Journal of Statistical Software

November 2020, Volume 96, Issue 3. doi: 10.18637/jss.v096.i03

Generating Optimal Designs for Discrete Choice Experiments in R: The idefix Package

Frits Traets
KU Leuven

Daniel Gil Sanchez
KU Leuven

Martina Vandebroek
KU Leuven

Abstract

Discrete choice experiments are widely used in a broad area of research fields to capture the preference structure of respondents. The design of such experiments will determine to a large extent the accuracy with which the preference parameters can be estimated. This paper presents a new R package, called **idefix**, which enables users to generate optimal designs for discrete choice experiments. Besides Bayesian D-efficient designs for the multinomial logit model, the package includes functions to generate Bayesian adaptive designs which can be used to gather data for the mixed logit model. In addition, the package provides the necessary tools to set up actual surveys and collect empirical data. After data collection, **idefix** can be used to transform the data into the necessary format in order to use existing estimation software in R.

Keywords: optimal designs, discrete choice experiments, adaptive, mixed logit, **shiny**, R.

1. Introduction

Discrete choice experiments (DCEs) are used to gather stated preference data. In a typical DCE, the respondent is presented several choice sets. In each choice set the respondent is asked to choose between two or more alternatives in which each alternative consists of specific attribute levels. By analyzing the stated preference data, one is able to gain information on the preferences of respondents. The use of stated preference data, and DCEs in particular, has strongly increased the past decades in fields such as transportation, environmental economics, health economics, and marketing.

To analyze stated preference data, a choice model needs to be assumed. The large majority of discrete choice models is derived under the assumption of utility-maximizing behavior by the decision maker (Marschak 1950). This family of models is known as random utility maximization (RUM) models, and the most well known members are the multinomial logit

model (MNL; McFadden 1974), and the mixed logit model (MIXL; Hensher and Greene 2003; McFadden and Train 2000; Train 2003). The idea behind such models is that people maximize their utility, which is modeled as a function of the preference weights and attribute levels. The deterministic part of the utility is most often linearly specified in the parameters, but the corresponding logit probabilities relate nonlinearly to the observed utility.

When setting up a DCE, the researcher is usually restricted by the limited number of choice sets they can present to a respondent, and the limited number of respondents they can question. Therewith, the set of all possible choice sets one could present (i.e., the full factorial design) increases rapidly by including more attributes and attribute levels. The researcher is therefore obliged to make a selection of the choice sets to be included in the experimental design.

At first, choice designs were built using concepts from the general linear model design literature, neglecting the fact that most choice models are nonlinear (Huber and Zwerina 1996). Afterwards different design approaches have been proposed focusing on one or more design properties such as: utility balance, task complexity, response efficiency, attribute balance, and statistical efficiency (for an overview see Rose and Bliemer 2014). Where orthogonal designs were mostly used at first, statistically optimal designs have now acquired a prominent place in the literature on discrete choice experiments (Johnson *et al.* 2013). The latter aims to select those choice sets that force the respondent to make trade-offs, hereby maximizing the information gained from each observed choice, or alternatively phrased, to minimize the confidence ellipsoids around the parameter estimates.

Optimal designs maximize the expected Fisher information. For choice designs, this information depends on the parameters of the assumed choice model. Consequently, the efficiency of a choice design is related to the accuracy of the guess of the true parameters before conducting the experiment. In order to reduce this sensitivity, Bayesian efficient designs were developed where the researcher acknowledges their uncertainty about the true parameters by specifying a prior preference distribution. It has been repeatedly proven that optimal designs outperform other design approaches when the researcher has sufficient information on the preference structure a priori (Kessels, Goos, and Vandebroek 2006; Rose, Bliemer, Hensher, and Collins 2008; Rose and Bliemer 2009). On the other hand, several researchers have pointed out that even Bayesian efficient designs are sensitive to a misspecification of the prior distribution (see Walker, Wang, Thorhauge, and Ben-Akiva 2018, for an elaborate study on the robustness of different design approaches).

Serial designs (Bliemer and Rose 2010a), and individually adaptive sequential Bayesian (IASB) designs (Yu, Goos, and Vandebroek 2011; Crabbe, Akinc, and Vandebroek 2014) have gone one step further in order to ensure that the prior guess is reasonable by sequentially updating it during the survey. In serial efficient designs, the design is changed across respondents based on the updated information. With the IASB approach individual designs are constructed during the survey within each respondent, based on the posterior distribution of the individual preference parameters. By sequentially updating this distribution, those methods have proven to be less vulnerable to a misspecification of the initial prior. In addition, the IASB approach has been shown to perform especially well when preference heterogeneity is large and to result in qualitative designs for estimating the MIXL model (Danthurebandara, Yu, and Vandebroek 2011; Yu *et al.* 2011).

So far, no R (R Core Team 2020) package is suited to generate optimal designs for DCEs.

In addition, there is no available software that implements any kind of adaptive designs for DCEs. Some general experimental design R packages exist, e.g., **AlgDes** (Wheeler 2019) and **OptimalDesign** (Harman and Filova 2019), which can be used for the generation of optimal designs. However, none of them can be used to apply methods designed for nonlinear models which are necessary for choice models (Train 2003). To our knowledge two R packages exist that are promoted for designing DCEs. The package **support.CEs** (Aizaki 2012) provides functions for generating orthogonal main-effect arrays, but does not support optimal designs for discrete choice models. The package **choiceDes** (Horne 2018), which depends on **AlgDes**, is able to generate D-optimal designs for linear models and makes use of DCE terminology but does not take into account the dependency on the unknown preference parameters. Furthermore, it is limited to effects coded designs, and does not allow the user to specify alternative specific constants. Such design packages are still often used in the context of DCEs, because some linearly optimized designs are also optimal for MNL models when the preference parameters are assumed to be zero.

We believe that efficient designs deserve a place in the toolbox of the DCE researcher and adaptive efficient designs appear to be a promising extension. Since implementing such procedures is time consuming we believe the current package can pave the way for researchers and practitioners to become familiar with these techniques. Therefore, the R package **idefix** (Traets 2020) available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=idefix` implements D-efficient, Bayesian D-efficient, and the IASB approach to generate optimal designs for the MNL and MIXL model. Furthermore, it provides functions that allow researchers to set up simulation studies in order to evaluate the performance of efficient and of adaptive designs for certain scenarios. In addition, a function is included that generates a **shiny** application (Chang, Cheng, Allaire, Xie, and McPherson 2020) in which designs are presented on screen, which supports both pregenerated and adaptive designs, and allows the researcher to gather empirical data. The data format of **idefix** can be easily transformed in order to use existing estimation packages in R such as package **bayesm** (Rossi 2019), **ChoiceModelR** (Sermas 2012), **RSGHB** (Dumont and Keller 2019), **mlogit** (Croissant 2020) and the **Rchoice** package (Sarrias 2016).

The outline of this paper is as follows: in the next section some guidance is provided on gathering and specifying prior information, essential to produce efficient designs. Section 3 explains how to generate statistically optimal designs for the MNL model using the R package **idefix**. In Section 4, the IASB methodology is discussed together with the functions related to that approach. Here one can also find an example of how the different functions can be combined to set up simulation studies. Section 5 describes the `SurveyApp` function which enables the researcher to gather empirical choice data by launching an interactive **shiny** application. In Section 6, it is shown how to transform the **idefix** data format into the format desired by the estimation package of interest. Lastly, in Section 7, we discuss planned future development of the package.

2. Guidelines on specifying an appropriate prior distribution

The advantage of an efficient design approach lies in the fact that a priori knowledge can be included. However, as recently pointed out by Walker *et al.* (2018), such designs can also become inefficient if the prior deviates much from the true parameter values. In order to ensure the robustness of the design, some thoughtfulness is thus needed when specifying a

prior. In most cases, if not all, the researcher has an idea about the sign and/or magnitude of the model coefficients before conducting the experiment. It is, however, strongly advised to collect additional information and to critically evaluate those beliefs in order to set up efficient and robust (choice) experiments. This is recommended for all algorithms provided in this package, including the sequential adaptive approach (see Section 4). We therefore include some guidelines on how to gather prior information, quantify this information, and how to verify whether the specified prior is conform the researcher's beliefs. Lastly we provide an example similar to Walker *et al.* (2018) in which we show how to evaluate the chosen prior regarding the robustness of a design.

2.1. Gathering prior information

We hereby list some methods that are commonly used to gather prior information:

Consult the literature

Often research concerning similar (choice) topics is available in the literature. The statistics of interest in economics frequently involve the amount of money someone is willing to pay (WTP) for a certain product or service. For example in transportation research, numerous choice experiments have been published on the value of time (VOT), in health economics the value of a statistical life (VOSL) is of interest, whereas in environmental economics the focus lies on WTP for improvement of environmental quality and conservation of natural resources. If one is interested in conducting a choice experiment involving price and time attributes, an option would be to first consult for example Abrantes and Wardman (2011), where the authors provide a comprehensive meta-analysis that covers 226 British studies in which 1749 valuations of different types of time (e.g., in-vehicle time, walk time, wait time, departure time shift, etc.) are included. For a similar meta-analysis covering 389 European studies one could consult Wardman, Chintakayala, and de Jong (2016).

One must be careful when copying coefficient estimates from other studies, since those are dependent on the context, and sensitive to design and model specifications. Nevertheless the goal is not to extract a point estimate, but rather to define a reasonable interval which most likely contains the true coefficient. As Wardman *et al.* (2016) write: "These implied monetary values serve as very useful benchmarks against which new evidence can be assessed and the meta-model provides parameters and values for countries and contexts where there is no other such evidence".

Invest in a pilot study

Given the cost of collecting data, efficient designs became more popular. The more efficient a design is, the less participants one needs to question to obtain a similar level of estimation accuracy. An efficient way of spending resources is therefore to first do a pilot study on a sample of the total respondents. This way meaningful parameter estimates can be obtained, which can serve as prior information for constructing the efficient design that will be given to the remaining respondents. For the pilot study one can use an orthogonal design or an efficient design assuming zero parameters.

Expert judgement and focus groups

Another method is to consult focus groups. By debating and questioning a representative sample, one can select relevant attributes, understand their importance and incorporate meaningful attribute levels in the survey. The same information can be used to define reasonable expectations about the influence each attribute will have on the utility. For an example in which attribute prioritization exercises, attribute consolidation exercises and semi-structured interviews are described see Pinto *et al.* (2017). Besides focus groups, one can also consult an expert related to the topic.

2.2. Quantifying prior information

To incorporate prior knowledge in the design construction, one must quantify their beliefs about the coefficients of interest. This can either be done by providing a point guess, or by specifying a prior distribution. The former will lead to an efficient design based on the D-error, the latter will lead to a Bayesian D-efficient design (or DB-efficient). We recommend to always specify a prior distribution, since such design solutions are more robust.

In theory, any distribution can serve as a prior distribution as long as the probability mass is distributed over the parameter space proportionally to one's degree of belief. In practice, it is advised to use a distribution for which there exists a convenient way to take draws from. Several types of distributions are commonly used to express prior belief such as the (truncated) normal, the uniform and the lognormal distribution. Let us assume one believes that the true coefficient of price will be somewhere between -2 and 0, with all values in between being equally likely, then specifying the uniform distribution $\beta_{\text{price}} \sim \mathcal{U}(-2, 0)$ would be in accordance with that belief. Similarly, specifying a normal distribution with mean -1.5 and standard deviation 1, i.e., $\beta_{\text{price}} \sim \mathcal{N}(-1.5, 1)$, would mean that the researcher believes the true coefficient will most likely be -1.5, but also gives credibility to values near -1.5. In case one has absolutely no idea about the value of a coefficient in advance, a large standard deviation and mean equal to zero would reflect that uncertainty.

To come up with specific numbers for the mean and variance, we recommend to start with defining, for each coefficient, a lower and upper boundary in between which the true coefficient should be located. These boundaries can be set by making use of any of the previously mentioned methods. Often one of both boundaries is known a priori, since for most attributes one can tell whether they will have a positive or negative effect on the utility. When only boundaries are known, specifying a uniform distribution that spans that region is possible. When only one boundary is known, a truncated normal can be appropriate. When one is able to make a more informed guess, usually a normal distribution with mean equal to the best guess is specified. By specifying the variance, one modifies the range of credible values around the best guess. A way to decide upon the variance could be to ensure that the pre-established boundaries are for example near $\mu \pm 2\sigma$, with μ the mean and σ the standard deviation of the normal distribution. The wider the variance, the more robust the design solution will be. On the other hand, the smaller the variance, the more informative the prior is, and thus the more efficient the design solution will be if the true parameters are close to the best guess. In the next section we will show how one can evaluate this trade-off.

A Bayesian D-efficient design will be optimized by minimizing the mean D-error, in which each D-error is computed with a draw from the prior. To ensure that the solution is based on a representative sample of the prior, it is important to use enough draws. It is hard to define

a minimum number of draws, since it depends on the dimension of the prior and the quality of the draws (for a comparison see Bliemer, Rose, and Hess 2008). Since the computation time of generating a DB efficient design depends on the number of draws, we advise to generate designs with a sample such that the computation remains feasible. Afterwards one can test the robustness of the design solution by recalculating the DB-error with a larger sample from the prior (see the `DBerr` function in Section 3.3).

2.3. Test prior assumptions and robustness

In order to check whether the specified prior is reasonable, one can simply evaluate the design solution. In the output of each design generating algorithm the probabilities of choosing each alternative in each choice set are included (see for example the `CEA` algorithm in Section 3.3). With the `Decode` function one can view the labeled alternatives as they would appear in the survey (see Section 3.3). If the prior specification reflects the researcher's belief, the probabilities of choosing an alternative given a choice set should make sense. If for example the alternative "travel time = 4 hour, price = \$10" and the dominant alternative "travel time = 2 hour, price = \$2" have equal probability of being chosen, the prior or the coding of the attribute levels is clearly wrong.

In what follows we will cover an example similar to the one used for the robustness analysis in Walker *et al.* (2018). We will consider the implications of the prior on the robustness of a design. In this choice experiment we are interested in the value of time, i.e., the amount of money a consumer would be willing to pay in order to reduce waiting time with one unit. There are two continuous attributes, price and time, each containing five attribute levels. For time we use the following levels: $[30, 36, 42, 48, 54]$ minutes, and for price we use the levels: [\$1, \$4, \$7, \$10, \$13]. Since VOT is defined as $= \beta_{\text{time}}/\beta_{\text{price}}$ in a choice model with time and price as the explanatory variables, if $\beta_{\text{time}} = -0.33$, and $\beta_{\text{price}} = -1$, one is willing to pay \$20 to reduce time by one hour, or VOT = \$20/hour.

We use the **idefix** package to select three designs, each containing 20 choice sets with two alternatives. Each design is based on a different prior belief. For each belief we fix $\beta_{\text{price}} = -1$. In the first condition, the researcher has no information about the true parameters. We refer to this condition as the naive prior in which we specify $\beta_{\text{time}} \sim \mathcal{N}(0, 1)$. This corresponds to a prior belief of VOT $\sim \mathcal{N}$ (\$0/hour, \$60/hour). In the second condition, we assume the researcher knows the sign of the VOT. We will call this the semi-informative condition in which the truncated normal distribution $\beta_{\text{time}} \sim \mathcal{TN}(0, 1)$ is employed. The support of this prior distribution is restricted such that only positive VOTs have credibility. In the last case, we assume the researcher performed a pilot study in which the estimated VOT equaled \$20/hour. Therefore, the researcher is highly confident that the true VOT should be close to that value. We will call this the informative condition in which $\beta_{\text{time}} \sim \mathcal{N}(-0.33, 0.1)$. This corresponds to a prior belief of VOT $\sim \mathcal{N}$ (\$20/hour, \$6/hour). We use the `CEA` algorithm to optimize 12 initial random designs (is the default in `CEA`) for each condition, of which we then select the design with the lowest DB-error. Afterwards we use the `DBerr` function in order to evaluate the robustness of each design given a range of possible true VOTs. The complete R code can be seen in Section 3.3 under the `DBerr` function.

We can compare the efficiency of the different designs for a predefined region of interest. Assume the researcher wants to evaluate the designs given that they believe the true VOT should lie in between \$10 and \$30/hour (indicated by the vertical lines in Figure 1). From

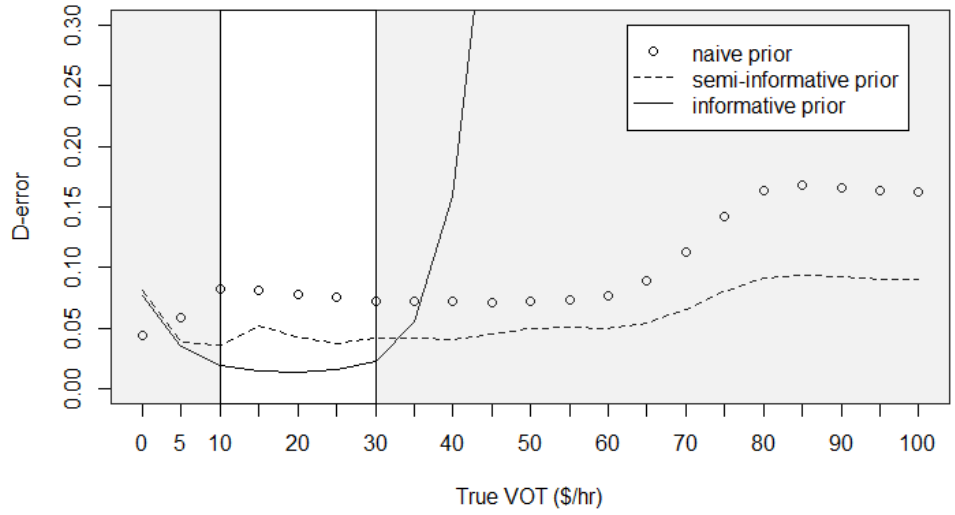

Figure 1: Comparison of the D-error for designs generated with different priors given a range of true value of time (VOT). The VOT range \$10–\$30/hour indicates the preset interval that is believed to enclose the true value, prior to the experiment.

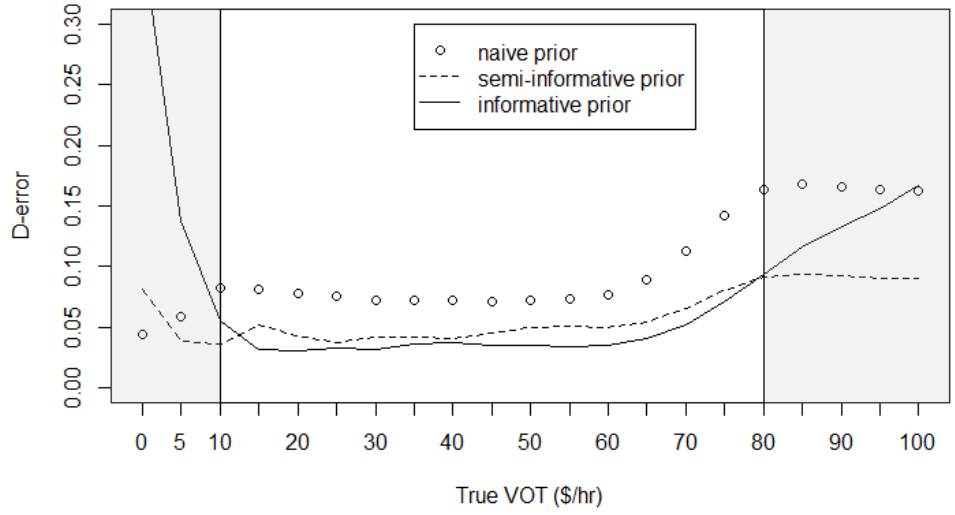

Figure 2: Comparison of the D-error for designs generated with different priors given a range of true value of time (VOT). The VOT range \$10–\$80/hour indicates the preset interval in which we want to evaluate different designs.

Figure 1 we can tell that the more informative the prior is, the more efficient the design will be given that the prior was correct (true VOT = \$20/hour). We can also see that this holds for the whole predefined region (\$10–\$30/hour). Above \$35/hour the semi-informative prior performs best, because that prior gave more credibility to this region than the informative did. Similarly, if the true VOT is below \$4/hour, the design with the naive prior performs best.

Let us now assume that we are less confident that the true VOT will be situated in between

\$10 and \$30/hour, and we would like to evaluate the designs in the VOT region of \$10–\$80/hour. We could then for example adjust the informative prior to: $\beta_{\text{time}} \sim \mathcal{U}(-1.3, -0.16)$. This corresponds to believing a priori that the true VOT will be in between \$9.8/hour and \$78/hour, With all values in that region being equally likely.

As can be seen in Figure 2, the updated informative prior results in the most efficient design for true VOTs between \$13 and \$80/hour. The difference with the semi-informative prior becomes smaller since the degree of belief is more spread out over the parameter space. The previous examples show that the more informative the prior is, the stronger the potential gain in efficiency is, but also the more inefficient the design will be if the prior information is incorrect. One can use the `DBerr` function to evaluate different designs to find a suitable prior such that prior knowledge can be incorporated without losing too much robustness. When in doubt between different design approaches, we encourage users to add other designs to the comparison in order to choose a design that best fits their needs.

2.4. Summary

There are multiple ways of gathering prior information. It should be feasible to at least decide upon reasonable boundaries in which the coefficients of interest should be located. One can evaluate a prior specification by inspecting the choice probabilities it produces. When little information is available, knowing the sign of most parameters can already improve the design substantially. Given predefined boundaries, the robustness of a design can be evaluated for that region by using the `DBerr` function.

3. D(B)-optimal designs for the MNL model

3.1. The multinomial logit model

The most applied choice model of all times is the multinomial logit model (MNL)[1], originally developed by McFadden (1974). The utility that decision maker n $(1, \ldots, N)$ receives from alternative k $(1, \ldots, K)$ in choice set s $(1, \ldots, S)$ is assumed to be composed of a systematic part and an error term

$$U_{ksn} = \boldsymbol{x}_{ksn}^{\top}\boldsymbol{\beta} + \epsilon_{ksn}.$$

The p-dimensional preference vector $\boldsymbol{\beta}$ denotes the importance of all attribute levels, represented by the p-dimensional vector $\boldsymbol{x}_{ksn}$. Unobserved influences on the utility function are captured by the independent error terms ϵ_{ksn}, which are assumed to be distributed according to a Gumbel distribution. The probability that individual n chooses alternative k in choice set s can then be expressed in closed form:

$$p_{ksn}(\boldsymbol{\beta}) = \frac{\exp(\boldsymbol{x}_{ksn}^{\top}\boldsymbol{\beta})}{\sum_{i=1}^{K}\exp(\boldsymbol{x}_{isn}^{\top}\boldsymbol{\beta})}. \tag{1}$$

In general, standard maximum likelihood techniques are applied to estimate the preference vector $\boldsymbol{\beta}$.

[1] Originally this model was known as the conditional logit model, the term MNL is, however, used more frequently nowadays.

3.2. Optimal designs for the MNL model

The more statistically efficient a design is, the smaller the confidence ellipsoids around the parameter estimates of a model will be, given a certain sample size. The information a design contains, given a choice model, can be derived from the likelihood L of that model by calculating the expected Fisher information matrix:

$$\mathcal{I}_{\text{FIM}} = -\mathsf{E}\left(\frac{\partial^2 L}{\partial\boldsymbol{\beta}\partial\boldsymbol{\beta}^\top}\right).$$

For the MNL model, this yields the following expression, where N is the number of respondents, $\boldsymbol{X_s} = (\boldsymbol{x}_{1sn}^\top, \ldots, \boldsymbol{x}_{Ksn}^\top)$ the design matrix of choice set s, $\boldsymbol{P_s} = \text{diag}[p_{1sn}, p_{2sn}, \ldots, p_{Ksn}]$, and $\boldsymbol{p_s} = [p_{1sn}, p_{2sn}, \ldots, p_{Ksn}]^\top$:

$$\mathcal{I}_{\text{FIM}}\left(\boldsymbol{\beta}|\mathbf{X}\right) = N\sum_{s=1}^{S}\boldsymbol{X}_s^\top\left(\boldsymbol{P}_s - \boldsymbol{p}_s\boldsymbol{p}_s^\top\right)\boldsymbol{X}_s. \tag{2}$$

Several efficiency measures have been proposed based on the information matrix, among which the D-efficiency has become the standard approach (Kessels *et al.* 2006; Rose and Bliemer 2014). A D-optimal design approach maximizes the determinant of the information matrix, therefore minimizing the generalized variance of the parameter estimates. The criterion is scaled to the power $1/p$, with p the number of parameters the model contains:

$$\boldsymbol{\Omega} = \mathcal{I}_{\text{FIM}}\left(\boldsymbol{\beta}|\mathbf{X}\right)^{-1},$$
$$\text{D-error} = \det\left(\boldsymbol{\Omega}\right)^{1/p}.$$

In order to calculate the D-error of a design, one must assume a model and parameter values. Since there is uncertainty about the parameter values, a prior distribution $\pi(\boldsymbol{\beta})$ can be defined on the preference parameters. In this case the expected D-error is minimized over the prior preference distribution and is referred to as the DB-error

$$\text{DB-error} = \int \det\left(\Omega\right)^{1/p}\pi(\boldsymbol{\beta})d\boldsymbol{\beta}. \tag{3}$$

To find the design that minimizes such criteria, different algorithms have been proposed (see Cook and Nachtsheim 1980, for an overview). We choose to implement both the modified Fedorov algorithm, which was adapted from the classical Fedorov exchange algorithm (Fedorov 1972), as well as a coordinate exchange algorithm. The former swaps profiles from an initial design matrix with candidate profiles in order to minimize the D(B)-error. The latter changes individual attribute levels in order to optimize the design. For more details see `Modfed` and `CEA` functions in Section 3.3.

3.3. Optimal designs for the MNL model with package idefix

Two main algorithms are implemented to generate optimal designs for the MNL model through the `Modfed` function and the `CEA` function. `Modfed` is an implementation of a modified Fedorov algorithm, whereas `CEA` is a coordinate exchange algorithm. The latter is substantially faster and better suited for large designs problems. Due to a less exhaustive search, it can produce slightly less efficient designs when there are few choice sets compared to the `Modfed` algorithm.

	Dummy coding			Effects coding		
Levels	β_1	β_2	β_3	β_1	β_2	β_3
level 1	0	0	0	1	0	0
level 2	1	0	0	0	1	0
level 3	0	1	0	0	0	1
level 4	0	0	1	−1	−1	−1

Table 1: Dummy and effects coding in the **idefix** package for an attribute with four levels.

The `Modfed` is much slower, but has the advantage that the user can specify a candidate list of alternatives, and thus is able to put restrictions on the design. We will start by explaining how to generate a candidate list of alternatives by using the `Profiles` function, and continue with the `Modfed` function. Afterwards we show how to decode a design with the `Decode` function. Lastly we explain the `CEA` algorithm, and we end with an example of the `DBerr` function that can be used to evaluate the robustness of different design solutions.

`Profiles`

The first step in creating a discrete choice design is to decide which attributes, and how many levels of each, will be included. This is often not a straightforward choice that highly depends on the research question. In general, excluding relevant attributes will result in increased error variance, whereas including too many can have a similar result due to the increase in task complexity. For more elaborated guidelines in choosing attributes and levels we refer to Bridges *et al.* (2011).

Afterwards one can start creating profiles as combinations of attribute levels. Most often, all possible combinations are valid profiles and then the `Profiles` function can be used to generate them. It could be that some combinations of attribute levels are not allowed for a variety of reasons. In that case the list of possible profiles can be restricted afterwards by deleting the profiles that do not suffice.

The function `Profiles` has three arguments of which one is optional. In the `lvls` argument one can specify how many attributes should be included, and how many levels each attribute should have. The number of elements in vector `at.lvls` indicates the number of attributes. The numeric values of that vector indicate the number of levels each attribute contains. In the example below there are three attributes, the first one has three, the second four, and the last one has two levels. The type of coding should be specified for each attribute, here with the vector `c.type`, with one character for each attribute. Attributes can be effects coded `"E"`, dummy coded `"D"` or treated as a continuous variable `"C"`. In this case all attributes will be effects coded. In Table 1, the different coding schemes in the **idefix** package are depicted for an attribute containing four levels.

```
R> library("idefix")
R> at.lvls <- c(3, 4, 2)
R> c.type <- c("E", "E", "E")
R> Profiles(lvls = at.lvls, coding = c.type)
```

```
   Var11 Var12 Var21 Var22 Var23 Var31
1      1     0     1     0     0     1
2      0     1     1     0     0     1
3     -1    -1     1     0     0     1
4      1     0     0     1     0     1
5      0     1     0     1     0     1
6     -1    -1     0     1     0     1
...
24    -1    -1    -1    -1    -1    -1
```

The output is a matrix in which each row is a possible profile.

When continuous attributes are desired, the levels of those attributes should be specified in `c.lvls`, with one numeric vector for each continuous attribute, and the number of elements should equal the number of levels specified in `lvls` for each attribute. In the example below there is one dummy coded attributed and two continuous attributes where the first one contains four levels (i.e., 4, 6, 8, and 10) and the second one two levels (i.e., 7 and 9).

```
R> at.lvls <- c(3, 4, 2)
R> c.type <- c("D", "C", "C")
R> con.lvls <- list(c(4, 6, 8, 10), c(7, 9))
R> Profiles(lvls = at.lvls, coding = c.type, c.lvls = con.lvls)

   Var12 Var13 Var2 Var3
1      0     0    4    7
2      1     0    4    7
3      0     1    4    7
4      0     0    6    7
5      1     0    6    7
6      0     1    6    7
...
24     0     1   10    9
```

The output is a matrix in which each row is a possible profile. The last two columns represent the continuous attributes.

`Modfed`

A modified Fedorov algorithm is implemented and can be used with the `Modfed` function. The function consists of eleven arguments of which seven are optional. The first argument `cand.set` is a matrix containing all possible profiles that could be included in the design. This can be generated with the `Profiles` function as described above, but this is not necessary. Furthermore, the desired number of choice sets `n.sets`, the number of alternatives `n.alts` in each choice set, and the draws from the prior distribution, for which the design should be optimized, should be specified in `par.draws`. By entering a numeric vector the D-error will be minimized given the parameter values and the MNL likelihood. By specifying a matrix in `par.draws`, in which each row is a draw from a multivariate prior distribution, the DB-error will be optimized. We recommend to always use the DB-error since a D-efficient design is more sensitive to a misspecification of the prior.

If alternative specific constants are desired, the argument `alt.cte` should be specified. In order to do this, a binary vector should be given with length equal to `n.alts`, indicating for each alternative whether an alternative specific constant should be present `1` or not `0`. Whenever alternative specific constants are present, the `par.draws` argument requires a list containing two matrices as input. The first matrix contains the parameter draw(s) for the alternative specific constant parameter(s). The second matrix contains the draw(s) for the remaining parameter(s). To verify, the total number of columns in `par.draws` should equal the sum of the number of nonzero parameters in `alt.cte` and the number of parameters in `cand.set`.

For some discrete choice experiments, a no choice alternative is desired. This is usually an alternative containing one alternative specific constant and zero values for all other attribute levels. If such an alternative should be present, the `no.choice` argument can be set to `TRUE`. When this is the case, the design will be optimized given that the last alternative of each choice set is a no choice alternative. Note that when `no.choice = TRUE`, `alt.cte[n.alts]` should be `1`, since the no choice alternative has an alternative specific constant.

The algorithm will swap profiles from `cand.set` with profiles from an initial design in order to maximize the D(B)-efficiency. In order to avoid local optima the algorithm will repeat this procedure for several random starting designs. The default of `n.start = 12` can be changed to any integer. By default `best = TRUE` so that the output will only show the design with the lowest D(B)-error. If `best = FALSE`, all optimized starting designs are shown in the output. One can also provide their own starting design(s) in `start.des`, in this case `n.start` is ignored.

The modified Fedorov algorithm is fairly rapid; however, for complex design problems (with lots of attributes and attribute levels), the computation time can be high. Therefore, we make use of parallel computing through the **parallel** package in R. By default `parallel = TRUE` and `detecCores` will detect the number of available CPU cores. The optimization of the different starting designs will be distributed over the available cores minus one.

The algorithm will converge when an iteration occurs in which no profile could be swapped in order to decrease the D(B)-error anymore. A maximum number of iterations can be specified in `max.iter`, but is by default infinite.

In the example below a DB-optimal design is generated for a scenario with three attributes. The attributes have respectively four, two and three levels each. All of them are dummy coded. The matrix `M`, containing draws from the multivariate prior distribution with mean `mean` and covariance matrix `sigma`, is specified in `par.draws`. The mean vector contains six elements. The first three are the parameters for the first attribute, the fourth is the parameter for the second attribute and the last two are the ones for the third attribute.

```
R> code <- c("D", "D", "D")
R> cs <- Profiles(lvls = c(4, 2, 3), coding = code)
R> mu <- c(-0.4, -1, -2, -1, 0.2, 1)
R> sigma <- diag(length(mu))
R> set.seed(123)
R> M <- MASS::mvrnorm(n = 500, mu = mu, Sigma = sigma)
R> D <- Modfed(cand.set = cs, n.sets = 8, n.alts = 2,
+    alt.cte = c(0, 0), par.draws = M)
R> D
```

```
$design
          Var12 Var13 Var14 Var22 Var32 Var33
set1.alt1     1     0     0     1     1     0
set1.alt2     0     1     0     0     0     1
set2.alt1     1     0     0     1     0     1
set2.alt2     0     0     0     0     0     0
set3.alt1     1     0     0     0     0     1
set3.alt2     0     0     0     1     1     0
set4.alt1     0     0     0     1     0     0
set4.alt2     0     1     0     0     1     0
set5.alt1     0     0     0     1     1     0
set5.alt2     0     0     1     0     0     0
set6.alt1     1     0     0     0     0     0
set6.alt2     0     0     1     1     0     1
set7.alt1     0     1     0     1     0     0
set7.alt2     0     0     0     1     0     1
set8.alt1     0     1     0     1     0     0
set8.alt2     0     0     1     0     1     0

$error
[1] 2.302946

$inf.error
[1] 0

$probs
      Pr(alt1)  Pr(alt2)
set1 0.3516382 0.6483618
set2 0.4503424 0.5496576
set3 0.7077305 0.2922695
set4 0.4842112 0.5157888
set5 0.6790299 0.3209701
set6 0.7231303 0.2768697
set7 0.1682809 0.8317191
set8 0.4540432 0.5459568
```

The output consists of the optimized design that resulted in the lowest DB-error since `best` is `TRUE` by default. Besides the final D(B)-error `$error`, `$inf.error` denotes the percentage of draws for which the design resulted in an infinite D-error. This could happen for extremely large parameter values, which result in probabilities of one or zero for all alternatives in all choice sets. In that case the elements of the information matrix will be zero, and the D-error will be infinite. This percentage should thus be preferably close to zero when calculating the DB-error and zero when generating a D-optimal design. Lastly, `$probs` shows the average probabilities for each alternative in each choice set given the sample from the prior preference distribution `par.draws`.

Decode

The `Decode` function allows the user to decode a coded design matrix into a readable choice design with labeled attribute levels, as they would appear in a real survey. Comparing the decoded alternatives with their predicted choice probabilities is a simple way of checking whether the prior specification is reasonable. It can for example happen that an efficient design contains some dominant alternatives. To what extent these are undesired depends on the researcher. Some argue that dominant alternatives can serve as a test to see whether the respondent takes the choice task seriously, others argue that it may lead to lower response efficiency. In any case, it is important that the predicted choice probabilities, which are a direct consequence of the prior, seem plausible. Otherwise either the prior or the used coding is presumably wrongly specified (Crabbe and Vandebroek 2012; Bliemer, Rose, and Chorus 2017).

The design that needs to be decoded has to be specified in `des`. This is a matrix in which each row is an alternative. Furthermore, the number of alternatives `n.alts` and the applied coding `coding` should be indicated. The labels of the attribute levels which will be used in the DCE should be specified in `lvl.names`. This should be a list containing a vector for each attribute. Each vector has equal elements as the corresponding attribute has attribute levels. In the example below we will decode the design previously generated with the `Modfed` function into a readable design. We specify all attribute levels in `lvls`. For this example we mimicked a transportation DCE. The first attribute is the "cost" with four different values: \$15, \$20, \$30 and \$50. The second attribute represents "travel time", an alternative that can have 2 or 30 minutes as attribute levels. Lastly the attribute "comfort" has three levels, the comfort can be bad, moderate or good. In `des` we specify the optimized design that resulted in the lowest DB error from the output of the `Modfed` function. In `coding` the same type of coding we used to generate the design matrix should be specified. In this case all attributes were dummy coded (as in the example above).

```
R> lvls <- list(c("$15", "$20", "$30", "$50"), c("2 min", "15 min"),
+    c("bad", "moderate", "good"))
R> DD <- Decode(des = D$design, lvl.names = lvls, coding = code)
R> DD

$design
           V1     V2       V3
set1.alt1 $20 15 min moderate
set1.alt2 $30  2 min     good
set2.alt1 $20 15 min     good
set2.alt2 $15  2 min      bad
set3.alt1 $20  2 min     good
set3.alt2 $15 15 min moderate
set4.alt1 $15 15 min      bad
set4.alt2 $30  2 min moderate
set5.alt1 $15 15 min moderate
set5.alt2 $50  2 min      bad
set6.alt1 $20  2 min      bad
set6.alt2 $50 15 min     good
```

```
set7.alt1 $30 15 min      bad
set7.alt2 $15 15 min     good
set8.alt1 $30 15 min      bad
set8.alt2 $50  2 min moderate

$lvl.balance
$lvl.balance$`attribute 1 `

$15 $20 $30 $50
  5   4   4   3

$lvl.balance$`attribute 2 `

15 min  2 min
     9      7

$lvl.balance$`attribute 3 `

     bad     good moderate
       6        5        5
```

The output of `Decode` contains two components. The first one, `$design`, shows the decoded design matrix. The second one, `lvl.balance`, shows the frequency of each attribute level in the design. As previously mentioned, besides statistical efficiency other criteria such as attribute level balance can be of importance too.

In the second example we optimize several starting designs for the same attributes as the ones in the first example. As before, all possible profiles are generated using the `Profiles` function. This time we include an alternative specific constant (asc) for the first alternative and we add a no choice alternative. A no choice alternative is coded as an alternative that contains one asc and zeros for all other attribute levels. The vector `c(1, 0, 1)` specified in `alt.cte` indicates that there are three alternatives of which the first and the third (the no choice alternative) have an asc. The mean vector `m` contains eight entries, the first one corresponds to the asc of the first alternative, the second one to the asc of the no choice alternative. The third, fourth and fifth elements correspond to the prior mean of the coefficients of the levels of the first attribute. The sixth element indicates the prior mean of the coefficient of the levels for the second attribute and the last two elements to the levels of the third attribute. In this example all attributes are effects coded (see Table 1 for an example). A sample is drawn from the multivariate normal prior distribution with mean `m` and covariance matrix `v` which is passed to `par.draws`. In order to avoid confusion the draws for the alternative specific constants are separated from the draws for the coefficients and passed on to `par.draws` in a list.

```
R> set.seed(123)
R> code <- c("E", "E", "E")
R> cs <- Profiles(lvls = c(4, 2, 3), coding = code )
R> alt.cte <- c(1, 0, 1)
R> m <- c(0.1, 1.5, 1.2, 0.8, -0.5, 1, -1.5, 0.6)
R> v <- diag(length(m))
```

```
R> ps <- MASS::mvrnorm(n = 500, mu = m, Sigma = v)
R> ps <- list(ps[, 1:2], ps[, 3:8])
R> D.nc <- Modfed(cand.set = cs, n.sets = 10, n.alts = 3,
+    alt.cte = alt.cte, par.draws = ps, no.choice = TRUE, best = FALSE)
R> for (i in 1:length(D.nc)) print(D.nc[[i]]$error)

[1] 1.230047
[1] 1.277368
[1] 1.241679
[1] 1.241667
[1] 1.250804
[1] 1.276689
[1] 1.252373
[1] 1.266141
[1] 1.248671
[1] 1.249621
[1] 1.274209
[1] 1.253853
```

Because `best` was set to `FALSE`, the outcome (i.e., the optimized design matrix, the DB-error and the probabilities) of each initial design was stored. We print out all the DB-errors and decide which design we would like to decode. Another option is to optimize more starting designs. In the example below the first design is decoded, since this was the one that resulted in the lowest DB-error. Note that we now have to specify which alternative is the no choice alternative in the `no.choice` argument.

```
R> test <- Decode(des = D.nc[[1]]$design, n.alts = 3,
+    lvl.names = lvls, alt.cte = alt.cte, coding = code, no.choice = 3)
R> cbind(test$design, probs = as.vector(t(D.nc[[1]]$probs)))

              V1     V2       V3     probs
set1.alt1    $50  2 min     good 0.3214916
set1.alt2    $15 15 min moderate 0.3133349
no.choice   <NA>   <NA>     <NA> 0.3651735
set2.alt1    $15  2 min moderate 0.4697414
set2.alt2    $20  2 min     good 0.3856513
no.choice.1 <NA>   <NA>     <NA> 0.1446072
set3.alt1    $20 15 min moderate 0.2784902
set3.alt2    $15  2 min      bad 0.3145071
no.choice.2 <NA>   <NA>     <NA> 0.4070027
set4.alt1    $30  2 min     good 0.4284075
set4.alt2    $50  2 min moderate 0.2310475
no.choice.3 <NA>   <NA>     <NA> 0.3405450
set5.alt1    $20  2 min     good 0.3565410
set5.alt2    $15  2 min     good 0.4518309
no.choice.4 <NA>   <NA>     <NA> 0.1916281
set6.alt1    $20  2 min      bad 0.3174667
```

```
set6.alt2    $50 15 min     good 0.1360625
no.choice.5 <NA>   <NA>     <NA> 0.5464708
set7.alt1    $15 15 min     good 0.4273929
set7.alt2    $30  2 min      bad 0.1274950
no.choice.6 <NA>   <NA>     <NA> 0.4451121
set8.alt1    $50  2 min moderate 0.3294313
set8.alt2    $30 15 min     good 0.1941744
no.choice.7 <NA>   <NA>     <NA> 0.4763942
set9.alt1    $15  2 min      bad 0.2896765
set9.alt2    $30  2 min moderate 0.3206922
no.choice.8 <NA>   <NA>     <NA> 0.3896313
set10.alt1   $30  2 min moderate 0.2405066
set10.alt2   $20  2 min moderate 0.4750211
no.choice.9 <NA>   <NA>     <NA> 0.2844724
```

CEA

It is known that the computation time of the `Modfed` algorithm increases exponentially by including additional attributes. Therefore, we also implemented the coordinate exchange algorithm (CEA), which is particularly effective for larger design problems (Tian and Yang 2017; Meyer and Nachtsheim 1995). Since the CEA approach runs in polynomial time, it is expected that computation time will be reduced by one or two orders of magnitude, while producing equally efficient designs as the `Modfed` algorithm. Only when using few initial starting designs, for small design problems (e.g., less than 10 choice sets), it could be that the `CEA` function produces designs that are slightly less efficient as the ones obtained from `Modfed`. This is because the `CEA` algorithm performs a less exhaustive search. To overcome this problem, one can increase the number of random initial designs `n.start`, but we advise to use the modified Fedorov algorithm for such problems. Similarly to the `Modfed` function, the `CEA` procedure improves random initial design matrices in which a row represents a profile and columns represent attribute levels. The latter, however, considers changes on an attribute-by-attribute basis, instead of swapping each profile with each possible profile. It is thus no longer needed to generate a candidate set of all possible profiles. The downside is that one can also no longer eliminate particular profiles in advance. All arguments, and the output that `CEA` produces, are exactly the same as the ones documented in the `Modfed` function. The only difference is that the `lvls` and `coding` argument now directly serve as input for the `CEA` function.

```
R> set.seed(123)
R> lvls <- c(4, 2, 3)
R> coding <- c("E", "E", "E")
R> alt.cte <- c(1, 0, 1)
R> m <- c(0.1, 1.5, 1.2, 0.8, -0.5, 1, -1.5, 0.6)
R> v <- diag(length(m))
R> ps <- MASS::mvrnorm(n = 500, mu = m, Sigma = v)
R> ps <- list(ps[, 1:2], ps[, 3:8])
R> D.nc_cea <- CEA(lvls = lvls, coding = coding, n.alts = 3, n.sets = 10,
+    alt.cte = alt.cte, par.draws = ps, no.choice = TRUE, best = TRUE)
R> D.nc_cea
```

```
$design
           alt1.cte alt3.cte Var11 Var12 Var13 Var21 Var31 Var32
set1.alt1         1        0     0     0     1    -1    -1    -1
set1.alt2         0        0     1     0     0    -1     0     1
no.choice         0        1     0     0     0     0     0     0
set2.alt1         1        0     1     0     0     1     0     1
set2.alt2         0        0     0     1     0     1    -1    -1
no.choice         0        1     0     0     0     0     0     0
set3.alt1         1        0    -1    -1    -1     1    -1    -1
set3.alt2         0        0     0     0     1    -1     0     1
no.choice         0        1     0     0     0     0     0     0
set4.alt1         1        0     1     0     0     1    -1    -1
set4.alt2         0        0     0     0     1     1    -1    -1
no.choice         0        1     0     0     0     0     0     0
set5.alt1         1        0     1     0     0     1     1     0
set5.alt2         0        0     0     1     0     1     0     1
no.choice         0        1     0     0     0     0     0     0
set6.alt1         1        0    -1    -1    -1     1     1     0
set6.alt2         0        0     1     0     0    -1    -1    -1
no.choice         0        1     0     0     0     0     0     0
set7.alt1         1        0     0     0     1     1     0     1
set7.alt2         0        0     1     0     0     1     1     0
no.choice         0        1     0     0     0     0     0     0
set8.alt1         1        0     0     1     0    -1    -1    -1
set8.alt2         0        0     0     0     1     1    -1    -1
no.choice         0        1     0     0     0     0     0     0
set9.alt1         1        0     0     1     0    -1     0     1
set9.alt2         0        0    -1    -1    -1    -1    -1    -1
no.choice         0        1     0     0     0     0     0     0
set10.alt1        1        0     0     1     0     1     1     0
set10.alt2        0        0    -1    -1    -1     1     0     1
no.choice         0        1     0     0     0     0     0     0

$error
[1] 1.237644

$inf.error
[1] 0

$probs
       Pr(alt1)  Pr(alt2) Pr(no choice)
set1  0.1870938 0.3265117     0.4863944
set2  0.4697414 0.3856513     0.1446072
set3  0.3605579 0.1517753     0.4876668
set4  0.6058905 0.1608716     0.2332379
set5  0.2186997 0.4996534     0.2816469
set6  0.1087114 0.4204388     0.4708497
```

```
set7  0.3627346 0.2461984     0.3910670
set8  0.2609301 0.3730199     0.3660500
set9  0.3204917 0.1179716     0.5615366
set10 0.2599124 0.2614408     0.4786468
```

DBerr

As previously mentioned, the choice of using an efficient design approach is related to the confidence one has in the prior specification of the parameters that are to be estimated (Walker *et al.* 2018). Other design approaches often rely on different assumptions regarding the true data generating process. Foreseeing the design implications given that some assumptions are violated, or the prior deviates from the true parameters, can be difficult. The `DBerr` function allows to evaluate designs resulting from different assumptions to be compared given a range of plausible true parameters. The function simply calculates the D(B)-error given a design `des` and parameter values `par.draws`. The latter can be a vector, in which case the D-error is calculated, or it can be matrix in which each row is a draw. If a matrix is supplied and `mean = TRUE` the D(B)-error is calculated. If `mean = FALSE` a vector with D-errors is returned. Lastly, the number of alternatives in each choice set needs to be specified in `n.alts`.

In what follows we show the code used to generate the robustness plots in Section 2.3. We compared different design solutions, each generated with different prior beliefs.

```
R> set.seed(123)
R> N <- 250
R> U <- rnorm(n = N, mean = 0, sd = 1)
R> S <- rtruncnorm(n = N, a = -Inf, b = 0, mean = 0, sd = 1)
R> I <- rnorm(n = N, mean = -0.33, sd = 0.1)
R> I2 <- runif(n = N, min= -1.3, max = -0.16)
```

We draw a sample from each of the prior distributions. The sample of the naive prior is stored in `U`, `S` contains the semi-informative sample, and `I` and `I2` are the informative samples for β_{time}. We will continue by showing how we generated the informative design and check the robustness for that design. The same can be done for the other priors by replacing `I` with the desired sample.

```
R> I <- cbind(I, -1)
R> lev_time <- c(30,36,42,48,54)
R> lev_price <- c(1,4,7,10,13)
R> D_I <- CEA(lvls = c(5, 5), coding = c("C", "C"),
+    c.lvls = list(lev_time, lev_price), n.sets = 20, n.alts = 2,
+    parallel = TRUE, par.draws = I, best = TRUE)
R> des <- D_I$design
```

We used the `CEA` algorithm to generate an efficient design given sample `I`. Afterwards the design matrix is stored in `des`. There where two attributes, each containing 5 levels. Since the coefficient of price was fixed to -1, we can define a range of possible true VOTs (\$0–\$100/hour) by varying β_{time} from 0 till -1.667. Finally the `DBerr` function can be used to calculate the efficiency of the design given each possible true VOT. Specifying `mean = TRUE`,

would return the mean which is the same as the DB-error for the specified range. One can thus test multiple designs for different plausible parameters.

```
R> range <- cbind(seq(-1.667, 0, 0.08333), -1)
R> I_robust <- DBerr(par.draws = range, des = des, n.alts = 2,
+    mean = FALSE)
R> I_robust

 [1] 19.41543197 15.12089434 11.77624555  9.17131336  7.14225121  5.56088291
 [7]  4.32537544  3.34943050  2.54316599  1.78598093  1.01289324  0.43448119
[13]  0.15842765  0.05476671  0.02232948  0.01558887  0.01327167  0.01476203
[19]  0.01931387  0.03505369  0.07640805
```

As can be seen in Figure 1, the very informative prior performs well when the best guess ($20/hour) equals the true VOT, but becomes quickly inefficient when the true VOT deviates from this value.

4. Individually adaptive sequential Bayesian designs

So far we generated statistically efficient designs, given a prior preference distribution. Since we optimize the same design for all respondents, we hope that the prior accurately reflects the preferences of those respondents. If we, however, assume there exists preference heterogeneity in the population, we must acknowledge that the design will be more efficient for some participants and less efficient to others. The IASB approach is motivated by the desire to overcome this problem. By making use of an individual adaptive design approach, preference heterogeneity is taken into account in the design stage of the experiment. Before explaining the IASB approach in more detail we will give a brief explanation of the MIXL model, which is the most commonly used choice model that takes preference heterogeneity into account and for which the IASB approach can be used.

4.1. The mixed logit model

Essentially the mixed logit model (MIXL) is an extension of the MNL model in the sense that it allows for heterogeneity in the preference parameters (Hensher and Greene 2003; McFadden and Train 2000; Train 2003). The mixed logit probability for choosing a particular alternative in a given choice set, unconditionally on $\boldsymbol{\beta}$, is expressed as follows:

$$\pi_{ks} = \int p_{ks}(\boldsymbol{\beta}) f(\boldsymbol{\beta}) d\boldsymbol{\beta}.$$

The logit probability $p_{ks}(\boldsymbol{\beta})$ as specified in Equation 1 is weighted over the density function $f(\boldsymbol{\beta})$, describing the preference heterogeneity. The likelihood of the observed responses for N respondents can then be written as

$$L(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \prod_{n=1}^{N} \int \left(\prod_{s=1}^{S} \prod_{k=1}^{K} \left(p_{ksn}(\boldsymbol{\beta}_n)\right)^{y_{ksn}} \right) f(\boldsymbol{\beta}_n|\boldsymbol{\theta}) d\boldsymbol{\beta}_n,$$

where $\mathbf{y}$ is a binary vector with elements y_{ksn} which equals one if alternative k in choice set s was chosen by respondent n, and zero otherwise, $\mathbf{X}$ is the full design matrix containing the

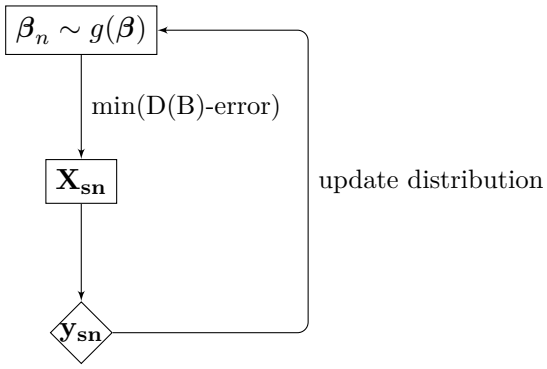

Figure 3: Flow chart of the IASB algorithm in which an individual response $\boldsymbol{y}_{sn}$ on choice set $\boldsymbol{X}_{sn}$ is used to update the prior preference distribution $g(\boldsymbol{\beta})$. The next choice set is selected based on the updated preference distribution and previously selected choice sets.

subdesigns for N respondents and $\boldsymbol{\theta}$ contains the parameters of the population distribution $f(\boldsymbol{\beta})$. MIXL models are either estimated with a hierarchical Bayesian estimation procedure or with a simulated likelihood approach.

4.2. Optimal designs for the mixed logit model

Because the Fisher information matrix for the MIXL model cannot be computed independently of the observed responses, generating DB-optimal designs requires extensive simulations and becomes quickly too time consuming (Bliemer and Rose 2010b). Another approach has been proposed by Yu *et al.* (2011), which exploits the idea that the MIXL model assumes respondents to choose according to a MNL model, but each with different preferences. The IASB design approach consists thus of generating individual designs, where each design is tailor-made for one unique participant. The approach is sequential because choice sets are selected one after the other during the survey. The approach is adaptive because the individual preference distribution is updated after each observed response.

The flow chart in Figure 3 represents the idea of the IASB approach. At the top we see the distribution of the individual preferences of a respondent $\boldsymbol{\beta_n}$. Initially a sample is drawn from the prior distribution $\boldsymbol{\beta}_n \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Based on that sample, one or more initial choice sets $\boldsymbol{X}_{sn}$ will be selected by minimizing the D(B)-error. After observing the response(s) $\boldsymbol{y}_{sn}$, the prior is updated in a Bayesian way and samples are now drawn from the posterior distribution $g(\boldsymbol{\beta})$. This process is repeated as many times as there are individual sequential choice sets, resulting in an individualized design for each respondent. It has been proven that generating designs according to this IASB approach results in choice data of high quality in order to estimate the MIXL model (Danthurebandara *et al.* 2011; Yu *et al.* 2011).

4.3. Optimal designs for the MIXL model with package idefix

In this part, the functions necessary to generate adaptive choice sets in a sequential way, as explained in Section 4.2, are described. The two main building blocks of the IASB approach consist of an importance sampling algorithm `ImpsampMNL` to sample from the posterior, and the `SeqMOD` algorithm, to select the most efficient choice set given a posterior sample. Furthermore, a function to generate responses `RespondMNL` is included, this way all elements to

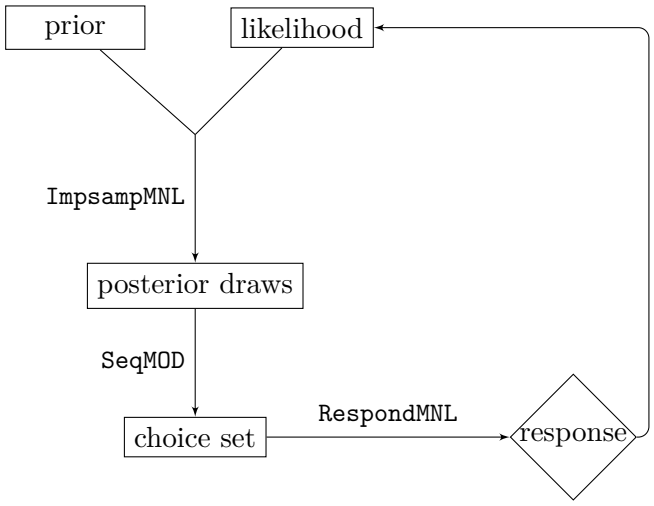

Figure 4: Simulation setup for the IASB approach. `ImpsampMNL`, `SeqMOD` and `RespondMNL` are functions included in the **idefix** package.

set up a simulation study, as depicted in Figure 4, are present. If the reader has no interest in evaluating an adaptive design approach with response simulation, but rather in collecting empirical data, one can proceed to Section 5.

In what follows an example is given of the workflow together with a description of each of the functions involved. In the example, choice data for one respondent will be generated, while part of the choice sets are selected making use of the IASB methodology. We will assume a scenario with three attributes. The first attribute has four levels, the second attribute has three levels and the last one has two levels. We choose to use dummy coding for all attributes and not to include an alternative specific constant. As a prior we use draws from a multivariate normal distribution with mean vector `m` and covariance matrix `v`. In each choice set there are two alternatives.

First we will generate a DB optimal design containing eight initial fixed choice sets in the same way as explained in Section 2.3.

```
R> set.seed(123)
R> cs <- Profiles(lvls = c(4, 3, 2), coding = c("D", "D", "D"))
R> m <- c(0.25, 0.5, 1, -0.5, -1, 0.5)
R> v <- diag(length(m))
R> ps <- MASS::mvrnorm(n = 500, mu = m, Sigma = v)
R> init.des <- Modfed(cand.set = cs, n.sets = 8, n.alts = 2,
+    alt.cte = c(0, 0), par.draws = ps)$design
R> init.des

          Var12 Var13 Var14 Var22 Var23 Var32
set1.alt1     0     0     1     0     1     1
set1.alt2     0     0     0     0     0     0
set2.alt1     0     1     0     0     0     1
set2.alt2     1     0     0     1     0     0
set3.alt1     0     0     0     0     0     1
```

```
set3.alt2     0     1     0     0     1     0
set4.alt1     0     0     0     1     0     1
set4.alt2     1     0     0     0     0     0
set5.alt1     0     0     0     0     0     0
set5.alt2     1     0     0     1     0     1
set6.alt1     0     1     0     1     0     0
set6.alt2     0     0     0     0     1     0
set7.alt1     0     0     1     1     0     0
set7.alt2     1     0     0     0     0     1
set8.alt1     0     1     0     0     1     1
set8.alt2     0     0     1     0     0     0
```

The next step is to simulate choice data for the initial design. We assume that the true preference parameter vector is the following:

```
R> truePREF <- c(0.5, 1, 2, -1, -1, 1.5)
```

RespondMNL

To simulate choices based on the logit model the `RespondMNL` function can be used. The true (individual) preference parameters can be set in `par`, in this case `truePREF`. In `des` a matrix should be specified in which each row is a profile. This can be a single choice set or a design matrix containing several choice sets, in this example our initial design `init.des` is used. The number of alternatives per choice set should be set in `n.alts`.

```
R> set.seed(123)
R> y.sim <- RespondMNL(par = truePREF, des = init.des, n.alts = 2)
R> y.sim

 [1] 1 0 1 0 1 0 0 1 0 1 1 0 0 1 0 1
```

The output is a binary vector indicating the simulated choices. Given that $K = 2$, the alternatives that were chosen in each choice set were respectively the first, first, first, second, second, first, second and second alternative.

At this point we have information about the true preferences captured in the responses `y.sim`, given the choice sets in `init.des`. We can now take this information into account by updating our prior distribution.

ImpsampMNL

Since the posterior has no closed form, an importance sampling algorithm is provided with the `ImpsampMNL` function, which can be used to take draws from the posterior after each observed response. As importance density a multivariate student t distribution, with degrees of freedom equal to the dimensionality, is used. The mean of the importance density is the mode of the posterior distribution and the covariance matrix $-\boldsymbol{H}^{-1}$, with $\boldsymbol{H}$ the Hessian matrix of the posterior distribution. The draws are taken systematically using extensible shifted lattice points (Yu *et al.* 2011).

As prior distribution the multivariate normal is assumed for which the mean vector can be specified in `prior.mean`, and covariance matrix in `prior.covar`. Here we use the same mean prior vector `m` and covariance matrix `v` as the ones we used to generate the initial design. Previously presented choice sets, in this case `init.des`, can be passed through `des`, whereas the simulated responses `y.sim` can be passed through `y`. The number of draws can be specified in `n.draws`, here we draw 200 samples from the posterior distribution. Note that the `ImpsampMNL` function includes three more arguments which are not required. If alternative specific constants are present, those should be specified in `alt.cte` argument which is by default `NULL`. The last two arguments `lower` and `upper` allow the user to sample from a truncated normal distribution. This can be desired when the researcher is certain about the sign of one or more parameters. For example when there are three parameters and the researcher knows the first parameter should be positive, a lower boundary of zero can be specified for that parameter with `lower = c(0, -Inf, -Inf)`. Equivalently when only negative draws are desired for the first parameter an upper boundary can be specified by stating `upper = c(0, Inf, Inf)`. In this case we, however, do not know the signs of the parameters and apply no boundaries.

```
R> set.seed(123)
R> draws <- ImpsampMNL(n.draws = 200, prior.mean =  m,
+    prior.covar = v, des = init.des, n.alts = 2, y = y.sim)
R> draws

$sample
              Var12        Var13        Var14         Var22       Var23
  [1,]  1.070865236  0.638213757  2.049646085 -1.334767e+00 -2.14475738
  [2,]  0.662390619  0.457390508  0.571839377 -1.842649e-01 -0.55735903
  [3,]  2.137729327  1.737021903  1.640171096 -3.223393e-01 -1.56516482
...
[200,]  0.584339280  0.275411214  1.928374787 -1.710515e+00 -1.36622933
            Var32
  [1,]  0.09857508
  [2,]  1.87854394
  [3,]  0.83483329
...
[200,]  1.51836041

$weights
  [1] 0.0064215784 0.0048492541 0.0061226039 0.0048566461 0.0068167857
  [6] 0.0044063350 0.0054392379 0.0067403456 0.0033642025 0.0028369841
 [11] 0.0043626513 0.0059685074 0.0090360051 0.0040963226 0.0034925534
...
[196] 0.0027203771 0.0037808188 0.0081262865 0.0040685285 0.0044834598

$max
[1]  0.8666279  0.5478021  1.3107427 -0.7595160 -1.3510582  0.9885595

$covar
```

```
            Var12       Var13       Var14        Var22        Var23
Var12  0.588263554  0.06313684  0.07088819  0.007553871 -0.005398027
Var13  0.063136836  0.63202630  0.07933276 -0.015982465 -0.040937948
Var14  0.070888189  0.07933276  0.67186035 -0.062542644 -0.025917076
Var22  0.007553871 -0.01598247 -0.06254264  0.533996254  0.064650234
Var23 -0.005398027 -0.04093795 -0.02591708  0.064650234  0.612430932
Var32  0.005892646 -0.04990347  0.04402294 -0.031610052 -0.072343996
            Var32
Var12  0.005892646
Var13 -0.049903465
Var14  0.044022941
Var22 -0.031610052
Var23 -0.072343996
Var32  0.462797374
```

The output contains the sample `$sample` in which each row is a draw from the posterior. The importance weight of each draw is given in `$weights`, the mode and the covariance matrix of the importance density are given respectively in `$max` and `$covar`.

Given the draws from the posterior distribution we can now select the next optimal choice set.

SeqMOD

The `SeqMOD` function can be used to select the next optimal choice set. The algorithm will evaluate each possible choice set in combination with the previously selected choice sets and select the one which maximizes efficiency, given the posterior preference distribution.

In the `SeqMOD` algorithm, the previously selected choice sets can be specified in `des`, which are stored in `init.des` in the example. The candidate set is the same as the one we used to generate `init.des`, namely `cs`. The sample we obtained from the posterior by using `ImpsampMNL` is saved in `draws`. The draws themselves are specified in `par.draws` and their importance weights in `weights`. Our prior covariance matrix `v` is passed through `prior.covar`. Optional arguments such as `alt.cte`, `no.choice` and `parallel` are the same as previously explained in the `Modfed` function. Lastly `reduce` is by default `TRUE` and reduces the set of all potential choice sets to a subset of choice sets that have a unique information matrix.

```
R> set <- SeqMOD(des = init.des, cand.set = cs, n.alts = 2,
+    par.draws =  draws$sample, prior.covar = v, weights = draws$weights)
R> set


$set
     Var12 Var13 Var14 Var22 Var23 Var32
[1,]     1     0     0     1     0     0
[2,]     0     1     0     0     1     1

$error
[1] 0.002874806
```

The output contains the most optimal next choice set given all possible choice sets, and the associated error.

It should be noted that the criterion used to evaluate the efficiency in the `SeqMOD` algorithm is slightly different from the one we specified in Equation 3. Since the adaptive approach is completely Bayesian, we wish to approximate the covariance matrix of the posterior preference distribution with the generalized Fisher information matrix (GFIM), which is computed as minus the Hessian matrix of the log-posterior density and is given by:

$$\mathcal{I}_{\text{GFIM}}\left(\boldsymbol{\beta}|\mathbf{X}\right) = \mathcal{I}_{\text{FIM}}\left(\boldsymbol{\beta}|\mathbf{X}\right) - \mathsf{E}_Y\left(\frac{\partial^2 \pi_0\left(\boldsymbol{\beta}\right)}{\partial\boldsymbol{\beta}\partial\boldsymbol{\beta}^\top}\right).$$

Here $\mathcal{I}_{\text{FIM}}\left(\boldsymbol{\beta}|\mathbf{X}\right)$ represents the Fisher information matrix of the MNL model as described in Equation 2. The second part represents the Fisher information contained in the prior density $\pi_0\left(\boldsymbol{\beta}\right)$. When a multivariate normal distribution is assumed as prior preference distribution, the second term simplifies to

$$\mathsf{E}_Y\left(\frac{\partial^2 \pi_0\left(\boldsymbol{\beta}\right)}{\partial\boldsymbol{\beta}\partial\boldsymbol{\beta}^\top}\right) = -\boldsymbol{\Sigma}_0^{-1},$$

i.e., minus the inverse of the covariance matrix of the prior preference distribution. The expression for the generalized Fisher information matrix then becomes:

$$\mathcal{I}_{\text{GFIM}}\left(\boldsymbol{\beta}|\mathbf{X}\right) = \mathcal{I}_{\text{FIM}}\left(\boldsymbol{\beta}|\mathbf{X}\right) + \boldsymbol{\Sigma}_0^{-1}.$$

The DB^A-error, i.e., the DB-error used in an adaptive scenario is then calculated as:

$$\text{DB}^A\text{-error} = \int \det\left(\mathcal{I}_{\text{GFIM}}\left(\boldsymbol{\beta}|\mathbf{X}\right)\right)^{-1/p} \pi_0(\boldsymbol{\beta}) d\boldsymbol{\beta}.$$

Note that the less informative the prior preference distribution is, i.e., the closer $\boldsymbol{\Sigma}_0^{-1}$ is to the zero matrix, the less difference there will be between the DB-error and the DB^A-error.

To set up a simulation study, the previous steps can be repeated as many times as additional adaptive choice sets are required. This can be done for N different participants, each time drawing a unique `truePREF` vector from an a priori specified population preference distribution. The researcher can vary the heterogeneity of this population distribution along with the number of adaptive choice sets and other specifications. The simulated choice data can be stored and prepared for estimation using existing R packages for which we refer to Section 6.

5. Real surveys with idefix

With the current package it is possible to conduct DCEs and collect empirical choice data by presenting choice sets on screen. The `SurveyApp` function can be used to present pregenerated designs, for example with the `Modfed` function, to participants. It can also be used to apply the IASB methodology in practice. If adaptive choice sets are required, the `SurveyApp` function will generate choice sets in the same way as described in Section 4.3. The choice data can be stored and easily loaded back into the R environment. In what follows, an example of a scenario with and without adaptive choice sets is given. Afterwards it is explained how one can use the `SurveyApp` function to deploy a **shiny** application online.

5.1. Discrete choice experiment without any adaptive sets

In the following example it is shown how to present a pregenerated design on screen and how to collect the users' responses.

The choice design itself should be specified in `des`, the structure of the design is the same throughout the package, namely a matrix in which each row is a profile. Choice sets consist of subsequent rows. For this example we use the example choice design `example_design`, which is included in the **idefix** package and is generated with the `Modfed` function, described in Section 3.3. In this choice design there are eight choice sets with 2 alternatives each. They consist of three attributes namely time, price and comfort.

```
R> data("example_design", package = "idefix")
R> xdes <- example_design
R> xdes
```

```
          Time1 Time2 Price1 Price2 Comfort1 Comfort2
set1.alt1     1     0      1      0        0        0
set1.alt2     0     1      0      0        1        0
set2.alt1     0     0      0      0        1        0
set2.alt2     1     0      1      0        0        0
set3.alt1     0     1      0      0        0        1
set3.alt2     0     0      0      1        0        0
set4.alt1     0     1      0      0        0        0
set4.alt2     1     0      1      0        1        0
set5.alt1     0     0      0      1        0        1
set5.alt2     0     1      0      1        0        0
set6.alt1     1     0      0      1        1        0
set6.alt2     0     1      1      0        0        0
set7.alt1     0     0      1      0        0        1
set7.alt2     1     0      0      0        1        0
set8.alt1     0     1      1      0        1        0
set8.alt2     1     0      0      1        0        1
```

The total number of choice sets that need to be presented are defined in the argument `n.total`. If no other choice sets besides the one in the specified design are desired, then `n.total` equals the number of choice sets in `des`. If `n.total` is larger than the number of sets provided in `des`, adaptive sets will be generated as explained in the third example. In `alts`, the names of the alternatives have to be specified. In this case there are two alternatives, named `Alt A` and `Alt B`. In `atts` the names of the attributes are specified, here `Price`, `Time`, and `Comfort`.

```
R> n.sets <- 8
R> alternatives <- c("Alt A", "Alt B")
R> attributes <- c("Price", "Time", "Comfort")
```

The attribute levels are specified in `lvl.names`, which is a list containing one character vector for each attribute. The levels for the first attribute are \$10, \$5, and \$1. The attribute time can take values 20, 12 and 3 minutes. Lastly, the comfort attribute can vary between bad, average, and good.

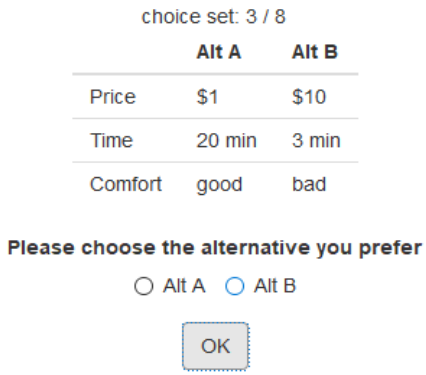

Figure 5: Example of a discrete choice experiment, generated with the `SurveyApp` function.

```
R> labels <- vector(mode = "list", length(attributes))
R> labels[[1]] <- c("$10", "$5", "$1")
R> labels[[2]] <- c("20 min", "12 min", "3 min")
R> labels[[3]] <- c("bad", "average", "good")
```

The type of coding used in `des` should be specified in `coding`. This is the same argument as explained in the `Profiles` function in Section 3.3. Here all attributes are dummy coded.

```
R> code <- c("D", "D", "D")
```

There are three arguments where some text can be provided. The character string `b.text` will appear above the options where the participant can indicate his or her choice. In this example the text `"Please choose the alternative you prefer"` appears in the choice task as can be seen in Figure 5. Before the discrete choice task starts, some instructions can be given. This text can be provided to `intro.text` in the form of a character string. In this case `"Welcome, here are some instructions ... good luck!"` will appear on screen before the survey starts. In the same way some ending note can be specified in `end.text`. The character string `"Thanks for taking the survey"` will appear on screen when the survey is completed.

```
R> b.text <- "Please choose the alternative you prefer"
R> i.text <- "Welcome, here are some instructions ... good luck!"
R> e.text <- "Thanks for taking the survey"
```

When running the `SurveyApp` function, a screen will pop up, starting with the initial text provided in `intro.text`. Next all the choice sets in the design provided in `des` will be presented on screen one after the other as can be seen in Figure 5.

```
R> 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,
+    data.dir = NULL)
```

Lastly the directory to store the observed responses, together with the presented design can be specified in `data.dir`. The default is `NULL`, and in this case no data will be stored. If a

file: 1542226175_char_data

	set	Alt A	Alt B	resp
Price	1	$5	$1	Alt B
Time	1	12 min	20 min	Alt B
Comfort	1	bad	average	Alt B
Price	2	$10	$5	Alt B
Time	2	20 min	12 min	Alt B
Comfort	2	average	bad	Alt B
Price	3	$1	$10	Alt A
Time	3	20 min	3 min	Alt A
Comfort	3	good	bad	Alt A
Price	4	$1	$5	Alt A
Time	4	20 min	12 min	Alt A
Comfort	4	bad	average	Alt A
Price	5	$10	$1	Alt B
Time	5	3 min	3 min	Alt B
Comfort	5	good	bad	Alt B
Price	6	$5	$1	Alt B
Time	6	3 min	12 min	Alt B
Comfort	6	average	bad	Alt B
Price	7	$10	$5	Alt B
Time	7	12 min	20 min	Alt B
Comfort	7	good	average	Alt B
Price	8	$1	$5	Alt A
Time	8	12 min	3 min	Alt A
Comfort	8	average	good	Alt A

file: 1542226175_num_data

	Time1	Time2	Price1	Price2	Comfort1	Comfort2	resp
set1.alt1	1	0	1	0	0	0	0
set1.alt2	0	1	0	0	1	0	1
set2.alt1	0	0	0	0	1	0	0
set2.alt2	1	0	1	0	0	0	1
set3.alt1	0	1	0	0	0	1	1
set3.alt2	0	0	0	1	0	0	0
set4.alt1	0	1	0	0	0	0	1
set4.alt2	1	0	1	0	1	0	0
set5.alt1	0	0	0	1	0	1	0
set5.alt2	0	1	0	1	0	0	1
set6.alt1	1	0	0	1	1	0	0
set6.alt2	0	1	1	0	0	0	1
set7.alt1	0	0	1	0	0	1	0
set7.alt2	1	0	0	0	1	0	1
set8.alt1	0	1	1	0	1	0	1
set8.alt2	1	0	0	1	0	1	0

Figure 6: Example of the two data files that get stored by the `SurveyApp` function.

directory is specified, two text files will be written to that directory at the end of the survey. One containing the decoded design as it was presented to the participant together with the sequence of alternatives that were chosen. This is similar to the `Decode` output previously described in Section 3.3. The second file has the same file name, which starts with the same number (based on `Sys.time()`), except for the fact that `"char"` is replaced with `"num"`. This file contains the coded design matrix and the binary response vector. This file can be used to estimate the preference parameters (see Section 6 for more information). The file containing the decoded design can be used to inspect the choice sets as they were perceived by the respondents. An example of both files can be seen in Figure 6.

If a no choice option is desired, the specified design should already contain a no choice alternative in each choice set. In this example we use a toy example design, generated with the `Modfed` function, and available in the **idefix** package. Except for the inclusion of a no choice alternative this design has the same properties as the previous example design.

```
R> data("nochoice_design", package = "idefix")
R> ncdes <- nochoice_design
R> ncdes
```

```
          no.choice.cte Time1 Time2 Price1 Price2 Comfort1 Comfort2
set1.alt1             0     0     1      1      0        1        0
set1.alt2             0     0     0      0      0        0        0
no.choice             1     0     0      0      0        0        0
set2.alt1             0     0     0      0      0        1        0
set2.alt2             0     1     0      1      0        0        1
no.choice             1     0     0      0      0        0        0
```

```
set3.alt1          0     0     1     0     0     0     1
set3.alt2          0     0     0     0     1     0     0
no.choice          1     0     0     0     0     0     0
set4.alt1          0     0     1     0     1     0     1
set4.alt2          0     1     0     1     0     0     0
no.choice          1     0     0     0     0     0     0
set5.alt1          0     0     0     0     0     0     1
set5.alt2          0     0     1     0     1     0     0
no.choice          1     0     0     0     0     0     0
set6.alt1          0     0     0     0     1     1     0
set6.alt2          0     0     1     0     0     0     0
no.choice          1     0     0     0     0     0     0
set7.alt1          0     0     0     1     0     0     1
set7.alt2          0     1     0     0     0     1     0
no.choice          1     0     0     0     0     0     0
set8.alt1          0     0     0     1     0     1     0
set8.alt2          0     1     0     0     1     0     1
no.choice          1     0     0     0     0     0     0
```

Now three alternative names are required. The last one indicates the no choice option.

```
R> alternatives <- c("Alternative A", "Alternative B", "None")
```

Since there is an alternative specific constant, the `alt.cte` argument should be specified. As before, this is done with a vector in which each element indicates whether an asc is present for the corresponding alternative or not. Furthermore, the option `no.choice`, which is by default `NULL`, should be altered to an integer indicating the no choice alternative.

```
R> SurveyApp(des = ncdes, n.total = n.sets, alts = alternatives,
+    atts = attributes, lvl.names = labels, coding = code,
+    alt.cte = c(0, 0, 1), no.choice = 3, buttons.text = b.text,
+    intro.text = i.text, end.text = e.text,  data.dir = NULL)
```

As can be seen in Figure 7, the no choice option is not part of the decoded alternatives, it is, however, possible to select *None* instead of *Alternative A* or *Alternative B*.

5.2. Discrete choice experiment containing adaptive sets

The `SurveyApp` function can also be used to generate sequential adaptive choice sets as explained in Section 4. Adaptive sets can be added to a pregenerated initial design or one can start without specifying any design in advance. An example with an initial design is given here.

As an initial design we use the design we introduced in the first example in Section 5.1. As a consequence, all arguments previously specified remain the same, except the number of sets `n.sets` is now set to twelve instead of eight, indicating we want four additional adaptive sets. Whenever `n.total` is larger than the number of choice sets in `des`, the `SurveyApp` will select the remaining number of sets by making use of the underlying `SeqMOD` algorithm (see

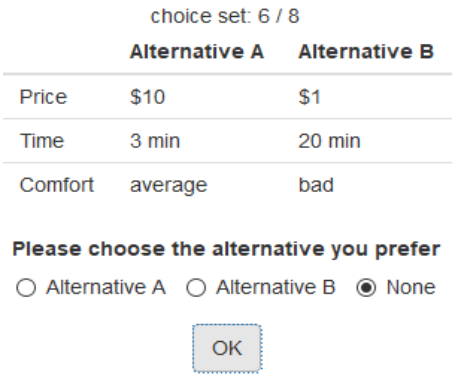

Figure 7: Example of a discrete choice experiment with a no choice option, generated with the `SurveyApp` function.

Section 4). In order to select the most efficient choice set based on the posterior probability of the preference parameters, the candidate set and the prior preference distribution needs to be given. In this example the mean vector `p.mean` consists of six elements, one for each parameter of the design. The covariance matrix `p.var` is a unit matrix, setting the prior variance for each of the parameters to one. The `Profiles` function is used, as explained in Section 3.3, to generate all possible profiles with three attributes, each containing three levels, and all are dummy coded. These should be the same characteristics as those of the initial design. Lastly, `n.draws` allows the user to specify how many draws from the posterior should be used in the underlying importance sampling procedure (see `ImpsampMNL` in Section 4.3).

```
R> n.sets <- 12
R> alternatives <- c("Alternative A", "Alternative B")
R> p.mean <- c(0.3, 0.7, 0.3, 0.7, 0.3, 0.7)
R> p.var <- diag(length(p.mean))
R> levels <- c(3, 3, 3)
R> cand <- Profiles(lvls = levels, coding = code)
R> 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 = 100)
```

```
          Time1 Time2 Price1 Price2 Comfort1 Comfort2  resp
set1.alt1     1     0      1      0        0        0     1
set1.alt2     0     1      0      0        1        0     0
set2.alt1     0     0      0      0        1        0     0
set2.alt2     1     0      1      0        0        0     1
set3.alt1     0     1      0      0        0        1     1
set3.alt2     0     0      0      1        0        0     0
set4.alt1     0     1      0      0        0        0     1
set4.alt2     1     0      1      0        1        0     0
set5.alt1     0     0      0      1        0        1     1
```

```
set5.alt2     0     1     0     1     0     0     0
set6.alt1     1     0     0     1     1     0     1
set6.alt2     0     1     1     0     0     0     0
set7.alt1     0     0     1     0     0     1     1
set7.alt2     1     0     0     0     1     0     0
set8.alt1     0     1     1     0     1     0     1
set8.alt2     1     0     0     1     0     1     0
```

The same files as in the previous example are saved in `data.dir` if a directory is specified. The design will now contain the initial choice sets and the additional four adaptive sets.

5.3. Online surveys

The previously described R code containing the `SurveyApp` function will launch a **shiny** application. This application will create a user interface with which the user can interact. This way it is possible to store responses. This procedure runs, however, locally on the computer where the R script is being evoked. More interesting is to gather data through an online survey. A way of doing this by remaining in the R environment is to deploy a **shiny** application to https://www.shinyapps.io/, a free server made available by RStudio (**RStudio** Team 2020) to host **shiny** applications. Unfortunately, at the moment there is no convenient way to directly deploy an application embedded in an R package to a **Shiny** server. It is, however, possible to disentangle the code of `SurveyApp` into a `ui.R` file and a `server.R` file, the files required to deploy applications. This can simply be done by printing the `SurveyApp` function in R and copy pasting the code behind `ui <-` into an R file, and to the same with the code behind `server <-` . To deploy the application online one can follow the clear explanation on https://shiny.rstudio.com/articles/shinyapps.html. After the application is deployed a web link where the survey can be found will be provided. This way data can be gathered by providing the web link to potential participants.

The only downside of this approach, for the time being, is that RStudio has not yet implemented persistent data storage on https://www.shinyapps.io/, they are, however, planning to do so. For now, this problem can be overcome by storing the data remotely. One can specify a temporary directory by using `tempdir()` in the `data.dir` argument of the `SurveyApp` function. Afterwards the files can be written to, for example, Dropbox with the **rdrop2** package (Ram and Yochum 2020), or to Amazon S3 with the **aws.s3** package (Leeper 2020) from that temporary directory. More information about remotely storing data while using https://www.shinyapps.io/ can be found on https://shiny.rstudio.com/articles/persistent-data-storage.html.

6. Data management

The **idefix** package always uses the same data format. An individual choice design is represented as a matrix in which each row is an alternative, and subsequent rows form choice sets (see for example the output of `Modfed` in Section 3.3). Individual choice data is contained in a binary vector with length equal to the number of alternatives (rows) in the design matrix. For each choice set the chosen alternative in indicated with a one and all others with zero. Individual design matrices as well as the choice vectors can be stacked together to form

aggregate choice designs and an aggregate data vector. The function `LoadData` can be used to do this. To estimate the preference parameters on the acquired data, several R packages can be used. Some of those packages, however, require different data formats. The function `Datatrans` allows one to switch between the format of **idefix** and the one of the desired estimation package.

6.1. Load data

Individual data files stored by `SurveyApp` in `data.dir` can be easily loaded back into R with the function `LoadData`. As explained in Section 5, two types of files will be stored in `data.dir`, a file containing character data and a file containing numeric data. For estimation purposes we need the numeric files, therefore we specify `type = "num"`. To get the character files one can specify `type = "char"`. In this case all files containing `"num"` in their file name will be loaded and stacked above each other. Each file, which represents the data of one respondent, will get a unique ID. All files in a specific directory should have the same number of columns (i.e., parameters).

Below an example is given with a directory containing data files generated by the `SurveyApp` function. There were seven participants who each responded to eight choice sets.

```
R> dataDir <- getwd()
R> data <- LoadData(data.dir = dataDir, type = "num")
R> data
```

```
   ID         X par.1 par.2 par.3 par.4 par.5 par.6 resp
1    1 set1.alt1     1     0     1     0     0     0    1
2    1 set1.alt2     0     1     0     0     1     0    0
3    1 set2.alt1     0     0     0     0     1     0    0
4    1 set2.alt2     1     0     1     0     0     0    1
5    1 set3.alt1     0     1     0     0     0     1    1
6    1 set3.alt2     0     0     0     1     0     0    0
7    1 set4.alt1     0     1     0     0     0     0    0
8    1 set4.alt2     1     0     1     0     1     0    1
9    1 set5.alt1     0     0     0     1     0     1    1
10   1 set5.alt2     0     1     0     1     0     0    0
11   1 set6.alt1     1     0     0     1     1     0    0
12   1 set6.alt2     0     1     1     0     0     0    1
13   1 set7.alt1     0     0     1     0     0     1    1
14   1 set7.alt2     1     0     0     0     1     0    0
15   1 set8.alt1     0     1     1     0     1     0    0
16   1 set8.alt2     1     0     0     1     0     1    1
17   2 set1.alt1     1     0     1     0     0     0    0
18   2 set1.alt2     0     1     0     0     1     0    1
19   2 set2.alt1     0     0     0     0     1     0    0
20   2 set2.alt2     1     0     1     0     0     0    1
...
111  7 set8.alt1     0     1     1     0     1     0    1
112  7 set8.alt2     1     0     0     1     0     1    0
```

6.2. Data transformation

`Datatrans`

To illustrate the data transformation, we make use of an example of an aggregate dataset included in **idefix**. The dataset `aggregate_design` is the same one as shown at the end of Section 6.1. There are seven respondents who each responded to eight choice sets containing two alternatives. The alternatives consist of three dummy coded attributes with each three levels. In order to use the `Datatrans` function we need to split the design matrix from the responses. To get the design matrix we remove the ID column (the first column), the column indicating the set number (second column), and the response column (column nine). What remains is the design matrix.

```
R> idefix.data <- aggregate_design
R> des <- as.matrix(idefix.data[, 3:8], ncol = 6)
R> des
```

```
      par.1 par.2 par.3 par.4 par.5 par.6
 [1,]     1     0     1     0     0     0
 [2,]     0     1     0     0     1     0
 [3,]     0     0     0     0     1     0
 [4,]     1     0     1     0     0     0
 [5,]     0     1     0     0     0     1
 [6,]     0     0     0     1     0     0
 [7,]     0     1     0     0     0     0
 [8,]     1     0     1     0     1     0
 [9,]     0     0     0     1     0     1
[10,]     0     1     0     1     0     0
[11,]     1     0     0     1     1     0
[12,]     0     1     1     0     0     0
[13,]     0     0     1     0     0     1
[14,]     1     0     0     0     1     0
[15,]     0     1     1     0     1     0
[16,]     1     0     0     1     0     1
[17,]     1     0     1     0     0     0
[18,]     0     1     0     0     1     0
[19,]     0     0     0     0     1     0
 ...
[111,]    0     1     1     0     1     0
[112,]    1     0     0     1     0     1
```

All responses are captured in a binary vector containing the choice data of all respondents. Since there were seven respondents who each responded to eight choice sets containing two alternatives, `y` contains 112 elements.

```
R> y <- idefix.data[, 9]
R> y
```

```
  [1] 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 0 1 1
 [36] 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1 1 0 1 0 1 0 1 0 1 0 0 1 1 0
 [71] 0 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0
[106] 1 0 1 0 1 1 0
```

The function `Datatrans` can be used to transform the data into the correct format in order to use existing estimation packages in R. In the `pkg` argument one can specify the package for which the data needs to be transformed. There are six options: `"bayesm"` indicates the `rhierMnlRwMixture` function of package **bayesm**, `"ChoiceModelR"` should be used for the `choicemodelr` function of package **ChoiceModelR**, `"RSGHB"` for the `doHB` function of the **RSGHB** package, `"Mixed.Probit"` for the `rbprobitGibbs` function of the **bayesm** package, `"mlogit"` for the `mlogit` function of package **mlogit** and lastly `"Rchoice"` for the `Rchoice` function of the **Rchoice** package.

Furthermore, the number of alternatives per choice set `n.alts`, the number of choice sets per respondent `n.sets`, the number of respondents `n.resp`, and the number of parameters `n.par` should be specified. If the response vector `y` is not in binary format, but in discrete format (one integer indicating the chosen alternative per choice set), the `bin` argument has to be set to `FALSE`. The `alt.var` argument which is by default `NULL` can be specified if the names of the alternatives should be different. This requires a character vector with length equal to the number of alternatives in each choice set. This only has influence on the output for `mlogit` and `Rchoice`.

```
R> Datatrans(pkg = "bayesm", des = des, y = y, n.alts = 2,
+    n.sets = 8, n.resp = 7, bin = TRUE)
```

In the example above, the data is transformed in order to use the `rhierMnlRwMixture` function of package **bayesm**. The data format required is a list with elements `$lgtdata`, and `$p` where `$lgtdata` is a list of `nlgt = length(lgtdata)` of length equal to the number of respondents. Each element contains a list containing the data for each cross-sectional unit MNL: `lgtdata[[i]]$y` is a vector of responses with one element per choice set indicating the chosen alternative for each respondent. `lgtdata[[i]]$X` is the design matrix for each respondent, and `p` is the number of alternatives in each choice set. The output of `Datatrans` is the following (only the first respondent is shown):

```
$p
[1] 2

$lgtdata
$lgtdata[[1]]
$lgtdata[[1]]$y
[1] 1 2 1 2 1 2 1 2

$lgtdata[[1]]$X
      par.1 par.2 par.3 par.4 par.5 par.6
 [1,]     1     0     1     0     0     0
 [2,]     0     1     0     0     1     0
 [3,]     0     0     0     0     1     0
```

```
 [4,]    1    0    1    0    0    0
 [5,]    0    1    0    0    0    1
 [6,]    0    0    0    1    0    0
 [7,]    0    1    0    0    0    0
 [8,]    1    0    1    0    1    0
 [9,]    0    0    0    1    0    1
[10,]    0    1    0    1    0    0
[11,]    1    0    0    1    1    0
[12,]    0    1    1    0    0    0
[13,]    0    0    1    0    0    1
[14,]    1    0    0    0    1    0
[15,]    0    1    1    0    1    0
[16,]    1    0    0    1    0    1
```

In the following example, the same data is transformed in order to use the **mlogit** package for estimation. The output is a `data.frame` in long format that contains a boolean choice variable `Choice`, the index of the alternative `alt.names` and an individual index `id.var`.

```
R> Datatrans(pkg = "mlogit", des = des, y = y,
+    alts = 2, n.sets = 8, n.resp = 7, bin = TRUE)

[1] "The dataset is ready to be used for mlogit package"
~~~~~~~
 first 10 observations out of 112
~~~~~~~
   id.var     alt.names par.1 par.2 par.3 par.4 par.5 par.6 Choice id1    idx
1       1 alternative.1     1     0     1     0     0     0   TRUE   1 1:ve.1
2       1 alternative.2     0     1     0     0     1     0  FALSE   1 1:ve.2
3       1 alternative.1     0     0     0     0     1     0  FALSE   2 2:ve.1
4       1 alternative.2     1     0     1     0     0     0   TRUE   2 2:ve.2
5       1 alternative.1     0     1     0     0     0     1   TRUE   3 3:ve.1
6       1 alternative.2     0     0     0     1     0     0  FALSE   3 3:ve.2
7       1 alternative.1     0     1     0     0     0     0  FALSE   4 4:ve.1
8       1 alternative.2     1     0     1     0     1     0   TRUE   4 4:ve.2
9       1 alternative.1     0     0     0     1     0     1   TRUE   5 5:ve.1
10      1 alternative.2     0     1     0     1     0     0  FALSE   5 5:ve.2

~~~ indexes ~~~~
   chid           alt
1     1 alternative.1
2     1 alternative.2
3     2 alternative.1
4     2 alternative.2
5     3 alternative.1
6     3 alternative.2
7     4 alternative.1
8     4 alternative.2
```

```
9     5 alternative.1
10    5 alternative.2
indexes:  1, 2
```

7. Future development

As previously mentioned, the package is mainly concerned with providing D(B) efficient designs. Of course any statistic or feature that is useful to practitioners in selecting an appropriate design can be included. We therefore encourage users to give feedback and suggestions to further improve the package. The most recent updates can be followed on https://github.com/traets/idefix. We plan to implement the following in the near future:

- We are currently evaluating the performance of a sequential selection algorithm based on a coordinate exchange approach. This would allow to speed up computations for adaptive design methodology.
- We plan to implement a blocking procedure that would allow to split a larger design in several sub designs based on, for example, attribute level balance.
- We would like to extend the main algorithms such that the user can impose further restrictions on the designs (e.g., not allowing specific attribute level combinations for particular labeled alternatives).
- In general, the layout of the output will be improved by making use of methods such as `print` and `summary`.

References

Abrantes PAL, Wardman MR (2011). "Meta-Analysis of UK Values of Travel Time: An Update." *Transportation Research Part A: Policy and Practice*, **45**(1), 1–17. doi:10.1016/j.tra.2010.08.003.

Aizaki H (2012). "Basic Functions for Supporting an Implementation of Choice Experiments in R." *Journal of Statistical Software, Code Snippets*, **50**(2), 1–24. doi:10.18637/jss.v050.c02.

Bliemer MC, Rose JM (2010a). "Serial Choice Conjoint Analysis for Estimating Discrete Choice Models." In S Hess, A Daly (eds.), *Choice Modelling: The State-of-the-Art and The State-of-Practice*, pp. 137–161. Emeral Publishing Group. doi:10.1108/9781849507738-006.

Bliemer MCJ, Rose JM (2010b). "Construction of Experimental Designs for Mixed Logit Models Allowing for Correlation Across Choice Observations." *Transportation Research Part B: Methodological*, **44**(6), 720–734. doi:10.1016/j.trb.2009.12.004.

Bliemer MCJ, Rose JM, Chorus CG (2017). "Detecting Dominance in Stated Choice Data and Accounting for Dominance-Based Scale Differences in Logit Models." *Transportation Research Part B: Methodological*, **102**, 83–104. doi:10.1016/j.trb.2017.05.005.

Bliemer MCJ, Rose JM, Hess S (2008). "Approximation of Bayesian Efficiency in Experimental Choice Designs." *Journal of Choice Modelling*, **1**(1), 98–126. doi:10.1016/s1755-5345(13)70024-1.

Bridges JFP, *et al.* (2011). "Conjoint Analysis Applications in Health – A Checklist: A Report of the ISPOR Good Research Practices for Conjoint Analysis Task Force." *Value in Health*, **14**(4), 403–413. doi:10.1016/j.jval.2010.11.013.

Chang W, Cheng J, Allaire JJ, Xie Y, McPherson J (2020). ***shiny**: Web Application Framework for R*. R package version 1.5.0, URL https://CRAN.R-project.org/package=shiny.

Cook RD, Nachtsheim CJ (1980). "A Comparison of Algorithms for Constructing Exact D-Optimal Designs." *Technometrics*, **22**(3), 315–324. doi:10.1080/00401706.1980.10486162.

Crabbe M, Akinc D, Vandebroek M (2014). "Fast Algorithms to Generate Individualized Designs for the Mixed Logit Choice Model." *Transportation Research Part B: Methodological*, **60**, 1–15. doi:10.1016/j.trb.2013.11.008.

Crabbe M, Vandebroek M (2012). "Using Appropriate Prior Information to Eliminate Choice Sets with a Dominant Alternative from D-Efficient Designs." *Journal of Choice Modelling*, **5**(1), 22–45. doi:10.1016/s1755-5345(13)70046-0.

Croissant Y (2020). "Estimation of Random Utility Models in R: The **mlogit** Package." *Journal of Statistical Software*, **95**(11), 1–41. doi:10.18637/jss.v095.i11.

Danthurebandara VM, Yu J, Vandebroek M (2011). "Sequential Choice Designs to Estimate the Heterogeneity Distribution of Willingness-to-Pay." *Quantitative Marketing and Economics*, **9**(4), 429–448. doi:10.1007/s11129-011-9106-3.

Dumont J, Keller J (2019). ***RSGHB**: Functions for Hierarchical Bayesian Estimation: A Flexible Approach*. R package version 1.2.2, URL https://CRAN.R-project.org/package=RSGHB.

Fedorov VV (1972). *Theory of Optimal Experiments*. Probability and Mathematical Statistics 12. Academic Press, New York.

Harman R, Filova L (2019). ***OptimalDesign**: A Toolbox for Computing Efficient Designs of Experiments*. R package version 1.0.1, URL https://CRAN.R-project.org/package=OptimalDesign.

Hensher DA, Greene WH (2003). "The Mixed Logit Model: The State of Practice." *Transportation*, **30**(2), 133–176. doi:10.1023/a:1022558715350.

Horne J (2018). ***choiceDes**: Design Functions for Choice Studies*. R package version 0.9-3, URL https://CRAN.R-project.org/package=choiceDes.

Huber J, Zwerina K (1996). "The Importance of Utility Balance in Efficient Choice Designs." *Journal of Marketing Research*, **33**(3), 307–317. doi:10.2307/3152127.

Johnson FR, *et al.* (2013). "Constructing Experimental Designs for Discrete-Choice Experiments: Report of the ISPOR Conjoint Analysis Experimental Design Good Research Practices Task Force." *Value in Health*, **16**(1), 3–13. doi:10.1016/j.jval.2012.08.2223.

Kessels R, Goos P, Vandebroek M (2006). "A Comparison of Criteria to Design Efficient Choice Experiments." *Journal of Marketing Research*, **43**(3), 409–419. doi:10.1509/jmkr.43.3.409.

Leeper TJ (2020). ***aws.s3**: **AWS S3** Client Package*. R package version 0.3.21, URL https://CRAN.R-project.org/package=aws.s3.

Marschak J (1950). "Rational Behavior, Uncertain Prospects, and Measurable Utility." *Econometrica*, **18**(2), 111–141. doi:10.2307/1907264.

McFadden D (1974). "Conditional Logit Analysis of Qualitative Choice Behavior." In *Frontiers in Econometrics*, pp. 105–142. Academic Press.

McFadden D, Train K (2000). "Mixed MNL Models for Discrete Response." *Journal of Applied Econometrics*, **15**(5), 447–470. doi:10.1002/1099-1255(200009/10)15:5<447::aid-jae570>3.0.co;2-1.

Meyer RK, Nachtsheim CJ (1995). "The Coordinate-Exchange Algorithm for Constructing Exact Optimal Experimental Designs." *Technometrics*, **37**(1), 60–69. doi:10.1080/00401706.1995.10485889.

Pinto D, Danilovich MK, Hansen P, Finn DJ, Chang RW, Holl JL, Heinemann AW, Bockenholt U (2017). "Qualitative Development of a Discrete Choice Experiment for Physical Activity Interventions to Improve Knee Osteoarthritis." *Archives of Physical Medicine and Rehabilitation*, **98**(6), 1210–1216.e1. doi:10.1016/j.apmr.2016.11.024.

Ram K, Yochum C (2020). ***rdrop2**: Programmatic Interface to the 'Dropbox' API*. R package version 0.8.2.1, URL https://CRAN.R-project.org/package=rdrop2.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org.

Rose JM, Bliemer MC (2014). "Stated Choice Experimental Design Theory: The Who, the What and the Why." In S Hess, A Daly (eds.), *Handbook of Choice Modelling*, pp. 152–177. Edward Elgar Publishing, Cheltenham.

Rose JM, Bliemer MCJ (2009). "Constructing Efficient Stated Choice Experimental Designs." *Transport Reviews*, **29**(5), 587–617. doi:10.1080/01441640902827623.

Rose JM, Bliemer MCJ, Hensher DA, Collins AT (2008). "Designing Efficient Stated Choice Experiments in the Presence of Reference Alternatives." *Transportation Research Part B: Methodological*, **42**(4), 395–406. doi:10.1016/j.trb.2007.09.002.

Rossi P (2019). ***bayesm**: Bayesian Inference for Marketing/Micro-Econometrics*. R package version 3.1-4, URL https://CRAN.R-project.org/package=bayesm.

RStudio Team (2020). ***RStudio**: Integrated Development Environment for R*. **RStudio**, PBC., Boston. URL http://www.rstudio.com/.

Sarrias M (2016). "Discrete Choice Models with Random Parameters in R: The **Rchoice** Package." *Journal of Statistical Software*, **74**(10), 1–31. doi:10.18637/jss.v074.i10.

Sermas R (2012). ***ChoiceModelR**: Choice Modeling in R*. R package version 1.2, URL https://CRAN.R-project.org/package=ChoiceModelR.

Tian T, Yang M (2017). "Efficiency of the Coordinate-Exchange Algorithm in Constructing Exact Optimal Discrete Choice Experiments." *Journal of Statistical Theory and Practice*, **11**(2), 254–268. doi:10.1080/15598608.2016.1203842.

Traets F (2020). ***idefix**: Efficient Designs for Discrete Choice Experiments*. R package version 1.0.1, URL https://CRAN.R-project.org/package=idefix.

Train KE (2003). *Discrete Choice Methods with Simulation*. Cambridge University Press. doi:10.1017/cbo9780511753930.

Walker JL, Wang Y, Thorhauge M, Ben-Akiva M (2018). "D-Efficient or Deficient? A Robustness Analysis of Stated Choice Experimental Designs." *Theory and Decision*, **84**(2), 215–238. doi:10.1007/s11238-017-9647-3.

Wardman M, Chintakayala VPK, de Jong G (2016). "Values of Travel Time in Europe: Review and Meta-Analysis." *Transportation Research Part A: Policy and Practice*, **94**, 93–111. doi:10.1016/j.tra.2016.08.019.

Wheeler B (2019). ***AlgDesign**: Algorithmic Experimental Design*. R package version 1.2-0, URL https://CRAN.R-project.org/package=AlgDesign.

Yu J, Goos P, Vandebroek M (2011). "Individually Adapted Sequential Bayesian Conjoint-Choice Designs in the Presence of Consumer Heterogeneity." *International Journal of Research in Marketing*, **28**(4), 378–388. doi:10.1016/j.ijresmar.2011.06.002.

Affiliation:

Frits Traets
Faculty of Economics and Business Research
KU Leuven
Naamsestraat 69 bus 3500
3000 Leuven, Belgium
E-mail: frits.traets@gmail.com

Martina Vandebroek
Leuven Statistics Research Centre
KU Leuven
Celestijnenlaan 200B
3001 Leuven, Belgium
E-mail: Martina.Vandebroek@kuleuven.be

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
November 2020, Volume 96, Issue 3 *Submitted:* 2017-08-22
doi:10.18637/jss.v096.i03 *Accepted:* 2019-11-10