



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

Frits Traets
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 based on the multinomial logit model. In addition, the package provides the necessary tools to set up online surveys with the possibility of making use of the individual adaptive sequential Bayesian design approach for estimating the mixed logit model. After data collection the package 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 (DCE's) 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. Commonly derived statistics are for example value of time (VOT) or willingness to pay (WTP). The use of stated preference data, and DCE's in particular, has strongly increased the past decades in fields such as transportation, environmental economy, health economy, 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's) 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 those is that people maximize their utility, which is modeled as a function of the preference weights and attribute levels. The utility is most often linearly specified in the parameters, but the corresponding logit probabilities relate nonlinear to the observed utility.

At first, choice designs (i.e., the combination of several choice sets) were build 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: orthogonality, utility balance, task complexity, response efficiency, attribute balance, and statistical efficiency (see Rose and Bliemer 2014, for an overview). 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 goal of such designs is to maximize the statistical information gained from conducting a DCE, or alternatively phrased, to minimize the confidence ellipsoids around the parameter estimates. When setting up a DCE, the researcher is often restricted by the limited number of choice sets one can present to a respondent, and the limited number of respondents one can question. On the other hand, 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. Some of those potential choice sets will be very informative while others will be less. An optimal design approach will select those choice sets that force the respondent to make trade-offs, hereby maximizing the information gained from each observed choice. A problem with the design of experiments for DCE's is that the efficiency depends on the unknown parameters (Huber and Zwerina 1996).

Some general experimental design R packages exist (e.g., **AlgDes** (Wheeler 2014) and **OptimalDesign** (Harman and Filova 2016)), which can be used for the generation of optimal designs. However none of them is ideal for DCE's, since they apply methods designed for linear models, whereas choice models are nonlinear (Train 2003). To our knowledge two R packages exist that are promoted for designing DCE's. 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 2014), 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, because some linearly optimized designs are also optimal for MNL models when the preference parameters are assumed to be zero. However, most often we do have valid prior information on the preference structure, in which case experimental designs that take this into account will outperform the ones who don't (Kessels, Goos, and Vandebroek 2006). Thus so far, no R package is suited to generate optimal designs for DCE's. In addition, there is no software that implements the individually adaptive sequential Bayesian design (IASB) methodology, which provides adaptive designs for the MIXL model. More specifically, there are no R packages that provide any kind of adaptive designs for DCE's in general.

The outline of this paper is as follows: Section 2 explains how to generate statistically optimal designs for the MNL model. In Section 3, the idea behind the IASB methodology is made clear 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 4 describes the **SurveyApp** function which enables the researcher to gather empirical choice data by launching an interactive **shiny** application. This function can be used with our without making use of the IASB methodology and can be used to set up online surveys. Lastly in Section 5, it is shown how to switch between different data formats in order to use existing estimation software in R (R Development Core Team

2008).

2. Non sequential D(B)-optimal designs for the MNL model

2.1. The multinomial logit model

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

$$U_{ksn} = \mathbf{x}_{ksn}^\top \boldsymbol{\beta} + \epsilon_{ksn}. \quad (1)$$

The p -dimensional preference vector $\boldsymbol{\beta}$ denotes the importance of all attribute levels, represented by the p -dimensional vector \mathbf{x}_{ksn} . Unobserved influences on the utility function are captured by the error term ϵ_{ksn} , which is 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:

$$p_{ksn}(\boldsymbol{\beta}) = \frac{\exp(\mathbf{x}_{ksn}^\top \boldsymbol{\beta})}{\sum_{i=1}^K \exp(\mathbf{x}_{isn}^\top \boldsymbol{\beta})}. \quad (2)$$

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

2.2. 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 of that model by calculating the Fisher Information matrix

$$\mathcal{I} = -E \left(\frac{\partial^2 L}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^\top} \right). \quad (3)$$

For the MNL model, this comes down to the following expression, where N is the number of respondents, $\mathbf{X}_s = (x_{1sn}^\top, \dots, x_{Ksn}^\top)$, the design matrix of choice set s , $\mathbf{P}_s = \text{diag}[p_{1sn}, p_{2sn}, \dots, p_{Ksn}]$, and $\mathbf{p}_s = [p_{1sn}, p_{2sn}, \dots, p_{Ksn}]^\top$:



$$\mathcal{I}(\boldsymbol{\beta}|\mathbf{X}) = N \sum_{s=1}^S \mathbf{X}_s^\top \left(\mathbf{P}_s - \mathbf{p}_s \mathbf{p}_s^\top \right) \mathbf{X}_s. \quad (4)$$

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:

$$\boldsymbol{\Omega} = \mathcal{I}(\boldsymbol{\beta}|\mathbf{X})^{-1}, \quad (5)$$

$$D - error = \det(\Omega)^{1/p}. \quad (6)$$

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 can be defined on those parameters. In this case the expected D-error is minimized over the prior distribution and is referred to as the DB-error

$$D_B - error = \int \det(\Omega)^{1/p} \pi(\beta) d\beta. \quad (7)$$

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 the modified Federov algorithm, which was adapted from the classical Federov exchange algorithm (Fedorov 1972). The algorithm swaps profiles from an initial random design matrix by candidate profiles in order to minimize the D or DB-error.


2.3. Optimal designs for the MNL model with package *idefix*

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 one 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.

```
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)
```

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

	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
...						
24	-1	-1	-1	-1	-1	-1

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 are two continuous attributes, 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)
```

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

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

Modfed

A modified Federov algorithm is implemented and can be used with the `Modfed` function. The first argument `cand.set` is a matrix with all possible profiles. 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 possible alternative specific constants `alt.cte` should be specified. The algorithm will swap profiles from `cand.set` with profiles from an initial design in order to maximize the D(B)-efficiency. By specifying a numeric vector in `par.draws`, the D-error will be calculated and the design will be optimized locally for these parameter values and a MNL model.

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. The number of columns should equal the number of parameters in `alt.cte` + the number of parameters in `cand.set`. This is also the order in which they should be sorted (first `alt.cte` parameters). An initial design can be provided in `start.des`, if not, a random start design will be generated first.

The modified algorithm is fairly rapid, however it is not guaranteed that it will find the optimal design since there is no exhaustive search possible. It is thus highly recommended to run the algorithm several times starting with different initial designs, to avoid getting stuck in local optima. 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 D-optimal design is generated for a scenario with three attributes. The first attribute has four levels and is dummy coded. The second attribute is treated as continuous and has three levels (i.e., 2, 4, and 6). The third attribute is again dummy coded and has three levels. A vector of parameter values is specified in `ps` and therefore the design will be optimized locally (D-error).

```
R> set.seed(123)
R> cs <- Profiles(lvls = c(4, 3, 3), coding = c("D", "C", "D"),
+   c.lvls = list(c(2, 4, 6)))
R> ps <- c(0.8, 0.2, 1.2, -0.3, 0.4, 0.8)
R> Modfed(cand.set = cs, n.sets = 8, n.alts = 2,
+   alt.cte = c(0, 0), par.draws = ps)
```

The output consist of the optimized design and the associated D-error. Furthermore `$inf.error` denotes the percentage of draws for which the design resulted in an infinite D-error. This could happen for extreme large parameter values, which result in probabilities of one or zero for all alternatives in all choice sets. In that case the elements of the information matrix will be zero, and the D-error will be infinite. This percentage should thus be preferably close to zero when calculating the DB-error and zero when generating a D-optimal design. Lastly `$prob.diff` shows the difference between the maximum and minimum probability of the alternatives in each choice set. Large differences indicate dominant alternatives, which are usually not desired (Crabbe and Vandebroek 2012; Bliemer, Rose, and Chorus 2017).

```
$design
      Var12 Var13 Var14 Var2 Var32 Var33
set1.alt1    1    0    0    2    0    0
set1.alt2    0    1    0    6    0    1
set2.alt1    0    1    0    2    1    0
set2.alt2    0    0    1    6    0    0
...
set8.alt1    0    0    0    4    1    0
set8.alt2    0    0    1    4    0    1

$error
[1] 0.8376661

$inf.error
[1] 0

$prob.diff
[1] 4.621172e-01 2.913126e-01 9.966799e-02 1.973753e-01 3.799490e-01
[6] 1.110223e-16 3.799490e-01 6.640368e-01
```

In the second example a matrix with multiple draws from a prior distribution is specified in `par.draws`, therefore a DB-optimal design will be generated. The scenario has again three attributes with respectively four, three, and two levels, all of which are effects coded. Because an alternative specific constant is specified in `alt.cte`, the dimensionality of the prior distribution is seven (one parameter for the alternative specific constant and six parameters for the coded attribute levels).

```

R> set.seed(123)
R> cs <- Profiles(lvls = c(4, 3, 2), coding = c("E", "E", "E"))
R> m <- c(0.4, 0.8, 0.2, 1.2, -0.3, -0.2, 0.7)
R> v <- diag(length(m))
R> ps <- MASS::mvrnorm(n = 10, mu = m, Sigma = v)
R> Modfed(cand.set = cs, n.sets = 8, n.alts = 2, alt.cte = c(1, 0),
+       par.draws = ps)

```

The output is the same as in the previous example, except that now the DB-error instead of the D-error is given.

```

$design
      alt1.cte Var11 Var12 Var13 Var21 Var22 Var31
set1.alt1      1     1     0     0     0     1    -1
set1.alt2      0     0     0     1    -1    -1    -1
set2.alt1      1     1     0     0    -1    -1     1
set2.alt2      0     0     1     0     0     1     1
...
set8.alt1      1    -1    -1    -1    -1    -1     1
set8.alt2      0     0     0     1     0     1    -1

$error
[1] 1.750335

$inf.error
[1] 0

$prob.diff
[1] 0.6379158 0.6518213 0.5060532 0.4193435 0.5405491 0.6354334
[7] 0.4690250 0.5983345

```

3. Individually Adaptive Sequential Bayesian designs for the MIXL model

3.1. The mixed logit model

Essentially the MIXL model 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 β , is then expressed as follows:

$$\pi_{ks} = \int p_{ks}(\beta) f(\beta) d\beta. \quad (8)$$

The logit probability $p_{ks}(\beta)$ as specified in equation 2, is weighted over the density function $f(\beta)$, describing the preference heterogeneity. The likelihood of the observed responses of all respondents \mathbf{y}_{full} , a binary vector indicating the chosen alternatives, can then be written as

$$L(\mathbf{y}_{\text{full}}|\mathbf{X}_{\text{full}}, \boldsymbol{\theta}) = \prod_{n=1}^N \int \left(\prod_{s=1}^S \prod_{k=1}^K (p_{ksn}(\boldsymbol{\beta}_n))^{y_{ksn}} \right) f(\boldsymbol{\beta}_n|\boldsymbol{\theta}) d\boldsymbol{\beta}_n, \quad (9)$$

where \mathbf{X}_{full} is the full design matrix containing the subdesigns for N respondents, and $\boldsymbol{\theta}$ containing 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.

3.2. Optimal designs for the mixed logit model

Because the Fisher information matrix for the MIXL model cannot be computed independent of the observed responses, generating DB-optimal designs requires extensive simulations and becomes quickly too time consuming (Bliemer and Rose 2010). Another approach has been proposed by Yu, Goos, and Vandebroek (2011), which exploits the idea that the MIXL model assumes respondents to choose according to an 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 and adaptive because each choice set is selected based on all available information about the respondent's preferences at that time.

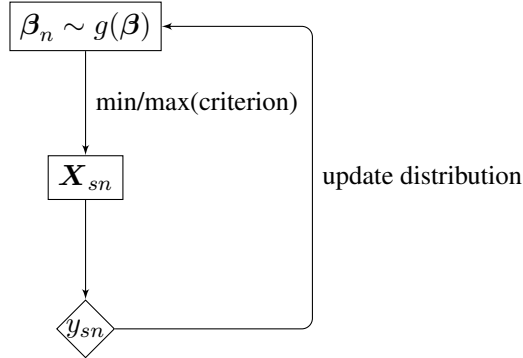


Figure 1: IASB algorithm

The flow chart in Figure 1 represents the idea of the IASB approach. At the top we see the distribution of the individual preferences of a respondent β_n . Initially samples are drawn from the prior distribution $\beta_n \sim \mathcal{N}(\mu, \Sigma)$. Based on those samples, one or more initial choice sets X will be selected by minimizing or maximizing a certain efficiency criterion. After observing the respons(es) y , the prior is updated in a Bayesian way and samples are now drawn from the posterior distribution $g(\beta)$. This process is repeated as many times as there are individual sequential choice sets, resulting in an individualised 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, Yu, and Vandebroek 2011; Yu *et al.* 2011). As an efficiency criterion the DB-error can be evaluated for each possible choice set, or to speed up the generation of a choice set, the Kullback-Leibler criterion can be used, as explained next.

Kullback-Leibler criterion

The Kullback-Leibler (KL) divergence between two density functions m and n is defined by [Kullback and Leibler \(1951\)](#) as follows:

$$KL(m, n) = E_m \left[\log \frac{m(x)}{n(x)} \right].$$

The KL divergence will increase when the two density functions differ more from each other. In the IASB design approach, the divergence between two subsequent posteriors on the parameters can be used as a measure of the information that is gained by presenting a particular choice set ([Crabbe, Akinc, and Vandebroek 2014](#)). A choice set resulting in a higher KL divergence is a more informative choice set than one that results in a lower KL divergence. The selection of choice set X_{sn} is then based on the expected KL divergence between the current posterior and the posterior after observing response y_{sn} . In order to do so, the following expression is maximized

$$KLP = \sum_{k=1}^K \pi(y_{ksn} | \mathbf{y}_n^{s-1}) KL \left[f(\beta_n | \mathbf{y}_n^{s-1}), f(\beta_n | \mathbf{y}_n^{s-1}, y_{ksn}) \right],$$

where s denotes the next choice set, n a particular respondent and k the chosen alternative. The posterior weighted choice probabilities for the alternatives are defined as

$$\pi(y_{ksn} | \mathbf{y}_n^{s-1}) = \int p_{ksn}(\beta_n) f(\beta_n | \mathbf{y}_n^{s-1}) d\beta_n.$$

3.3. Optimal designs for the MIXL model with package idefix

In this part, the functions necessary for generating adaptive choice sets in a sequential way, as explained in Section 3.2, are described. The main building blocks of the IASB approach consist of an importance sampling algorithm `ImpsampMNL` and a choice set selection algorithm. The latter can be either based on the DB criterion implemented in the `SeqDB` function, or based on the Kullback-Leibler criterion, implemented in the `SeqKL` function. Furthermore, a function to generate responses `RespondMNL` is also included, this way all elements to set up a simulation study, as depicted in Figure 2, are present. If the reader has no interest in simulating choice data while making use of the IASB approach but rather in collecting empirical data, one can proceed to Section 4.

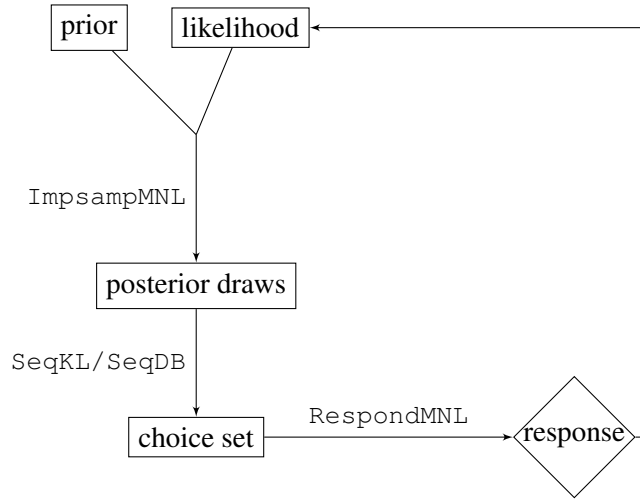


Figure 2: Simulation setup for the IASB approach.

In what follows an example of the workflow, together with a description of each of the functions involved, is given. 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 effects 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 \mathbf{m} and covariance matrix \mathbf{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("E", "E", "E"))
R> m <- c(0.5, 0.5, 1, -0.3, -0.7, 0.7)
R> v <- diag(length(m))
R> ps <- MASS::mvrnorm(n = 10, 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

```

	Var11	Var12	Var13	Var21	Var22	Var31
set1.alt1	-1	-1	-1	1	0	-1
set1.alt2	-1	-1	-1	0	1	1
set2.alt1	0	0	1	-1	-1	-1
set2.alt2	0	1	0	1	0	1
set3.alt1	-1	-1	-1	0	1	-1
set3.alt2	-1	-1	-1	1	0	1
set4.alt1	1	0	0	1	0	1
set4.alt2	0	0	1	1	0	1
set5.alt1	-1	-1	-1	-1	-1	1

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

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(0.8, 1, 1.2, -0.4, -0.8, 1.3)
```



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
```

The output is a binary vector indicating the simulated choices. In this case, for the first five choice sets the second alternative is chosen, whereas for the last three the first alternative is chosen.

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

At this point we have information about the true preferences captured in the responses `y.sim`, given to 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 $-H^{-1}$, with 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. Observed 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 by adjusting the `m` argument, where the number of draws equals 2^m . For this example, the number of samples drawn equals $2^6 = 64$.

```
R> set.seed(123)
R> draws <- ImpsampMNL(prior.mean = m, prior.covar = v,
+   des = init.des, n.alts = 2, y = y.sim, m = 6)
R> draws
```

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`.

```
$sample
      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[1,] 0.815710679 0.29945996 2.3279126 0.01850146 -0.914443423 0.27768450
[2,] 0.514464399 0.44535057 0.8742994 0.84613723 0.550771381 2.10385817
[3,] 1.419726482 1.16875137 1.8508186 0.74272714 -0.306573715 1.09351844
...
[64,] 1.011703004 0.16938730 1.6699865 -0.95228248 0.194338610 1.20516300

$weights
[1] 8.000760e-03 3.580512e-02 1.699195e-02 6.916065e-03 6.127430e-03 3.241708e-06
[7] 1.001706e-02 3.414615e-02 3.778920e-03 2.029903e-02 1.118700e-02 1.332692e-02
...
[61] 5.256763e-03 7.505885e-04 1.753621e-02 1.273443e-02

$max
[1] 0.6637305 0.3730624 1.5945580 0.4360475 -0.1752358 1.1989975

$covar
      Var11      Var12      Var13      Var21      Var22      Var31
Var11 0.661671614 -0.017061942 0.007319621 -0.07919544 0.02191193 0.03868335
Var12 -0.017061942 0.547240782 0.002543944 0.04453052 -0.05330017 -0.01675242
Var13 0.007319621 0.002543944 0.615892582 -0.02388446 -0.12888913 0.14332186
Var21 -0.079195439 0.044530523 -0.023884463 0.44363964 -0.04912680 -0.01186809
Var22 0.021911930 -0.053300169 -0.128889133 -0.04912680 0.47689172 -0.07259163
Var31 0.038683346 -0.016752417 0.143321862 -0.01186809 -0.07259163 0.45608684
```

Given the draws from the posterior distribution we can now select the next optimal choice set. This can either be done by **minimizing the DB-error or by maximizing the KL criterion**.

SeqDB

The SeqDB function can be used to select the next optimal choice set based on the DB-error. **The algorithm will evaluate each possible future choice set in combination with the previously selected choice sets and select the one which minimizes the DB-error, given draws from the posterior preference distribution.** The previously selected choice sets need to be specified in `des`, in this example the initial design `init.des`. 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 `dr` are specified in `par.draws` and their importance weights `w` in `weights`. Our prior covariance matrix `v` is passed through `prior.covar`.

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

The output contains the most DB-optimal choice set, and the associated DB-error.

```
$set
      Var11 Var12 Var13 Var21 Var22 Var31
[1,]      1      0      0     -1     -1      1
[2,]      0      0      1      1      0     -1

$db.error
[1] 0.00808191
```

SeqKL

The next optimal choice set can also be selected with the `SeqKL` function, in which case the KL divergence criterion is maximized. Note that here the design matrix is not required, since the KL criterion only depends on the expected change in the posterior distribution that a particular choice set would induce. Since not a single determinant is computed, this approach is way faster in comparison with the `SeqDB` function. In the example below the same candidate set, draws and importance weights as in the `SeqDB` example are used. There are no alternative specific constants, as specified in `alt.cte`.

```
R> set <- SeqKL(cand.set = cs, n.alts = 2, alt.cte = c(0, 0),
+   par.draws = dr, weights = w)
R> set
```

The output contains the most optimal choice set based on the KL criterion and the associated KL-divergence.

```
$set
      Var11 Var12 Var13 Var21 Var22 Var31
[1,]     -1     -1     -1      1      0      1
[2,]      1      0      0     -1     -1     -1

$kl
[1] 0.801879
```

At this point the `RespondMNL` function can be used again to simulate a choice for the newly selected choice set, and all subsequent 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 then vary the heterogeneity of this population distribution along with the number of adaptive choice sets and other specifications in each simulation study. By inspecting the choice sets and looking at the recovery of the true population parameters, one can decide which design approach to use. The simulated choice data can be stored and prepared for estimation using existing R packages for which we refer to Section 5.

4. Real surveys with `idefix`

Once the appropriate design approach is chosen, which can be based on simulations as described in Section 3.3, the next step could be to collect empirical data. In order to use the IASB approach in

real DCE's, one should be able to do calculations in between the choices of respondents, because after each observed choice, the posterior should be updated and the next choice set should be selected. Therefore a reactive environment is necessary. In R, a **shiny** application can be used to create such a reactive webenvironment (Chang, Cheng, Allaire, Xie, and McPherson 2017). The package **idefix** incorporates a function that launches such a **shiny** application.

With the `SurveyApp` function it is possible to conduct discrete choice experiments and collect the choice data by presenting choice sets on screen. The `SurveyApp` function can be used to present pregenerated designs without any adaptive choice sets. It can also be used to include additional sequential adaptive choice sets by making use of the IASB methodology. Whenever sequential adaptive sets are required, the `SurveyApp` function will generate choice sets in a similar way as described in Section 3.3. Lastly one can choose not to start with a pregenerated design and to generate all choice sets using the IASB approach. In what follows, an example of each scenario is given. Afterwards it is explained how a **shiny** script can be deployed online, such that the application, in this case a survey, can be consulted through a weblink.

4.1. 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 user's responses.

The choice design should be specified in `des`, this takes the same format as for example the output of `Modfed` in Section 2.3, namely a matrix in which each row is a profile and choice sets consist of several subsequent rows. For this example we use the example choice design `example_design`, which is included in the **idefix** package. 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")
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 `n.total`. If no other choice sets, besides the one specified in `des`, are required 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 second example. In `alts` the names of the alternatives have to be specified. In this case there are two alternatives, named Alternative A and Alternative B. In `atts`, the names of the attributes are specified, here price, time, and comfort.

```
R> n.sets <- 8
R> alternatives <- c("Alternative A", "Alternative 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. Here the levels for price are 10, 5, and 1 dollar. The attribute time can take values 20, 12 and 3 minutes. Lastly, the comfort attribute can vary between bad, average, and good.

```
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 2.3. 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 3. Before the discrete choice task starts, some instructions can be given, which 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 3.

```
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)
```



choice set: 3 / 8

	Alternative A	Alternative B
Price	\$1	\$10
Time	20 min	3 min
Comfort	good	bad

Please choose the alternative you prefer

☐ Alternative A ☐ Alternative B

Figure 3: Example of a **shiny** survey.

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 directory is specified, two text files will be written to that directory at the end of the survey. One containing the uncoded design as it was presented to the participant together with the sequence of alternatives that were chosen. 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 5 for more information). The file containing the uncoded 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 4.

file: 1513325455_char_data.txt					file: 1513325455_num_data.txt								
	set	Alternative A		Alternative B	resp		par.1	par.2	par.3	par.4	par.5	par.6	resp
Price	1	\$5	\$1	Alternative A		set1.alt1	1	0	1	0	0	0	0
Time	1	12 min	20 min	Alternative A		set1.alt2	0	1	0	0	1	0	1
Comfort	1	bad	average	Alternative A		set2.alt1	0	0	0	0	1	0	1
Price	2	\$10	\$5	Alternative B		set2.alt2	1	0	1	0	0	0	0
Time	2	20 min	12 min	Alternative B		set3.alt1	0	1	0	0	0	1	0
Comfort	2	average	bad	Alternative B		set3.alt2	0	0	0	1	0	0	1
Price	3	\$1	\$10	Alternative A		set4.alt1	0	1	0	0	0	0	1
Time	3	20 min	3 min	Alternative A		set4.alt2	1	0	1	0	1	0	0
Comfort	3	good	bad	Alternative A		set5.alt1	0	0	0	1	0	1	0
Price	4	\$1	\$5	Alternative B		set5.alt2	0	1	0	1	0	0	1
Time	4	20 min	12 min	Alternative B		set6.alt1	1	0	0	1	1	0	1
Comfort	4	bad	average	Alternative B		set6.alt2	0	1	1	0	0	0	0
Price	5	\$10	\$1	Alternative B		set7.alt1	0	0	1	0	0	1	0
Time	5	3 min	3 min	Alternative B		set7.alt2	1	0	0	0	1	0	1
Comfort	5	good	bad	Alternative B		set8.alt1	0	1	1	0	1	0	1
Price	6	\$5	\$1	Alternative B		set8.alt2	1	0	0	1	0	1	0
Time	6	3 min	12 min	Alternative B									
Comfort	6	average	bad	Alternative B									
Price	7	\$10	\$5	Alternative B									
Time	7	12 min	20 min	Alternative B									
Comfort	7	good	average	Alternative B									
Price	8	\$1	\$5	Alternative B									
Time	8	12 min	3 min	Alternative B									
Comfort	8	average	good	Alternative B									

Figure 4: Example of two data files.

4.2. Discrete choice experiment containing adaptive sets.

The `SurveyApp` function can also be used to generate sequential adaptive choice sets as explained in Section 3. 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 first, afterwards an example of the latter is given.

With initial design

The difference with the previous example, in which there were no adaptive sets, is that `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 IASB methodology. In order to do so, some additional arguments need to be specified. All arguments before `crit` are specified exactly the same as in the previous example except the number of sets is now twelve instead of eight, indicating we want 4 additional adaptive sets.

```
R> n.sets <- 12
R> p.mean <- c(0.3, 0.7, 0.3, 0.7, 0.3, 0.7)
R> p.var <- diag(length(p.mean))
```

In order to generate the adaptive sets we must specify whether we want to use the DB of KL criterion as explained in Section 3. The `crit` argument can either be "KL" to make use of the `SeqKL` function

or "DB" to make use of the `SeqDB` function. Here the Kullback Leibler criterion is chosen. The prior is a normal distribution for which we can specify the mean vector and covariance matrix. In this example the mean vector `p.mean` consist 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.

```
R> levels <- c(3, 3, 3)
R> code <- c("D", "D", "D")
R> cand <- Profiles(lvls = levels, coding = code)
```

In order to select the most efficient choice set based on the posterior probability of the preference parameters the candidate design needs to be given. Again we use the `Profiles` function as explained in Section 2.3 to generate all possible profiles with 3 attributes, each with 3 levels and all dummy coded. The characteristics of the initial design should be the same.

```
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 = dataDir, crit= "KL",
+   prior.mean = p.mean, prior.covar = p.var,
+   cand.set = cand, m = 6)
```

Lastly `m = 6` indicates the number of draws we want to use in the importance sampling procedure, needed to sample from the posterior distribution. The number of draws equals 2^m . 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 as can be seen in Figure 5.

file: 1513869174_num_data.txt

	par.1	par.2	par.3	par.4	par.5	par.6	resp
set1.alt1	0	1	0	1	1	0	1
set1.alt2	1	0	1	0	0	1	0
set2.alt1	0	1	1	0	1	0	1
set2.alt2	1	0	0	1	0	1	0
set3.alt1	1	0	1	0	1	0	0
set3.alt2	0	1	0	1	0	1	1
set4.alt1	1	0	0	1	1	0	0
set4.alt2	0	1	1	0	0	1	1
set5.alt1	1	0	0	1	1	0	1
set5.alt2	0	0	1	0	0	1	0
set6.alt1	0	0	0	1	0	0	0
set6.alt2	1	0	1	0	1	0	1
set7.alt1	0	0	1	0	1	0	0
set7.alt2	1	0	0	1	0	1	1
set8.alt1	0	0	0	1	1	0	0
set8.alt2	1	0	1	0	0	1	1
set9.alt1	1	0	1	0	0	0	1
set9.alt2	0	0	0	1	1	0	0
set10.alt1	0	1	0	0	0	0	1
set10.alt2	1	0	1	0	1	0	0
set11.alt1	0	1	1	0	1	0	1
set11.alt2	1	0	0	1	0	1	0
set12.alt1	0	0	1	0	0	0	0
set12.alt2	1	0	0	0	1	0	1

Figure 5: Example of the numeric data file with four additional adaptive sets.

Without initial design

The use of an initial design is not required when the Kullback Leibler criterion is used to select efficient choice sets. In this example of the `SurveyApp` function, all arguments are the same as in the previous example except for the fact that no initial design is specified (`des = NULL`). The data files saved in `data.dir` are the same as in the previous examples.



```
R> SurveyApp (des = NULL, n.total = n.sets, alts = alternatives,
+   atts = attributes, lvl.names = labels, coding = code,
+   buttons.text = b.text, intro.text = i.text,
+   end.text = e.text, data.dir = dataDir, crit = "KL",
+   prior.mean = p.mean, prior.covar = p.var,
+   cand.set = cand, m = 6)
```

4.3. Online surveys

The previously described scripts 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 the responses. This procedure happens however local on the computer where the R script is being evoked. What would be more interesting is to gather the data through an online survey. A convenient way of doing this is to deploy the R script, containing code to generate a **shiny** application

as explained in Section 4.1 and 4.2, to [Shinyapps.io](https://shinyapps.io), a free server made available by RStudio to host **shiny** applications. The first step in doing this is to make sure all the settings are modified as desired and the properties of the survey satisfy the researcher. Afterwards one can simply place `runApp()` around the `SurveyApp` function as shown in the example below.

```
R> runApp(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 = tempdir()))
```

R will now recognize the whole script as a **shiny** script. The application can now be seen by clicking the "run app" button that will be visible in the top right corner of the script. To deploy the application online one can follow the clear explanation on <https://shiny.rstudio.com/articles/shinyapps.html>. After the application is deployed you will receive a weblink where the survey can be found. This way data can be gathered by providing the weblink to potential participants. The participants will only see the user interface, thereby making it possible to present sequentially adaptive choice sets without them noticing anything of the R script running in the background.

The only downside of this approach, for the time being, is that RStudio has not yet implemented persistent data storage on shinyapps.io, however they are planning to do so soon. 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 2017), or to Amazon with the **aws.s3** package (Leeper 2017) from that temporary directory. More information about remotely store data while using shinyapps.io can be found on <https://shiny.rstudio.com/articles/persistent-data-storage.html>.

5. Data management

5.1. Load data

The datafiles stored by `SurveyApp` in `data.dir` can be easily loaded back into R with the function `LoadData`. As explained in Section 4, 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 specify `type = "char"`. In this case all files containing "num" in their filename will be loaded and stacked above each other. Each file, which represents a 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 set1.alt1 1 0 1 0 0 0 1
1 set1.alt2 0 1 0 0 1 0 0
1 set2.alt1 0 0 0 0 1 0 0
1 set2.alt2 1 0 1 0 0 0 1
1 set3.alt1 0 1 0 0 0 1 1
1 set3.alt2 0 0 0 1 0 0 0
1 set4.alt1 0 1 0 0 0 0 0
1 set4.alt2 1 0 1 0 1 0 1
1 set5.alt1 0 0 0 1 0 1 1
1 set5.alt2 0 1 0 1 0 0 0
1 set6.alt1 1 0 0 1 1 0 0
1 set6.alt2 0 1 1 0 0 0 1
1 set7.alt1 0 0 1 0 0 1 1
1 set7.alt2 1 0 0 0 1 0 0
1 set8.alt1 0 1 1 0 1 0 0
1 set8.alt2 1 0 0 1 0 1 1
2 set1.alt1 1 0 1 0 0 0 0
2 set1.alt2 0 1 0 0 1 0 1
2 set2.alt1 0 0 0 0 1 0 0
2 set2.alt2 1 0 1 0 0 0 1
...
7 set6.alt1 1 0 0 1 1 0 0
7 set6.alt2 0 1 1 0 0 0 1
7 set7.alt1 0 0 1 0 0 1 0
7 set7.alt2 1 0 0 0 1 0 1
7 set8.alt1 0 1 1 0 1 0 1
7 set8.alt2 1 0 0 1 0 1 0

```

5.2. Data transformation

Datatrans

The data, simulated or gathered with **idefix**, consists of a design matrix in which each row is an alternative. Several subsequent rows form choice sets, see for example the output of `Modfed` in Section 2.3. Individual designs can be combined in the same manner into an aggregate design. 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 dataset as shown at the end of Section 5.1. 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. Thus after we load in the dataset, 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 `X`.

```

R> data("aggregate_design")
R> X <- aggregate_design[, -c(1, 2, 9)]
R> 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
17	1	0	1	0	0	0
18	0	1	0	0	1	0
19	0	0	0	0	1	0
...						
112	1	0	0	1	0	1

The responses will be captured in a matrix in which each column vector represents the binary choice data of a unique respondent. Since there were seven respondents who each responded to eight choice sets containing two alternatives, the dimension of our respons matrix is then as follows: $Y(16 \times 7)$.

```
R> Y <- matrix(aggregate_design$resp, ncol = 7, byrow = FALSE)
R> Y
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]
[1,]	1	0	0	1	1	1	0
[2,]	0	1	1	0	0	0	1
[3,]	0	0	1	0	0	0	0
[4,]	1	1	0	1	1	1	1
[5,]	1	0	0	1	1	1	1
[6,]	0	1	1	0	0	0	0
[7,]	0	0	1	0	0	0	1
[8,]	1	1	0	1	1	1	0
[9,]	1	0	0	1	0	1	0
[10,]	0	1	1	0	1	0	1
[11,]	0	0	1	1	0	0	0
[12,]	1	1	0	0	1	1	1
[13,]	1	1	0	1	0	1	0
[14,]	0	0	1	0	1	0	1
[15,]	0	0	1	1	0	1	1
[16,]	1	1	0	0	1	0	0

The function `Datatrans` can be used to transform the data into the correct format in order to use existing estimation packages in R.

The `pkg` argument can take four values: 1 for the `rhierMnlRwMixture` function of package **bayesm** (Rossi 2015), 2 for the `choicemodelr` function of package **ChoiceModelR** (Sermas 2012), 3 for the `doHB` function of the **RSGHB** (Dumont and Keller 2015) package, and 4 for the

`rbprobitGibbs` function of the **bayesm** 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 matrix `y` is not yet in binary format, but in discrete format (one number per choice set, indicating the chosen alternative), the `bin` argument has to be set to `FALSE`. The `no.choice` option can be set to `TRUE` if the choice data contains responses in which no alternative was chosen. If the response data is already binary, the `no.choice` argument is neglected and a no choice should be represented by a zero for each alternative.

```
R> Datatrans(pkg = 1, des = X, y = Y, n.alts = 2, n.sets = 8,
+           n.resp = 7, n.par = 6, no.choice = FALSE, bin = TRUE)
```

In the example above, the data is transformed in order to use the `rhierMnlRwMixture` function of package **bayesm**. The dataformat required is a list with elements `$lgtdata`, and `$p` where `$lgtdata` is a list of `nlgt = length(lgtdata)` lists with each cross-section unit MNL data.

`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 **RSGHB** package for estimation. The required format is a data frame with one row per choice task. The columns variables are: an ID identifier `ID`, the choice set number `Choice Set`, attribute one for alternative one `V3`, attribute one for the second alternative `V4`, ... , attribute three for the first alternative `V7`, attribute three for the second alternative `V8`, and the response variable `Choice`.

```
R> Datatrans(pkg = 3, des = X, y = Y, n.alts = 2, n.sets = 8,
+           n.resp = 7, n.par = 6, no.choice = FALSE, bin = TRUE)
```

	ID	Choice	Set	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	Choice
1	1		1	1	0	0	1	1	0	0	0	0	1	0	0	1
2	1		2	0	1	0	0	0	1	0	0	1	0	0	0	2
3	1		3	0	0	1	0	0	0	0	1	0	0	1	0	1
4	1		4	0	1	1	0	0	1	0	0	0	1	0	0	2
5	1		5	0	0	0	1	0	0	1	1	0	0	1	0	1
6	1		6	1	0	0	1	0	1	1	0	1	0	0	0	2
7	1		7	0	1	0	0	1	0	0	0	0	1	1	0	1
8	1		8	0	1	1	0	1	0	0	1	1	0	0	1	2
9	2		1	1	0	0	1	1	0	0	0	0	1	0	0	2
10	2		2	0	1	0	0	0	1	0	0	1	0	0	0	2
...																
56	7		8	0	1	1	0	1	0	0	1	1	0	0	1	1

6. Conclusion

With the R package **idefix** it is possible to generate optimal designs for discrete choice experiments taking into account the non-linearity of the model. The designs can be optimized locally or in a Bayesian way. In addition, the IASB approach is implemented, in which choice sets can be generated sequentially, based on the updated prior distribution of the parameters of a respondent. This can be done by minimizing the D(B)-error or by maximizing the KL-criterion. An example is given how to set up a simulation study by using the functions of the **idefix** package. It is shown how the `SurveyApp` function can be used to launch a **shiny** application, in order to gather empirical data. Lastly a function is provided to aid users in transforming their data into different formats, such that estimation can be done easily without leaving the R environment.

References

- 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 (2010). “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 MC, 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.

- Bridges JF, *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 J, Xie Y, McPherson J (2017). *shiny: Web Application Framework for R*. R package version 1.0.3, 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. URL <http://www.jstor.org/stable/1268315>.
- 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.
- 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 (2015). *RSGHB: Functions for Hierarchical Bayesian Estimation: A Flexible Approach*. R package version 1.1.2, URL <https://CRAN.R-project.org/package=RSGHB>.
- Fedorov VV (1972). *Theory of Optimal Experiments*. Probability and mathematical statistics 12. Academic press, New York (N.Y.).
- Harman R, Filova L (2016). *OptimalDesign: Algorithms for D-, A-, and IV-Optimal Designs*. R package version 0.2, 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 (2014). *choiceDes: Design Functions for Choice Studies*. R package version 0.9-1, 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. URL <http://www.jstor.org/stable/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. URL <http://www.jstor.org/stable/30162415>.

- Kullback S, Leibler RA (1951). “On Information and Sufficiency.” *The Annals of Mathematical Statistics*, **22**(1), 79–86. URL <http://www.jstor.org/stable/2236703>.
- Leeper TJ (2017). *aws.s3: AWS S3 Client Package*. R package version 0.3.3.
- Marschak J (1950). “Rational Behavior, Uncertain Prospects, and Measurable Utility.” *Econometrica*, **18**(2), 111–141. doi:[10.2307/1907264](https://doi.org/10.2307/1907264).
- McFadden D (1974). “Conditional Logit Analysis of Qualitative Choice Behavior.” *Frontiers in Econometrics*, pp. 105–142.
- McFadden D, Train K (2000). “Mixed MNL Models for Discrete Response.” *Journal of Applied Econometrics*, **15**(5), 447–470. URL <http://www.jstor.org/stable/2678603>.
- Ram K, Yochum C (2017). *rdrop2: Programmatic Interface to the 'Dropbox' API*. R package version 0.8.1, URL <https://CRAN.R-project.org/package=rdrop2>.
- R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL <http://www.R-project.org>.
- Rose JM, Bliemer MC (2014). *Stated Choice Experimental Design Theory: the Who, the What and the Why*, chapter 7, pp. 152–177. Cheltenham, UK.
- Rossi P (2015). *bayesm: Bayesian Inference for Marketing/Micro-Econometrics*. R package version 3.0-2, URL <https://CRAN.R-project.org/package=bayesm>.
- Sermas R (2012). *ChoiceModelR: Choice Modeling in R*. R package version 1.2, URL <https://CRAN.R-project.org/package=ChoiceModelR>.
- Train KE (2003). *Discrete Choice Methods with Simulation*. Cambridge University Press. doi:[10.1017/CBO9780511753930](https://doi.org/10.1017/CBO9780511753930).
- Wheeler B (2014). *AlgDesign: Algorithmic Experimental Design*. R package version 1.1-7.3, 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](https://doi.org/10.1016/j.ijresmar.2011.06.002).

Affiliation:

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

Frits Traets
Faculty of Economics and Business research
KULeuven
Naamsestraat 69 bus 3500
3000 Leuven, Belgium
E-mail: frits.traets@kuleuven.be