  n_alts   = 3,   # Number of alternatives per question
  n_q      = 6    # Number of questions per respondent
)

# Inspect the design balance
cbc_balance(design)

# Inspect the design overlap
cbc_overlap(design)
```

---

# cbc_choices *Simulate choices for a survey design*

---

## Description

Simulate choices for a survey design, either randomly or according to a utility model defined by user-provided prior parameters. All choices are simulated using the 'logitr' package. For more details see the JSS article on the 'logitr' package (Helveston, 2023).

## Usage

```
cbc_choices(design, obsID = "obsID", priors = NULL, n_draws = 100)
```

## Arguments

| | |
|---|---|
| design | A data frame of a survey design. |
| obsID | The name of the column in design that identifies each choice observation. Defaults to "obsID". |
| priors | A list of one or more prior parameters that define a prior (assumed) utility model used to simulate choices for the survey data frame. If NULL (the default), choices will be randomly assigned. |
| n_draws | The number of Halton draws to use for simulated choices for mixed logit models. Defaults to 100. |

## Value

Returns the design data frame with an additional choice column identifying the simulated choices.

## References

Helveston, J. P. (2023). logitr: Fast Estimation of Multinomial and Mixed Logit Models with Preference Space and Willingness-to-Pay Space Utility Parameterizations. Journal of Statistical Software, 105(10), 1–37, doi:10.18637/jss.v105.i10

## Examples

```
library(cbcTools)

# A simple conjoint experiment about apples

# Generate all possible profiles
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  type      = c("Fuji", "Gala", "Honeycrisp"),
  freshness = c('Poor', 'Average', 'Excellent')
)

# Make a survey design from all possible profiles
# (This is the default setting where method = 'full' for "full factorial")
design <- cbc_design(
  profiles = profiles,
  n_resp   = 300, # Number of respondents
  n_alts   = 3,   # Number of alternatives per question
  n_q      = 6    # Number of questions per respondent
)

# Simulate random choices
data <- cbc_choices(
  design = design,
  obsID  = "obsID"
)

# Simulate choices according to a prior utility model
data <- cbc_choices(
```

```
  design = design,
  obsID = "obsID",
  priors = list(
    price     = 0.1,
    type      = c(0.1, 0.2),
    freshness = c(0.1, 0.2)
  )
)

# Simulate choices according to a prior model with interactions
data <- cbc_choices(
  design = design,
  obsID = "obsID",
  priors = list(
    price     = 0.1,
    type      = c(0.1, 0.2),
    freshness = c(0.1, 0.2),
    `price*type` = c(0.1, 0.5)
  )
)

# Simulate choices according to a prior utility model with random parameters
data <- cbc_choices(
  design = design,
  obsID = "obsID",
  priors = list(
    price = 0.1,
    type = randN(mean = c(0.1, 0.2), sd = c(1, 2)),
    freshness = c(0.1, 0.2)
  )
)
```

---

cbc_design *Make a choice-based conjoint survey design*

---

## Description

This function creates a data frame containing a choice-based conjoint survey design where each row is an alternative. Generate a variety of survey designs, including full factorial designs, orthogonal designs, and Bayesian D-efficient designs as well as designs with "no choice" options and "labeled" (also known as "alternative specific") designs.

## Usage

```
cbc_design(
  profiles,
  n_resp,
  n_alts,
  n_q,
```

```
  n_blocks = 1,
  n_draws = 50,
  n_start = 5,
  no_choice = FALSE,
  label = NULL,
  method = "random",
  priors = NULL,
  prior_no_choice = NULL,
  probs = FALSE,
  keep_d_eff = FALSE,
  keep_db_error = FALSE,
  max_iter = 50,
  parallel = FALSE
)
```

## Arguments

| | |
|---|---|
| profiles | A data frame in which each row is a possible profile. This can be generated using the cbc_profiles() function. |
| n_resp | Number of survey respondents. |
| n_alts | Number of alternatives per choice question. |
| n_q | Number of questions per respondent. |
| n_blocks | Number of blocks used in Orthogonal or Bayesian D-efficient designs. Max allowable is one block per respondent. Defaults to 1, meaning every respondent sees the same choice set. |
| n_draws | Number of draws used in simulating the prior distribution used in Bayesian D-efficient designs. Defaults to 50. |
| n_start | A numeric value indicating the number of random start designs to use in obtaining a Bayesian D-efficient design. The default is 5. Increasing n_start can result in a more efficient design at the expense of increased computational time. |
| no_choice | Include a "no choice" option in the choice sets? Defaults to FALSE. If TRUE, the total number of alternatives per question will be one more than the provided n_alts argument. |
| label | The name of the variable to use in a "labeled" design (also called an "alternative-specific design") such that each set of alternatives contains one of each of the levels in the label attribute. Currently not compatible with Bayesian D-efficient designs. If used, the n_alts argument will be ignored as its value is defined by the unique number of levels in the label variable. Defaults to NULL. |
| method | Choose the design method to use: "random", "full", "orthogonal", "dopt", "CEA", or "Modfed". Defaults to "random". See details below for complete description of each method. |
| priors | A list of one or more assumed prior parameters used to generate a Bayesian D-efficient design. Defaults to NULL |
| prior_no_choice | Prior utility value for the "no choice" alternative. Only required if no_choice = TRUE. Defaults to NULL. |

| | |
|---|---|
| probs | If TRUE, for Bayesian D-efficient designs the resulting design includes average predicted probabilities for each alternative in each choice set given the sample from the prior preference distribution. Defaults to FALSE.' |
| keep_d_eff | If TRUE, for D-optimal designs (method = "dopt") the returned object will be a list containing the design and the D-efficiency score. Defaults to FALSE. |
| keep_db_error | If TRUE, for Bayesian D-efficient designs the returned object will be a list containing the design and the DB-error score. Defaults to FALSE. |
| max_iter | A numeric value indicating the maximum number allowed iterations when searching for a Bayesian D-efficient design. The default is 50. |
| parallel | Logical value indicating whether computations should be done over multiple cores. The default is FALSE. |

## Details

The method argument determines the design method used. Options are:

- "random"
- "full"
- "orthogonal"
- "dopt"
- "CEA"
- "Modfed"

All methods ensure that the two following criteria are met:

1. No two profiles are the same within any one choice set.
2. No two choice sets are the same within any one respondent.

The table below summarizes method compatibility with other design options, including the ability to include a "no choice" option, the creation of a "labeled" design (also called a "alternative-specific" design), the use of restricted profile, and the use of blocking.

| Method | Include "no choice"? | Labeled designs? | Restricted profiles? | Blocking? |
|---|---|---|---|---|
| "random" | Yes | Yes | Yes | No |
| "full" | Yes | Yes | Yes | Yes |
| "orthogonal" | Yes | No | No | Yes |
| "dopt" | Yes | No | Yes | Yes |
| "CEA" | Yes | No | No | Yes |
| "Modfed" | Yes | No | Yes | Yes |

The "random" method (the default) creates a design where choice sets are created by randomly sampling from the full set of profiles *with *replacement. This means that few (if any) respondents will see the same sets of choice sets. This method is less efficient than other approaches and may lead to a deficient experiment in smaller sample sizes, though it guarantees equal ability to estimate main and interaction effects.

The "full" method for ("full factorial") creates a design where choice sets are created by randomly sampling from the full set of profiles *without replacement*. The choice sets are

then repeated to meet the desired number of survey respondents (determined by n_resp). If blocking is used, choice set blocks are created using mutually exclusive subsets of profiles within each block. This method produces a design with similar performance with that of the "random" method, except the choice sets are repeated and thus there will be many more opportunities for different respondents to see the same choice sets. This method is less efficient than other approaches and may lead to a deficient experiment in smaller sample sizes, though it guarantees equal ability to estimate main and interaction effects. For more information about blocking with full factorial designs, see ?DoE.base::fac.design as well as the JSS article on the DoE.base package (Grömping, 2018).

The "orthogonal" method creates a design where an orthogonal array from the full set of profiles is found and then choice sets are created by randomly sampling from this orthogonal array *without replacement*. The choice sets are then repeated to meet the desired number of survey respondents (determined by n_resp). If blocking is used, choice set blocks are created using mutually exclusive subsets of the orthogonal array within each block. For cases where an orthogonal array cannot be found, a full factorial design is used. This approach is also sometimes called a "main effects" design since orthogonal arrays focus the information on the main effects at the expense of information about interaction effects. For more information about orthogonal designs, see ?DoE.base::oa.design as well as the JSS article on the DoE.base package (Grömping, 2018).

The "dopt" method creates a "D-optimal" design where an array from profiles is found that maximizes the D-efficiency of a linear model using the Federov algorithm, with the total number of unique choice sets determined by n_q*n_blocks. Choice sets are then created by randomly sampling from this array *without replacement*. The choice sets are then repeated to meet the desired number of survey respondents (determined by n_resp). If blocking is used, choice set blocks are created from the D-optimal array. For more information about the underlying algorithm for this method, see ?AlgDesign::optFederov.

The "CEA" and "Modfed" methods use the specified priors to create a Bayesian D-efficient design for the choice sets, with the total number of unique choice sets determined by n_q*n_blocks. The choice sets are then repeated to meet the desired number of survey respondents (determined by n_resp). If "CEA" or "Modfed" is used without specifying priors, a prior of all 0s will be used and a warning message stating this will be shown. In the opposite case, if priors are specified but neither Bayesian method is used, the "CEA" method will be used and a warning stating this will be shown. Restricted sets of profiles can only be used with "Modfed". For more details on Bayesian D-efficient designs, see ?idefix::CEA and ?idefix::Modfed as well as the JSS article on the idefix package (Traets et al, 2020).

## Value

The returned design data frame contains a choice-based conjoint survey design where each row is an alternative. It includes the following columns:

- profileID: Identifies the profile in profiles.
- respID: Identifies each survey respondent.
- qID: Identifies the choice question answered by the respondent.
- altID:Identifies the alternative in any one choice observation.
- obsID: Identifies each unique choice observation across all respondents.
- blockID: If blocking is used, identifies each unique block.

## References

Grömping, U. (2018). R Package DoE.base for Factorial Experiments. Journal of Statistical Software, 85(5), 1–41 doi:10.18637/jss.v085.i05

Traets, F., Sanchez, D. G., & Vandebroek, M. (2020). Generating Optimal Designs for Discrete Choice Experiments in R: The idefix Package. Journal of Statistical Software, 96(3), 1–41, doi:10.18637/jss.v096.i03

Wheeler B (2022)._AlgDesign: Algorithmic Experimental Design. R package version 1.2.1, https://CRAN.R-project.org/package=AlgDesign.

## Examples

```
library(cbcTools)

# A simple conjoint experiment about apples

# Generate all possible profiles
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  type      = c("Fuji", "Gala", "Honeycrisp"),
  freshness = c('Poor', 'Average', 'Excellent')
)

# Make a survey by randomly sampling from all possible profiles
# (This is the default setting where method = 'random')
design_random <- cbc_design(
  profiles = profiles,
  n_resp   = 100, # Number of respondents
  n_alts   = 3,   # Number of alternatives per question
  n_q      = 6    # Number of questions per respondent
)

# Make a survey using a full factorial design and include a "no choice" option
design_full <- cbc_design(
  profiles = profiles,
  n_resp   = 100, # Number of respondents
  n_alts   = 3,   # Number of alternatives per question
  n_q      = 6,   # Number of questions per respondent
  method   = 'full', # Change this to use a different method, e.g. 'orthogonal', or 'dopt'
  no_choice = TRUE
)

# Make a survey by randomly sampling from all possible profiles
# with each level of the "type" attribute appearing as an alternative
design_random_labeled <- cbc_design(
  profiles = profiles,
  n_resp   = 100, # Number of respondents
  n_alts   = 3,   # Number of alternatives per question
  n_q      = 6,   # Number of questions per respondent
  label    = "type"
)
```

```
# Make a Bayesian D-efficient design with a prior model specified
# Note that by speed can be improved by setting parallel = TRUE
design_bayesian <- cbc_design(
    profiles  = profiles,
    n_resp    = 100, # Number of respondents
    n_alts    = 3,   # Number of alternatives per question
    n_q       = 6,   # Number of questions per respondent
    n_start   = 1,   # Defaults to 5, set to 1 here for a quick example
    priors = list(
        price     = -0.1,
        type      = c(0.1, 0.2),
        freshness = c(0.1, 0.2)
    ),
    method = "CEA",
    parallel = FALSE
)
```

---

`cbc_overlap` *Counts of attribute overlap*

---

## Description

This function prints out a summary of the amount of "overlap" across attributes within the choice questions. For example, for each attribute, the count under `"1"` is the number of choice questions in which the same level was shown across all alternatives for that attribute (because there was only one level shown). Likewise, the count under `"2"` is the number of choice questions in which only two unique levels of that attribute were shown, and so on.

## Usage

```
cbc_overlap(design)
```

## Arguments

| | |
|---|---|
| `design` | A data frame of a survey design. |

## Value

Prints the counts of the number of choice questions that contain the unique number of levels for each attribute.

## Examples

```
library(cbcTools)

# A simple conjoint experiment about apples

# Generate all possible profiles
profiles <- cbc_profiles(
```

```
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  freshness = c("Excellent", "Average", "Poor"),
  type      = c("Fuji", "Gala", "Honeycrisp")
)

# Make a randomized survey design
design <- cbc_design(
  profiles = profiles,
  n_resp   = 300, # Number of respondents
  n_alts   = 3, # Number of alternatives per question
  n_q      = 6 # Number of questions per respondent
)

# Inspect the design balance
cbc_balance(design)

# Inspect the design overlap
cbc_overlap(design)
```

---

cbc_power *Estimate the same model on different size subsets of data*

---

## Description

This function estimates the same model multiple times using different size subsets of a set of choice data and then returns a data frame of the estimated model coefficients and standard errors for each sample size. This is useful for determining the required sample size for obtaining a desired level of statistical power on each coefficient. The number of models to estimate is set by the nbreaks argument, which breaks up the data into groups of increasing sample sizes. All models are estimated models using the 'logitr' package. For more details see the JSS article on the 'logitr' package (Helveston, 2023).

## Usage

```
cbc_power(
  data,
  outcome,
  obsID,
  pars,
  randPars = NULL,
  nbreaks = 10,
  n_q = 1,
  return_models = FALSE,
  panelID = NULL,
  clusterID = NULL,
  robust = FALSE,
  predict = FALSE,
  n_cores = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| data | The data, formatted as a data.frame object. |
| outcome | The name of the column that identifies the outcome variable, which should be coded with a 1 for TRUE and 0 for FALSE. |
| obsID | The name of the column that identifies each observation. |
| pars | The names of the parameters to be estimated in the model. Must be the same as the column names in the data argument. |
| randPars | A named vector whose names are the random parameters and values the distribution: 'n' for normal or 'ln' for log-normal. Defaults to NULL. |
| nbreaks | The number of different sample size groups. |
| n_q | Number of questions per respondent. Defaults to 1 if not specified. |
| return_models | If TRUE, a list of all estimated models is returned. This can be useful if you want to extract other outputs from each model, such as the variance-covariance matrix, etc. Defaults to FALSE. |
| panelID | The name of the column that identifies the individual (for panel data where multiple observations are recorded for each individual). Defaults to NULL. |
| clusterID | The name of the column that identifies the cluster groups to be used in model estimation. Defaults to NULL. |
| robust | Determines whether or not a robust covariance matrix is estimated. Defaults to FALSE. Specification of a clusterID will override the user setting and set this to 'TRUE' (a warning will be displayed in this case). Replicates the functionality of Stata's cmcmmixlogit. |
| predict | If TRUE, predicted probabilities, fitted values, and residuals are also included in the returned model objects. Defaults to FALSE. |
| n_cores | The number of cores to use for parallel processing. Set to 1 to run serially Defaults to NULL, in which case the number of cores is set to parallel::detectCores() - 1. Max cores allowed is capped at parallel::detectCores(). |
| ... | Other arguments that are passed to logitr::logitr() for model estimation. See the logitr documentation for details about other available arguments. |

## Value

Returns a data frame of estimated model coefficients and standard errors for the same model estimated on subsets of the data with increasing sample sizes.

## References

Helveston, J. P. (2023). logitr: Fast Estimation of Multinomial and Mixed Logit Models with Preference Space and Willingness-to-Pay Space Utility Parameterizations. Journal of Statistical Software, 105(10), 1–37, doi:10.18637/jss.v105.i10

## Examples

```
library(cbcTools)

# A simple conjoint experiment about apples

# Generate all possible profiles
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  type      = c("Fuji", "Gala", "Honeycrisp"),
  freshness = c('Poor', 'Average', 'Excellent')
)

# Make a survey design from all possible profiles
# (This is the default setting where method = 'full' for "full factorial")
design <- cbc_design(
  profiles = profiles,
  n_resp   = 300, # Number of respondents
  n_alts   = 3,   # Number of alternatives per question
  n_q      = 6    # Number of questions per respondent
)

# Simulate random choices
data <- cbc_choices(
  design = design,
  obsID  = "obsID"
)

# Conduct a power analysis
power <- cbc_power(
  data    = data,
  pars    = c("price", "type", "freshness"),
  outcome = "choice",
  obsID   = "obsID",
  nbreaks = 10,
  n_q     = 6,
  n_cores = 2
)
```

---

`cbc_profiles` *Make a data frame of all combinations of attribute levels*

---

## Description

This function creates a data frame of of all possible combinations of attribute levels.

## Usage

```
cbc_profiles(...)
```

### Arguments

| | |
|---|---|
| `...` | Any number of named vectors defining each attribute and their levels, e.g. `price = c(1, 2, 3)`. Separate each vector by a comma. |

### Value

A data frame of all possible combinations of attribute levels.

### Examples

```
library(cbcTools)

# Generate all profiles for a simple conjoint experiment about apples
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  type      = c("Fuji", "Gala", "Honeycrisp"),
  freshness = c('Poor', 'Average', 'Excellent')
)
```

---

## cbc_restrict *Obtain a restricted set of profiles*

---

### Description

This function returns a restricted set of profiles as a data frame.

### Usage

```
cbc_restrict(profiles, ...)
```

### Arguments

| | |
|---|---|
| `profiles` | A data frame in which each row is a possible profile. This can be generated using the `cbc_profiles()` function. |
| `...` | Any number of restricted pairs of attribute levels, defined as pairs of logical expressions separated by commas. For example, the restriction `type == 'Fuji' & freshness == 'Poor'` will eliminate profiles such that `"Fuji"` type apples will never be shown with `"Poor"` freshness. |

### Value

A restricted set of profiles as a data frame.

## Examples

```
library(cbcTools)

# Generate all profiles for a simple conjoint experiment about apples
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  type      = c("Fuji", "Gala", "Honeycrisp"),
  freshness = c('Poor', 'Average', 'Excellent')
)

# Obtain a restricted subset of profiles based on pairs of logical
# expressions. The example below contains the following restrictions:

# - `"Gala"` apples will not be shown with the prices `1.5`, `2.5`, & `3.5`.
# - `"Honeycrisp"` apples will not be shown with prices less than `2`.
# - `"Honeycrisp"` apples will not be shown with the `"Poor"` freshness.
# - `"Fuji"` apples will not be shown with the `"Excellent"` freshness.

profiles_restricted <- cbc_restrict(
    profiles,
    type == "Gala" & price %in% c(1.5, 2.5, 3.5),
    type == "Honeycrisp" & price > 2,
    type == "Honeycrisp" & freshness == "Poor",
    type == "Fuji" & freshness == "Excellent"
)
```

---

# miscmethods.cbc_errors

*Methods for cbc_errors objects*

---

## Description

Miscellaneous methods for `cbc_errors` class objects.

## Usage

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

## Arguments

| | |
|---|---|
| `x` | is an object of class `cbc_errors`. |
| `...` | further arguments. |

## Value

Returns a ggplot2 object plotting standard errors versus sample size.

## Examples

```
library(cbcTools)

# A simple conjoint experiment about apples

# Generate all possible profiles
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5),
  type      = c("Fuji", "Gala", "Honeycrisp"),
  freshness = c('Poor', 'Average', 'Excellent')
)

# Make a survey design from all possible profiles
# (This is the default setting where method = 'full' for "full factorial")
design <- cbc_design(
  profiles = profiles,
  n_resp   = 300, # Number of respondents
  n_alts   = 3,   # Number of alternatives per question
  n_q      = 6    # Number of questions per respondent
)

# Simulate random choices
data <- cbc_choices(
  design = design,
  obsID  = "obsID"
)

# Conduct a power analysis
power <- cbc_power(
  data    = data,
  pars    = c("price", "type", "freshness"),
  outcome = "choice",
  obsID   = "obsID",
  nbreaks = 10,
  n_q     = 6
)

# Visualize the results
plot(power)
```

---

`miscmethods.cbc_models` *Methods for cbc_models objects*

---

## Description

Miscellaneous methods for `cbc_models` class objects.

### Usage

```
## S3 method for class 'cbc_models'
print(
  x,
  digits = max(3, getOption("digits") - 2),
  width = getOption("width"),
  ...
)
```

### Arguments

| | |
|---|---|
| x | is an object of class cbc_models. |
| digits | the number of digits for printing, defaults to 3. |
| width | the width of the printing. |
| ... | further arguments. |

### Value

No return value, prints a summary of estimated models.

---

## plot_compare_power *Plot a comparison of different design powers*

---

### Description

This function creates a ggplot2 object comparing the power curves of different designs. Each design is color coded and each facet (sub plot) is a model coefficient.

### Usage

```
plot_compare_power(...)
```

### Arguments

| | |
|---|---|
| ... | Any number of data frame containing power results obtained from the cbc_power() function, separated by commas. |

### Value

A plot comparing the power curves of different designs.

## Examples

```
## Not run:
library(cbcTools)

# Generate all possible profiles
profiles <- cbc_profiles(
  price     = c(1, 1.5, 2, 2.5, 3),
  type      = c("Fuji", "Gala", "Honeycrisp"),
  freshness = c('Poor', 'Average', 'Excellent')
)

# Make designs to compare: full factorial vs bayesian d-efficient
design_random <- cbc_design(
  profiles = profiles,
  n_resp = 100, n_alts = 3, n_q = 6
)
# Same priors will be used in bayesian design and simulated choices
priors <- list(
  price     = -0.1,
  type      = c(0.1, 0.2),
  freshness = c(0.1, 0.2)
)
design_bayesian <- cbc_design(
  profiles  = profiles,
  n_resp = 100, n_alts = 3, n_q = 6, n_start = 1, method = "CEA",
  priors = priors, parallel = FALSE
)

# Obtain power for each design by simulating choices
power_random <- design_random |>
cbc_choices(obsID = "obsID", priors = priors) |>
  cbc_power(
    pars = c("price", "type", "freshness"),
    outcome = "choice", obsID = "obsID", nbreaks = 5, n_q = 6, n_cores = 2
  )
power_bayesian <- design_bayesian |>
  cbc_choices(obsID = "obsID", priors = priors) |>
  cbc_power(
    pars = c("price", "type", "freshness"),
    outcome = "choice", obsID = "obsID", nbreaks = 5, n_q = 6, n_cores = 2
  )

# Compare power of each design
plot_compare_power(power_bayesian, power_random)

## End(Not run)
```

---

randLN *Define prior (assumed) model parameter as log-normally-distributed.*

---

## Description

Define prior (assumed) model parameter as log-normally-distributed. Used in the `cbc_choices()` function.

## Usage

```
randLN(mean = 0, sd = 1)
```

## Arguments

| | |
|---|---|
| `mean` | Mean of the distribution on the log scale, defaults to `0`. |
| `sd` | Standard deviation of the distribution on the log scale, defaults to `1`. |

## Value

A list defining log-normally-distributed parameters of the prior (assumed) utility model used to simulate choices in the `cbc_choices()` function.

## Examples

```
# Insert example
```

---

# `randN` *Define a prior (assumed) model parameter as normally-distributed.*

---

## Description

Define a prior (assumed) model parameter as normally-distributed. Used in the `cbc_choices()` function.

## Usage

```
randN(mean = 0, sd = 1)
```

## Arguments

| | |
|---|---|
| `mean` | Vector of means, defaults to `0`. |
| `sd` | Vector of standard deviations, defaults to `1`. |

## Value

A list defining normally-distributed parameters of the prior (assumed) utility model used to simulate choices in the `cbc_choices()` function.

## Examples

```
# Insert example
```

# Index