KU LEUVEN

FACULTY OF SCIENCE

Extending the Idefix package

Daniel Gerardo GIL SANCHEZ

Supervisor: Prof. Martina Vandebroek
Mentor: *Frits Traets*

Thesis presented in
fulfillment of the requirements
for the degree of Master of Science
in Statistics

Academic year 2018-2019

Preface

Discrete choice models are widely used in applied sciences such as transportation, marketing, medicine, among others. These kind of models are characterized by a discrete choice response that can be nominal or ordinal. To collect information of this nature, Discrete Choice Experiments are usually conducted because it offers a framework to get reliable conclusions about the problem at hand.

The *idefix* package, implemented in R, generates D-optimal (or D_B-optimal) designs for discrete choice experiments based on the Multinomial Logit model under the assumption of homogeneity in the respondents, and invidually adaptive designs based on the Mixed Multinomial Logit Model, when the homogeneity assumption does not hold.

The objective of this thesis is to improve the routine that has already been developed in the package in terms of speed and usability, and implement other algorithms or criteria to create a wider range of optimal designs.

I would like to thank my supervisor Prof. Martina Vandebroek for supporting me throughout the whole process with good energy and patience. My sincere thanks also goes to Frits Traets, developer of the package and daily mentor, for helping me to overcome those little problems that appeared once in a while.

I would also like to thank my parents for teaching me the value of education, and supporting me in any adventure I could imagine. Finally, I want to thank Juliana for accompanying me in this process and giving me strength to continue until the end.

Summary

As in experimental design for linear models, in Discrete Choice Experiments the number of treatments explodes as the number of attributes and levels increase. As a consequence, the use of optimal designs has increased because it allows researchers to get reliable conclusions without the use of factorial designs. To get an optimal design, different optimality criteria can be chosen depending on the goal of the problem that is being tackled. Since the focus of the majority of applications is to obtain precise estimates from a specified model, the D-optimality criterion is commonly used.

The *idefix* package, implemented in `R`, generates D-optimal (or D_B-optimal) designs for discrete choice experiments based on the Multinomial Logit model under the assumption of homogeneity in the respondents, and individually adaptive designs based on the Mixed Multinomial Logit Model, when the homogeneity assumption does not hold. To generate these optimal designs, the package uses a computationally expensive algorithm called Modified Fedorov.

Since the implementation of the package makes difficult its application in real situations, with a large number of attributes or levels, the goal of this thesis is to improve the routine that has already been developed in the package in terms of speed and usability, and implement other algorithms or criteria to create a wider range of optimal designs.

To improve computational speed, unnecessary loops in the code were avoided and some functions were implemented in `C++`. As a result, optimal designs are now obtained from 9 to 10 times faster than before. In terms of usability, alternative specific constants and "none of the above" options were implemented in the functions that did not have these options available.

The Coordinate Exchange algorithm was also implemented as an alternative of the Modified Fedorov algorithm. A comparison between both implementations, conducted through simulations, showed that the Coordinate Exchange algorithm produced designs as efficient as the Modified Fedorov in significantly less time.

The implementation of Kullback-Leibler criterion was modified and updated in order to give similar results as any other function in the package. A comparison between this criterion and the D_B-optimality criterion showed that this implementation works well when a semi-informative prior is used. More simulations are required to determine under which scenarios this criterion outperforms the D_B-optimality criterion.

List of Figures

List of Tables

List of Abbreviations

CEA	Coordinate Exchange Algorithm
CPU	Central Processing Unit
CRAN	The Comprehensive R Archive Network
DCE	Discrete Choice Experiment
i.i.d	Independent and identically distributed
KL	Kullback-Leibler distance
KLP	Kullback-Leibler criterion
MB	Megabytes
MNL	Multinomial Logit model
MIXL	Mixed Multinomial Logit model
OLS	Ordinary Least Squares

List of Symbols

U_j	Utility function of decision maker j
$\boldsymbol{X}$	Design matrix
$\boldsymbol{Y}$	Response vector
x_j	Alternative $j-$th in $\boldsymbol{X}$
$\boldsymbol{\beta}$	Vector of parameters
ϵ_j	Error term
p_{js}	Probability of $j-$th alternative in choice set s
L_{js}	Probability of $j-$th alternative in choice set s
p_s	Vector of probabilities in choice set s
$\boldsymbol{P_s}$	Diagonal matrix of p_s
LL	Log-Likelihood
M	Information Matrix
f	Probability distribution
$\pi(\boldsymbol{\beta})$	Prior distribution of $\boldsymbol{\beta}$
D_B	Bayesian D-optimal criterion
det	Determinant

Contents

Chapter 1

Introduction

Discrete choice models are widely used in applied sciences such as transportation, marketing, medicine, among others. These kinds of models are characterized by a discrete choice response that can be nominal or ordinal. To collect information of this nature, fundamental research has focused on the development of the theory of discrete choice experiments, where the respondents are asked to select an alternative, composed by different combinations of attribute levels, from a series of choice sets.

As in experimental design for linear models, in Discrete Choice Experiments (DCE) the number of possible profiles or treatments explodes as the number of attributes and levels increase. Optimal experimental designs, also known as efficient designs, are widely used in several scenarios because it allows to get reliable conclusions without the use of factorial designs. Consequently, there are different optimality criteria to choose a final design, such as D, A, G and V optimal, each using different methodologies (see Kessels et al., 2006). Since the focus of the majority of applications is to obtain precise estimates from a specified model, the D-optimality criterion is commonly used.

In regard to software implementation, there were no available packages to generate discrete choice designs in R. What people used to do, and still do, is to generate an optimal design for a linear model, use it to collect the data and then analyze it with a discrete choice model. This approach can be improved in the sense that instead of using the information matrix of a linear model (usually OLS) to maximize its determinant, the information matrix of a nonlinear discrete choice model can be used, such as the Multinomial Logit (MNL).

The *idefix* package, already available in CRAN, generates D-optimal (or D_B-optimal) designs for discrete choice experiments based on the Multinomial Logit model assuming homogeneity in the respondents and implements individually adapted designs for the Mixed Multinomial model when the assumption of homogeneity does not hold. The current state of this package generates optimal designs using the Modified Fedorov Algorithm and sequential Modified Fedorov Algorithm for the Multinomial Logit and Mixed Multinomial Logit model, respectively. This algorithm is computationally expensive and makes difficult its application in real situations with a large number of attributes or levels. For this reason, the purpose of this thesis is to improve the routine that has already been developed in the package and implement other algorithms or criteria to

create optimal designs, and make a simulation study to compare results from different approaches.

In the following, the process developed to finish this project is presented. Chapter 2 contains formal definitions, explains the algorithms that are used in the package and describes the current state of the package. In Chapter 3, the specific objectives of this thesis are presented. Chapter 4 describes the process to optimize the computing time in the implementation of the Modified Fedorov algorithm. In Chapter 5, the steps taken to validate the implementation of the Kullback-Leibler criterion are presented. Chapter 6 shows how the Coordinate Exchange algorithm is implemented in the package. In Chapter 7, a simulation study is presented with the purpose of describing the efficiency of two different optimality criteria in the search for an optimal design, and a comparison between two different algorithms is shown. Chapter 8 describes the modifications done on the package to improve its readability and maintenance. Finally, Chapter 9 concludes the work done and present suggestions for future work in the package.

Chapter 2

Background

2.1 Discrete Choice Experiments

In a practical approach, a discrete choice experiment can be described as a series of questions presented to different respondents with the goal of understanding the decision making process conducted by them, in the selection of a specific response. This questionnaire is composed of different *choice sets* that present *alternatives* or *profiles* from which the respondents have to choose. Each *alternative* is a combination of *levels* from different *attributes*, also called *factors*. Note that the respondent is not always a person but can be households, firms or any other decision unit.

To understand how the decisions are made, a probabilistic model is usually defined before the experiment is conducted, which under different assumptions may give clues about what is important in the final selection of a response. The following two sections give a brief presentation of two different kinds of models and the remainder two discuss some considerations about DCEs. Then, the current implementation of the package is discussed.

2.1.1 Multinomial Logit Model

This model is derived from the random utility model, where it is assumed that a respondent makes its final choice by measuring the utility of each of the alternatives presented in a choice set (see Train, 2009, chap. 3). This utility is computed as a linear combination of the attributes that are present in the alternative and can be expressed as:

$$U_j = \mathbf{x}_j^\top \boldsymbol{\beta} + \epsilon_j \tag{2.1}$$

Where j represents the different alternatives shown within a choice set, $\mathbf{x}_j$ is the corresponding row of the j-th alternative/profile in the design matrix $\mathbf{X}$, β denotes the importance of each attribute level and ϵ_j is an i.i.d extreme value error term.

Now, the Multinomial Logit probability that a respondent chooses the j-th profile in the

s-th choice set is defined as:

$$p_{js}(\boldsymbol{\beta}) = \frac{\exp(\mathbf{x}_{js}^{\top}\boldsymbol{\beta})}{\sum_{i=1}^{J}\exp(\mathbf{x}_{is}^{\top}\boldsymbol{\beta})} \tag{2.2}$$

Because the errors are assumed to be independent, the choices of N respondents represent independent draws from a Multinomial distribution. Therefore, the log-likelihood can be written as:

$$LL(\mathbf{Y}|\mathbf{X},\boldsymbol{\beta}) = \sum_{s=1}^{S}\sum_{j=1}^{J}\sum_{n=1}^{N} y_{jsn}\ln(p_{js}(\boldsymbol{\beta})) \tag{2.3}$$

Where $\mathbf{Y}$ is the vector of choices from N respondents with elements y_{jsn}.

The correspondent Fisher information matrix is of vital importance in the context of optimal designs, because most of the optimality criteria depend on this expression:

$$\mathbf{M}(\mathbf{X},\boldsymbol{\beta}) = N\sum_{s=1}^{S}\mathbf{X}_s^{\top}(\mathbf{P}_s - \mathbf{p}_s\mathbf{p}_s^{\top})\mathbf{X}_s \tag{2.4}$$

Where $\mathbf{p}_s = [p_{1s},\ldots,p_{js}]$ and $\mathbf{P}_s = diag[p_{1s},\ldots,p_{js}]$. It is important to notice that the information matrix depends on the unknown parameters through the probabilities, so initial parameter values are required before computing this matrix. Kessels et al. (2006) show that the Bayesian design approach works well in this scenario, specifying a prior normal distribution for β, $N(\beta|\beta_0,\Sigma_0)$.

2.1.2 Mixed Multinomial Logit Model

According to Train (2009), this kind of models obviates the tree limitations that a standard logit model has: it allows for random variation between respondents, unrestricted substitution patterns and correlation in unobserved factors over time.

The Mixed Logit probabilities are defined as the integrals of standard Logit probabilities with respect to the density of the parameters. These probabilities can be expressed as:

$$P_{js}(\boldsymbol{\beta}) = \int L_{js}(\boldsymbol{\beta})f(\boldsymbol{\beta})d\boldsymbol{\beta} \tag{2.5}$$

Where L_{js} is the logit probability, as defined in 2.2, evaluated at $\boldsymbol{\beta}$ and $f(\boldsymbol{\beta})$ is the density function of the parameters. In other words, the Mixed Logit probability is a weighted average of the Multinomial Logit probabilities with weights given by the distribution of $\boldsymbol{\beta}$.

2.1.3 Individually Adaptive Designs

This approach, proposed by Yu et al. (2011), is useful when the assumption of homogeneity between respondents does not hold, i.e., when only Mixed Multinomial Logit

(MIXL) model can be used to analyze the data. It considers that when the respondents are heterogeneous, each of them has its own preference structure that behaves according to a Multinomial Logit Model. In this sense, the MIXL model can be seen as a combination of different Multinomial Logit models, one for each respondent. As a consequence, the Individually Adaptive Sequential Bayesian approach generates an individual optimal design for each of the respondents involved in the study, instead of applying the same design for everyone.

According to Yu et al. (2011), this procedure consists of two stages: an initial static stage, where the prior distribution of β is the same for all respondents and it is used to generate an initial design. And the fully adaptive sequential stage, where the prior information for the design is sequentially updated after each choice made by a respondent. As a result, each respondent will have a tailor made design.

2.1.4 Optimality criteria

The idea of an optimal design is to select those choice sets that are most informative on the respondent's choice behavior (Crabbe et al., 2014). The decision of which optimality criteria should be used in the selection of the best design is mainly influenced by the use of the final model. In general, if the purpose of the study is to obtain as precise estimates as possible, then the D- and A- optimality criteria can be used. On the other hand, if the purpose of the study is to develop a model that gives precise response prediction, then G- and V- optimality criteria can be used. Kessels et al. (2006) explain in detail each of these criteria applied on the MNL.

Since the *idefix* package chooses the best design based on the D-optimality criterion and the Kullback-Leibler criterion, a brief explanation of each of them is given in the following.

D-optimality

D-optimality stands for determinant, and the idea behind this criterion is to find an optimal design that minimizes the determinant of the variance-covariance matrix of the parameter estimators or conversely maximizes the determinant of the Information Matrix. As it was mentioned before, this Information Matrix depends on unknown parameters, thus two kinds of designs can be generated: local optimal designs, which are based on a single prior parameter, and Bayesian optimal designs, which involves the specification of a probabilistic distribution on the unknown parameters (Kessels et al., 2006).

As a consequence of the Bayesian approach, a modification in the calculation of the D-optimal criterion has to be done, including the prior distribution of the coefficients. The Bayesian D-optimal criterion, or D_B criterion, is defined as:

$$D_B = \int_{\mathcal{R}^k} \left\{ det\left(\mathbf{M}^{-1}(\mathbf{X}, \boldsymbol{\beta}\right) \right\}^{1/k} \pi(\boldsymbol{\beta}) d\boldsymbol{\beta} \tag{2.6}$$

Where k is the number of unknown parameters in the model, i.e., the number of columns in the design matrix $\mathbf{X}$; and $\pi(\boldsymbol{\beta})$ is the prior distribution of $\boldsymbol{\beta}$, which according to (Kessels et al., 2006) can be a multivariate Normal distribution. From this expression can be seen that the D_B-optimality criterion is the expected value of the D-optimality criterion with respect to the distribution of the coefficients.

In the *idefix* package, these D_B-optimal designs are found using the Modified Fedorov Algorithm, which is explained in the following section.

Kullback-Leibler

In the context of Individually Adaptive designs for each of the respondents in a study (MIXL model), Crabbe et al. (2014) present the Kullback-Leibler criterion to obtain designs that estimate model coefficients as precise as possible. They claim that designs generated with this approach are equally efficient to those generated by the D_B-optimality criterion and at the same time are much faster to calculate. It consists in the Kullback-Leibler distance (KL), which is a measure to compare two density functions. The KL increases when these densities become more divergent. This distance is defined as:

$$KL(f, g) = E_f \left[log \frac{f(x)}{g(x)} \right] \tag{2.7}$$

where f and g are densities of a continuous variable X and E_f is the expected value with respect to f.

Now, the selection of the best next choice set for a specific respondent is based on the maximization of the KL distance between the current posterior of the coefficients and the updated posterior that one will obtain with the additional response information from the next choice (Crabbe et al., 2014). Since in a choice set there are multiple possible alternatives, the expectation over all possible alternatives is maximized. The KL criterion is therefore defined as follows:

$$KLP = \sum_{j=1}^{J} \pi(y_{jsn}|\mathbf{y}_n^{s-1}) KL \left[f(\boldsymbol{\beta})_n|\mathbf{y}_n^{s-1}), f(\boldsymbol{\beta})_n|\mathbf{y}_n^{s-1}, y_{jsn}) \right] \tag{2.8}$$

Where s is the next choice set, n is a particular respondent and j is the chosen alternative. The densities $f(\boldsymbol{\beta})_n|\mathbf{y}_n^{s-1})$ and $f(\boldsymbol{\beta})_n|\mathbf{y}_n^{s-1}, y_{jsn})$ are the updated posteriors and $\pi(y_{jsn}|\mathbf{y}_n^{s-1})$ is the posterior weighted choice probabilities for the alternatives in the choice set s, given the previous responses.

After some calculations, the KL criterion can be rewritten as:

$$KLP = \sum_{j=1}^{J} \pi(y_{jsn}|\mathbf{y}_n^{s-1}) \left[log\ \pi(y_{jsn}|\mathbf{y}_n^{s-1}) - \int log\ p_{jsn}(\boldsymbol{\beta}_n) f(\boldsymbol{\beta}_n|\mathbf{y}_n^{s-1}) d\boldsymbol{\beta}_n \right] \tag{2.9}$$

According to Crabbe et al. (2014), since the KLP only involves the calculation of the posterior weighted choice probabilities, it is much faster than the D-optimal criterion,

where not only the next choice set but also the previous sets in the design are used in the calculation of the information matrix.

In the *idefix* package, this algorithm has already been implemented; however, simulations in R are not consistent with the results presented by Crabbe et al. (2014).

2.1.5 Algorithms

Modified Fedorov Algorithm

Proposed by Cook et al. (1980), it begins with an initial random design, with its corresponding optimality criterion, represented by the blue rectangle in the left in Figure 2.1. The algorithm exchanges an entire row/profile from the initial design matrix with each of the rows from a list of candidate profiles (green rectangle in the top of the figure); usually, this candidate list is just the list of all possible combinations of attribute levels. As a result, there will be different versions of the initial design and in each of them the optimality criterion is computed (represented by blue rectangles in the middle of the figure). Then, the initial design matrix is updated by replacing the existing profile with the profile from the candidate list that improves the optimality criterion (blue rectangle in the right of the figure). This process is repeated for each of the rows/profiles in the initial design and stops when no exchanges are performed.

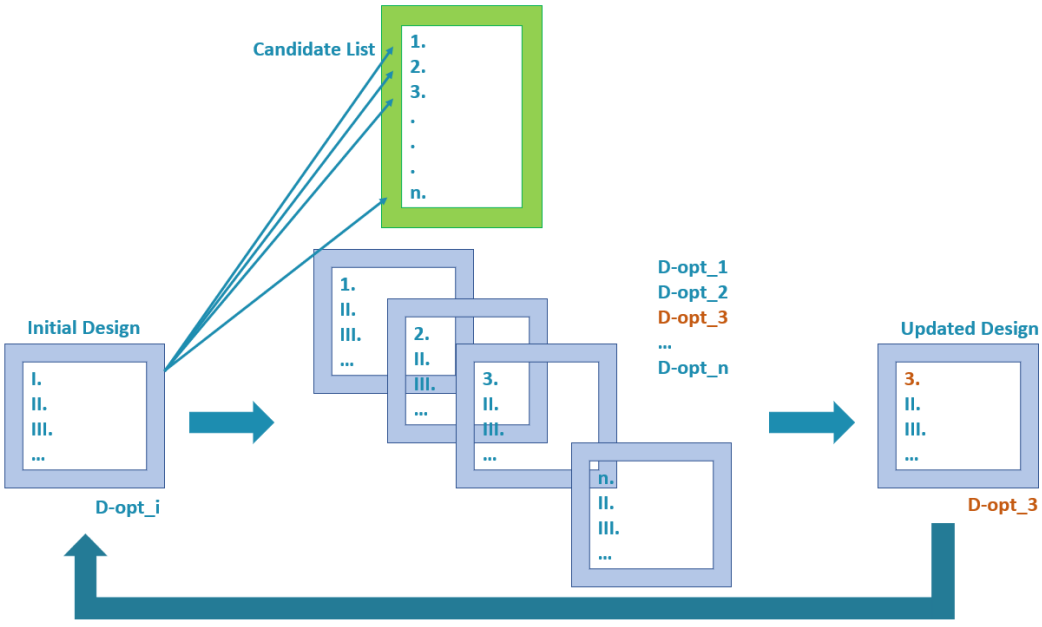


Figure 2.1: Illustration of the Modified Fedorov Algorithm

This algorithm is computationally expensive because in each iteration, the information matrix and its determinant have to be computed for each exchange in the initial design. An easy illustration of how time consuming it can be, suppose that the initial design has 12 profiles, corresponding to 6 choice sets with 2 alternatives each. The candidate

list has, say, 32 profiles. Then, the algorithm has to calculate $12 * 32 = 384$ information matrices and their respective determinants to decide which is the best optimal design.

Furthermore, on top of this number of calculations, remember that the information matrix needs prior values for β. Assuming just 10 draws from the normal prior distribution, then the number of information matrices and determinants to compute will be 3840. And taking into account that the optimal design depends on the random initial design, it is desirable to have as many initial designs as possible to avoid local optima. So let's assume that 100 initial designs are enough. In this sense, the final number of calculations of information matrices and determinants to obtain the best possible optimal design is $3,840,000$.

For this reason, different algorithms, such as the Coordinate Exchange, and optimality criteria, like the KL criterion, that are not as computationally expensive as this one have been proposed.

Coordinate Exchange

Proposed by Meyer and Nachtsheim (1995) as an alternative to profile exchange algorithms, where the whole row of the design matrix is exchanged by a profile from a candidate list (such as the Modified Fedorov algorithm). This algorithm exchanges one attribute at the time instead of the full profile from the design matrix, resulting in coordinate exchanges in the initial design as its name suggests[1].

There are some benefits in the search for an optimal design with this algorithm: First, there is no need to give a list of candidate profiles because it does not use any candidate profiles to improve the initial design. Second, discrete and continuous design spaces are easily handled because for continuous attributes only one coordinate/column is changed and for discrete attributes, changing one level results in a simultaneous change of all coordinates/columns related to the same attribute. Third and maybe the most important, a reduction in computing time of one or two orders of magnitude in large problems is found when compared with profile exchange algorithms (Meyer and Nachtsheim, 1995).

The algorithm begins with a random initial design and its corresponding optimality criterion, such as D_B-optimality or KL criterion. Then, for every profile/row in the initial design, each attribute level is exchanged with all possible levels in the attribute and the initial design is updated by keeping the level that presents the best value of the optimality criterion. Having done this in each element of the design matrix, the process is repeated again until no elements have changed or until the algorithm reaches a prespecified number of iterations. Since the algorithm starts with a random design, it is advisable to have several initial designs to avoid local optima.

Figure 2.2 shows an easy illustration of how this algorithm works. The first step is to

[1]Note that when an attribute is continuous, there is just one coordinate or column in the design matrix. When an attribute is discrete with a levels, there are $a - 1$ coordinates/columns in the design matrix.

create an initial random design, represented by the rectangle on the left of the Figure, and compute its corresponding optimality criterion. In this case, two attributes with three levels each is considered using dummy coding. Then, the first attribute is exchanged with all possible levels and the corresponding optimality is computed. In this example, the second level produces a better optimality criterion and therefore the row is updated (this change is indicated in orange). Next, do the same with the following attribute, which in this case the first level is the one that produces better optimality criterion (also colored in orange). Having updated the first profile, continue with the second profile and do the same. Then continue until convergence.

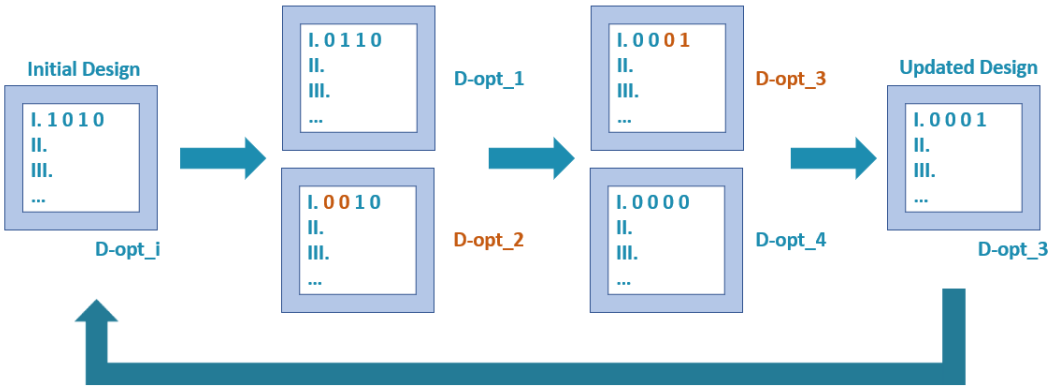


Figure 2.2: Illustration of the Coordinate Exchange Algorithm

2.2 Implementation in R

The best way to show which is the current status of the package and explain the functions that has been developed is through Figure 2.3. It represents a flow chart with the capabilities of the package with respect to the type of design the final user wants.

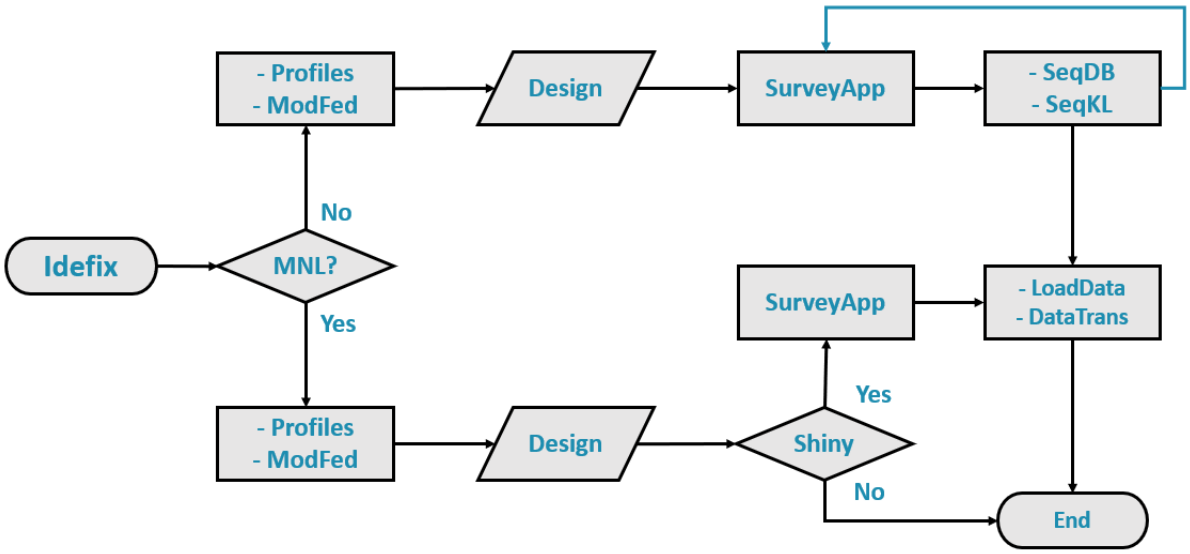


Figure 2.3: Current status of the package

When the user wants to conduct a DCE with this package, the first question that needs to be considered is what kind of model is going to be used in the analysis, represented by the first diamond in the flow chart. First, let's say that a Multinomial Logit Model is desired, so the path in the bottom is followed. At this point there are two functions that work together: `Profiles` and `Modfed`. As it was mentioned above, the Modified Fedorov algorithm depends on a candidate list, which is used to exchange profiles from the initial design. This candidate list can be created with `Profiles` function, where a matrix with all possible combinations of attribute levels is generated. Then, the `Modfed` function is used to generate an optimal design starting from an initial random design or a pre-specified initial design.

Once the design is obtained, represented by the parallelogram in the middle, the user has the option to collect the data by running a `Shiny` application online or by deploying the questionnaire in an external software or even on paper (Chang et al., 2018). If the collection of the data is conducted online via a `Shiny` application, `SurveyApp` can be easily used by setting some specific parameters. The data is then saved as a text file in a specific location and the functions `LoadData` and `DataTrans` can be used to load and shape the data in a certain way that some specific analysis packages can take advantage of them. At this point, the whole flow for a MNL model is finished.

On the other hand, if the model desired is a Mixed Multinomial Model, the path in the top is followed. The first step that needs to be considered in adaptive designs is the initial static design that is going to be the same for all respondents. This can be created by using the same functions mentioned above, `Profiles` and `Modfed`. Once the initial static design is created, `SurveyApp` function runs a `Shiny` application that allows the respondent to answer the corresponding choice sets. When the adaptive design comes to play, this function internally calls `SeqDB` or `SeqKL` function to update the posterior distribution of the coefficients and use it again to show the next choice set. This iterative process is represented by the blue arrow in the top-right corner of the diagram. When the data collection process is complete, the data is exported to a text file in a specific location and the functions `LoadData` and `DataTrans` can be used to load and shape the data in a certain way that some specific packages can take advantage of them, as it was mentioned before.

It is important to mention that even though `SeqKL` appears in the diagram, this function has been temporarily removed from CRAN until further checks are conducted on that routine. As soon as it is ready for production, it will be available again to the public.

Chapter 3

Objectives

Currently, the *idefix* package uses the Modified Fedorov algorithm to create optimal designs for the Multinomial Logit model and Mixed Multinomial Logit model. As it has been shown above, there are different algorithms and optimality criteria that can be implemented in the package to improve the current processing time to generate optimal designs. The specific objectives of this thesis can be listed as follows:

- Optimize the processing time of the Modified Fedorov algorithm by implementing some parts of the algorithm in C++.
- Implement the KL criterion and compare it with the function that is already available in the package.
- Implement the Coordinate Exchange algorithm to create optimal designs.
- Make a simulation study to compare processing times and optimality of designs between the Modified Fedorov algorithm, the Coordinate Exchange algorithm and the use of D_B and KL criteria.
- Reorganize some functions inside the package and remove possible redundancy in the code.

Chapter 4

Optimization of Modified Fedorov Algorithm

The success of this package is related to the possibility of creating a great variety of designs in a reasonable amount of time. As it was mentioned before, the Modified Fedorov algorithm requires a lot of computations, so an efficient implementation is crucial to obtain convergence as quickly as possible.

This is actually more important in cases where individually adaptive designs are implemented in the collection of responses. Recall that in this kind of designs, the information of the previous responses is used to obtain the next choice set, resulting in a tailor made design for each respondent. Therefore, each time a respondent decides what alternative is the best in the current choice set and continues to the next, the algorithm has to update the posterior distribution and find the next choice set by trying all possible combinations of alternatives from the candidate list.

The current implementation already performs the whole process of updating the posterior distribution and selecting the choice set in a reasonable amount of time. To give an example, Figure 4.1 shows a screenshot of how the `SurveyApp` function (`Shiny` application) collects the data interactively. In this case, a $3^3/2/8$ design is applied to each respondent where the first 4 choice sets correspond to the initial static design. At the top of the picture, the current choice set is presented and at the bottom the respondent can select the best alternative. Since this is already the last choice set from the static design, when the respondent clicks on *OK*, the whole algorithm starts working to get the next tailor made choice set. This computing time takes between 2 to 3 seconds to finish, and even though it may seem like it is not much time, respondents can run out of patience easily when the number of choice sets is larger. Additionally, if the experiment is larger in any way, such as more attributes, levels or alternatives per choice set, the computing time increases in exponentially.

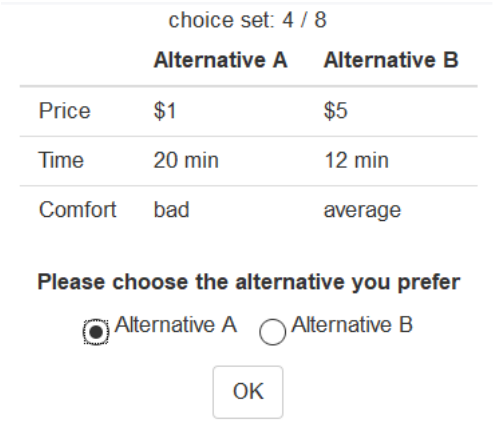


Figure 4.1: Screenshot of `SurveyApp` function

For this reason, it was imperative to improve first the processing time of the `SeqDB` function, which is used for individually adaptive designs, and then the processing time of the `Modfed` function, used for regular static designs. The following sections present with detail this development process.

4.1 Improving processing time in individually adapted designs

There are different ways to improve the execution time of any code in `R`, such as avoiding unnecessary loops with built-in functions, implementing faster functions in `R` language or implementing parts of the code in `C++`. One may think that in order to gain speed, these options should be followed one after the other: First, look for loops in the code and improve them, then try to write a better code and finally, if the processing time is not fast enough, use `C++` to boost the code as a whole. However, efforts can be made, but the code could not be faster.

For this reason, the approach suggested by Wickham (2015a) is followed. This approach consists in:

- Find the slowest part of the code (also known as bottleneck).
- Try to improve it.
- Repeat until the code is fast enough.

To find the slowest part of the function, `profvis` package was used because it leads to identify in an objective manner which parts of the code were the slowest (Chang and Luraschi, 2018). In this respect, it was necessary to specify a case scenario with the purpose of measuring the improvements made. Hence, a $4 \times 3 \times 2/2/8$ adaptive design was chosen, where the first four choice sets were static, the same for all respondents, and the last four choices sets were individually adaptive. Ten respondents were included in the process, by drawing 10 samples from a multivariate normal distribution, and their corresponding responses were randomly simulated.

One may think that this was a really small scenario; however, it was purposefully chosen in this way to avoid a large number of computations. It is important to note that this process of improving speed requires to make modifications in the code and try them all in an iterative way. Therefore, a larger case scenario would not have been ideal.

It is also important to clarify that 10 samples from the prior distribution is not enough to get a good design. In practice, at least 1000 samples are used because by the law of large numbers, such estimates are known to converge to the true value of the integral. Kessels et al. (2009) discussed about the optimal number of samples one should consider and mentioned different approaches to get these samples, such as orthogonal arrays and quasi-Monte Carlo samples.

Figure 4.2 shows the profiling results obtained from the `SeqDB` function. There are three columns: Code, memory and time. Code specifies what part of the code is being measured. Memory shows the amount of memory allocated or deallocated measured in megabytes (MB) and time is the time spent in milliseconds (see RStudio, 2019).

Code	Memory (MB)		Time (ms)
▼ SeqDB	-156.3	159.2	3000
▼ apply	-156.3	159.2	3000
▼ FUN	-156.3	159.2	3000
▼ apply	-156.3	158.6	2990
▼ FUN	-143.9	157.5	2990
▶ InfoDes	-131.6	134.6	2630
▶ det	-12.3	16.1	290
rbind	0	1.8	30
▶ InfoDes	0	0	10

Figure 4.2: Profiling `SeqDB` function

The first row shows that 159.2 MB were allocated and 156.3 were deallocated, so the problem of speed was not related to memory allocation because the difference was only about 3 MB. Additionally, the amount of time required to get the next choice in a small example as this was 3 seconds; as it was mentioned above. Now, on the second and the third row there is an `apply` function that calls another function. This part of the code corresponds to the computation of the D_B-optimality criterion for each possible choice set, taken from the candidate list. Rows 4 and 5 show that inside that function that computes the D_B- optimality, there is another `apply` function that calls another function. This corresponds to the computation of the D-optimality criterion for each respondent. Inside this function there are two blocks of code: `InfoDes`, where the information matrix is computed and `det`, where the determinant of the information matrix is computed. At this point, it is easy to see that most of the processing time was

related to the function that computes the information matrix. Therefore, this was the piece of code that needed to be improved.

`InfoDes` is a function of 10 lines implemented in `R` that does not have any loop or complicated scheme on it. It is just matrix algebra (see the definition of the information matrix in equation 2.4). Since this function could not be improved further with just `R` language, `C++` was the solution. This implementation required `Rcpp` and `RcppArmadillo` packages for two reasons: First, to make an easy transition between `R` objects and `C++` objects (Eddelbuettel and François, 2011; Eddelbuettel, 2013; Eddelbuettel and Balamuta, 2017), and second to being able to use a `C++` linear algebra library, called `Armadillo` (Sanderson and Curtin, 2016, 2018; Eddelbuettel and Sanderson, 2014).

This implementation in `C++` was not as straightforward as one may think. From a function of 10 lines of code in plain `R` language, the resulting equivalent in `C++` was a function of almost 120 lines. This huge difference is mainly explained by how objects and memory are used inside `C++` environment. As a result, this new implementation was almost $6\times$ times faster than the original one (see Figure 4.3). It is important to mention that the measure of performance was conducted by the package `microbenchmark`, where each function was executed 100 times and the corresponding processing time was saved (Mersmann, 2018). Then, the comparison is straightforward by means of plots or summary statistics, such as max, min, quartiles, mean and median.

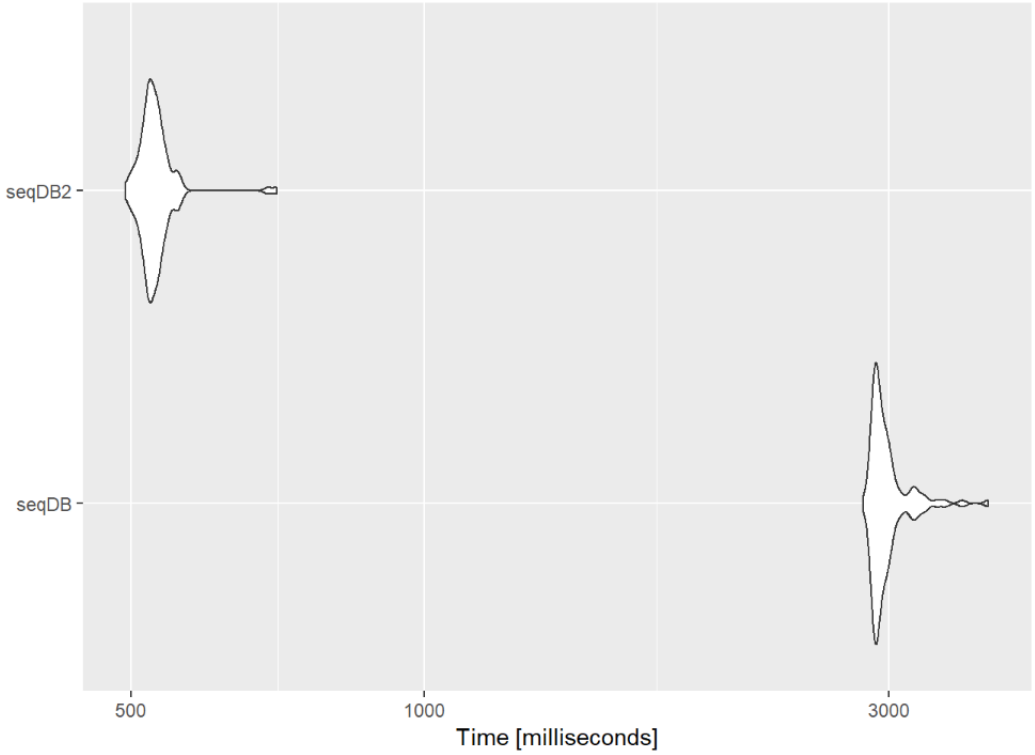


Figure 4.3: Difference in processing time

This process of finding the next bottleneck and improve it, continued several times up to a point where it was considered that the function was fast enough, Figure 4.4

presents these results. Five different versions of the same function are compared: in some cases the improvement was successful as in `SeqDB2`, `SeqDB4` and `SeqDB5`, in other case, `SeqDB3`, it was not possible to improve it further. The final version, shown in the plot as `SeqDB5`, is a combination of `R` and `C++` code, where regular conditionals and basic functions are implemented in `R` and linear algebra expressions are evaluated in `C++`, specifically the computation of the information matrix and its determinant. It is important to mention that even though `SeqDB4` looks slightly better than `SeqDB5` in the plot, the quartiles of the latter implementation were actually better, this why it was chosen.

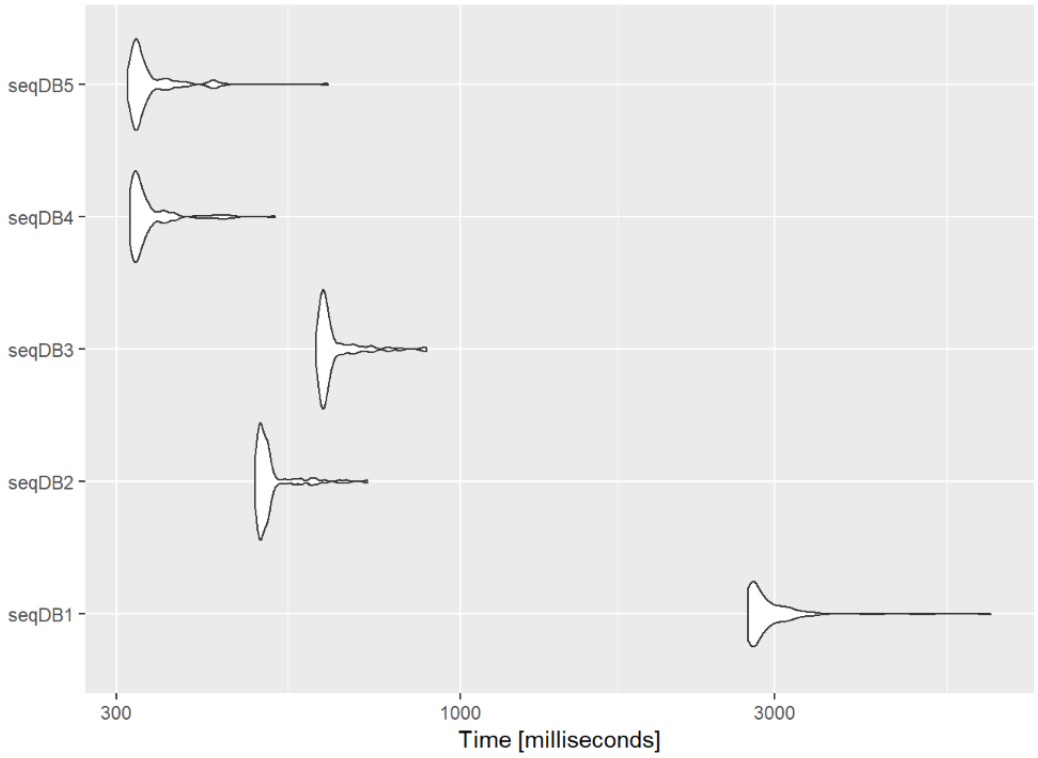


Figure 4.4: Difference in processing time different improvements

Additionally, since most of the calculations are repetitive and independent of each other, parallel computing was also used. The idea behind parallel computing is to exploit the architecture of modern Central Processing Units (CPU), composed by different number of processors and cores, in the computation of algorithms that do not follow a sequential order (see Jones, 2017). In `R`, specifically the package `parallel`, this is internally handled in different steps, as it is mentioned in R-core (2018):

- Create a set of copies of R running in parallel.
- Send data required to those copies.
- Split the task in chunks and send each chunk to a copy.
- Wait for all copies to complete their tasks.
- Combine results.

- Close and delete these copies.

In this particular situation, the computation of the D_B-optimality criterion in each possible choice set is necessary to decide which is the one that will maximize the information given by the respondent. This computation can be conducted in parallel because the result of one particular choice set is completely independent of the other ones. As a consequence, this option was included in the function so that the package user can decide whether parallel computing should be used or not.

In Figure 4.5 the comparison between both scenarios is presented. Four different functions are compared here: The original implementation with the only modification of using parallel computing, called *Parallel*. The improvement in the computation of the information matrix with parallel computing, called *Parallel_Infodes*. The improved function using `C++` linear algebra computations, called *NoParallel_cpp*. Finally, the improved version of the function combined with parallel computing, called *Parallel_cpp*. In these results, it is pretty clear that the implementation that involves parallel computing and `C++` outperforms any other implementation done so far. Hence, this was the final implementation of the `SeqDB` function.

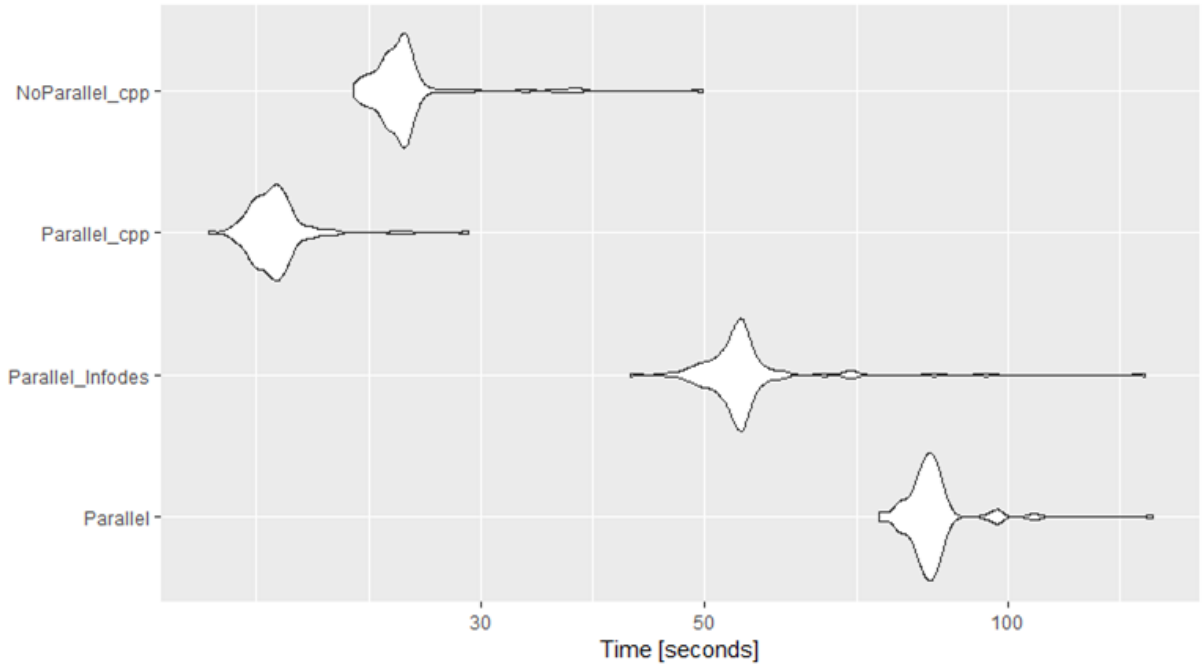


Figure 4.5: Difference in processing time different using parallel computing

4.2 Improving processing time in regular static designs

Contrary to adaptive designs, static designs are easier to get because the algorithm consists in improving an initial random design without the consideration of new responses of any kind. However, the need of having the results as quickly as possible

also applies in this situation due to the number of initial designs considered. The larger this number, the better because the probability of finding a good solution increases. As a consequence, `Modfed` function needed to be improved as well.

In terms of coding, this function can be seen as a simpler version of the `SeqDB` function. The information matrix and its determinant have to be computed each time a new profile is exchanged from the candidate list. Hence, the implementation in `C++` made before was also used here. These `C++` functions improved enough the processing time of the algorithm, so no further improvements were needed.

4.3 Comments about update formulas

It has been mentioned that the search for an optimal design requires the computation of an optimality criterion several times. Most of these optimality criteria depend on the Information Matrix and its determinant as it is the case with the D_B-optimality. In the literature, several ways to update this determinant without the need of computing the determinant of the whole matrix have been developed. Kessels et al. (2009) proposed updating the Cholesky decomposition of the information matrix as an intermediate process in any algorithm, Modified Fedorov or Coordinate Exchange, to get the updated optimality criterion after changing one profile from the initial design. Arnouts and Goos (2010) made something similar with linear models and the use of split-plot designs, where the Sherman-Morrison–Woodbury formula is used to update the optimality criterion after a change made in the initial design.

The particular way the Information Matrix is computed in the MNL model, consists in making some matrix multiplications for each choice set and then sum these results, see equation 2.4. This structure can be used in the iteration process of updating the optimality criterion because a change in a particular profile leads to a change in just one term of this summation.

This is, however, not the case for the package as it is implemented now. Every time a new change is conducted on the initial design, the whole information matrix is computed again as well as its determinant. This is happening because the Information Matrix is computed with a different formula:

$$\mathbf{M}(\mathbf{X}, \boldsymbol{\beta}) = (\mathbf{X} \circ \mathbf{p})^\top \mathbf{X} - (\mathbf{R}^\top \mathbf{R}) \tag{4.1}$$

Where $\mathbf{X}$ is the full design matrix, $\mathbf{p}$ is the probability of each alternative in each choice set, $\circ$ is the Hadamard product, also known as elementwise product, and $\mathbf{R}$ is the row sum of $(\mathbf{X} \circ \mathbf{p})$ within each choice set.

This implementation of the Information Matrix in the package makes the use of fast update formulas difficult for two reasons: First, the majority of fast formulas take advantage of the original expression by just updating the corresponding term of the modified choice set, which is, of course, not possible to do it with this expression. Second, the whole package depends on this computation, so a change in this implementation would require a change in practically every function developed.

For these reasons, fast formulas were not considered in the process of improving the processing time of both functions.

Chapter 5

Implementation of Kullback-Leibler criterion

As it was mentioned before, the advantage of the Kullback-Leibler criterion with respect to the D_B-optimality criterion is that it does not need to compute the information matrix and its determinant iteratively to get the next choice set. Instead, the KL-distance between the current posterior of the coefficients and the updated posterior one can obtain with the additional response is maximized.

This algorithm was already implemented in the package, but the result of some simulations were not as successful as it is mentioned in Crabbe et al. (2014). For this reason, the associated functions were removed temporarily from the package itself until further checks. The goal of this section is to explain the process to get a useful function that performs well.

5.1 Looking for possible mistakes in the implementation

Given that some simulations made by the developer of the package showed that the implementation in `R` was not giving good results, it was necessary to identify possible mistakes in the translation and generalization of the original code. Translation because the simulation presented in Crabbe et al. (2014) was performed in `SAS` and generalization because the purpose of the function created in `R` was to gain speed generating a new choice set in any possible design, not only those scenarios considered in the simulation.

Consequently, the first step consisted in understanding the theory explained in the paper and associating specific calculations to its implementation in `SAS`. Although this task seems to be straightforward, some problems arose when the `SAS` code was checked. It turns out that the script had almost 500 lines of code with no comments or indentation whatsoever. Thus, the burden of trying to understand what the programmer was thinking at the moment the function was implemented is multiplied by the fact that the code

needed to be organized and commented.

Once this process was considered as finished, it was necessary to repeat the same process of checking the code and locating the important calculations implemented in R. Fortunately, this process was easier because of the similarities between both implementations. Nonetheless, some differences were found when comparing both implementations using the exact same parameters. In the following, the comparison is presented together with an impact measure of possible differences. The scale of the impact was subjectively defined according to the differences seen in each result:

- *Generation of a list with all possible candidate sets based on a list with all possible profiles:* This algorithm depends on a list of candidate sets to get the next choice that maximizes the KL-criterion. The implementation in SAS made sure the same profile was not presented to the respondent more than once in the same choice set, meaning that it was not possible to have a choice set with repeated alternatives. On the contrary, the implementation in R considered all possible combinations of profiles with no restriction whatsoever.
  - **Impact: Low**. From a computational point of view, having choice sets with the same alternatives increase the computing time because the dimension of the full list is larger. From a practical perspective, although the possibility that the final design is composed of candidate sets with the same alternative exists, it is highly unlikely that this happens because the information is not being maximized in such scenario.
- *Assignation of random response to a candidate set:* In order to get the following choice set, a random response is needed to update the posterior distribution of the coefficients. This function is only needed for simulations. There was no difference between both implementations.
  - **Impact: None**
- *Computation of the logarithm of the posterior distribution:* The logarithm of the posterior distribution is needed in the calculation of the KL-criterion (see equation 2.9). There was a difference between both implementations corresponding to a minus sign in the constant term of the logarithm of the density function. This minus sign was missing in the implementation in R.
  - **Impact: High**. Missing this minus sign led to completely different results.
- *Evaluation of the hessian at a specific point:* Once the logarithm of the posterior distribution is optimized, the hessian needs to be evaluated at that point. There was no difference between both implementations.
  - **Impact: None**
- *Sampling from the posterior distribution of the coefficients:* Samples from the prior and importance distributions are needed in the computation of the KL-criterion. The multivariate t-distribution is considered as the importance distribution with a specific degrees of freedom. The only difference between both implementations was the number of degrees of freedom: In SAS 8 degrees of freedom were used, whereas in R only 6 were used.

  - **Impact: Low**. With respect to the degrees of freedom, Yu et al. (2011) stated that any value between 5 and 15 seems to give satisfactory results.

- *Computation of the probability of each alternative in a candidate set:* it is needed in the computation of the KL-criterion. There was no difference between both implementations.
  - **Impact: None**

- *Computation of the KL-criterion:* Assuming that both implementations use the same samples and weights from the importance sampling function, no differences between both implementations were found.
  - **Impact: None**

Having discussed these differences with the supervisor and the developer of the package and updated some parts of the code, it could be considered that the implementation of the core of the algorithm in `R` was equivalent to the implementation in `SAS`. Now, to be completely sure about the quality of the function, simulations were conducted, and the results are presented later in this document.

5.2 Details about the `SeqKL` function

Since the core of the algorithm was already finished, there were some details that were necessary to implement in the function to ensure the proper work inside the package. This section enumerates these details:

- *Handling input errors:* All functions in the package handle possible errors associated with the input parameters. Easy mistakes such as the class of an object, the dimension of a matrix, whether there are problems with specific constants in the design are also controlled in this updated function.
- *Adding alternative specific constants:* As in `SeqDB` function, this function can also deal with alternative specific constants.
- *Adding no choice alternative in the choice sets:* There are some situations when neither of the profiles in the choice set are good enough to the respondent. There is now an option to create designs when one of the alternative is ”none of the above”, as it happens in any other function in the package. This was implemented by adding an alternative with all attributes at zero and its correspondent alternative specific constant in the design.

Chapter 6

Implementation of Coordinate Exchange Algorithm

As it was mentioned before, the success of this package is related to the possibility of creating discrete choice experiments in a reasonable amount of time. In cases where the number of attributes or levels (or both) increase significantly, the computation time explodes if the Modified Fedorov Algorithm is used. This is caused by how the algorithm is defined to optimize each row, because it exchanges all possible profiles from the candidate list.

On the other hand, the Coordinate Exchange Algorithm (CEA) is well known to produce efficient designs in less computing time that the Modified Fedorov. The reason being how the optimization works within in each profile: instead of trying all possible profiles from a candidate list, it exchanges all possible levels within the attributes. Tian and Yang (2017) illustrate the difference between both algorithms in an easy way: To update a row in the initial design, the Modified Fedorov requires the calculation of the D_B-optimality criterion, or any criterion, $\prod_{k=1}^{K} l_k$ times, whereas the CEA requires the calculation of only $\sum_{k=1}^{K} l_k$ D_B-optimalities, where K is the number of attributes in the design and l_k the corresponding levels.

To give an easy example, let's say that a $3^3/2/12$ discrete choice experiment is needed. Then, the candidate list has $3^3 = 27$ possible profiles and the design matrix is then composed of 24 rows. To optimize the first alternative of the initial design, the Modified Fedorov will have to compute 27 D_B-optimality criteria, while the CEA will have to compute only 9 $(= 3 + 3 + 3)$. Since there are 24 profiles to optimize, the first algorithm will have to compute the optimality criterion 648 $(= 24 \times 27)$ times and the second just 216 $(= 24 \times 9)$ times. Therefore, the CEA algorithm is by definition faster in terms of computing time.

To implement this algorithm, the `Modfed` function, used to create an optimal design with the Modified Fedorov algorithm, was taken as a reference for further developments. This was considered because many details had to be included in this new function such as the input parameters, which should be similar to other functions in the package, and the output of the function, which should be equal to the `Modfed` function.

On the other hand, different characteristics had to be implemented that were not in any other function:

1. Since the CEA does not depend on a candidate list, the initial random design has to be created with respect to the levels of each attribute. To do so, the levels of each attribute have to take into account the corresponding coding scheme. This task is not easy because all possible combinations of schemes can be implemented in a design, and each of them has their own particularities.
2. The CEA function has to be able to create designs with alternative specific constants and also with the possibility of creating a "None of the above" option. Although this was already implemented in the Modified Fedorov algorithm, some modifications were necessary to ensure the function to run properly.
3. The development of the core of the algorithm was by far the hardest part to code for the same reason that a candidate list is not provided. From a programming point of view, the difference is that in the Modified Fedorov algorithm, only two matrices have the information required to optimize a profile, namely the initial design and the candidate list. On the other hand, in the Coordinate exchange there is a vector for each attribute considered in the design that allocates the information about the corresponding levels. So several objects have the information required to optimize a profile, namely the initial design and a vector for each attribute.

It is important to mention that parallel computing was also implemented in this function to boost the processing time, and the calculation of the D_B-optimality criterion was also performed using the same improved `C++` functions used in the Modified Fedorov algorithm. As a result, this new function is able to find optimal designs in a more efficient way than any function present in the package. However, some simulations were needed to be completely sure about the efficiency of this implementation. The next chapter provides these results.

Additionally, the CEA was also implemented in adaptive designs by taking `SeqDB` function as a reference for further developments. As it was previously mentioned, this was considered because the input parameters should be similar to other functions in the package, and the output had to be equal to `SeqDB` function to ensure the correct functioning of the `Shiny` application.

The main difference between the sequential Modified Fedorov and the sequential Coordinate Exchange algorithm is how the following choice set is chosen. The first version makes an exhaustive search by considering all possible choice sets to get the following choice set, whereas the second implementation only considers a subset of possible choice sets by selecting randomly the level of each attribute.

Chapter 7

Simulation study

Since new implementations were developed in this thesis, it was important to measure in an objective way, whether the improvements made were indeed working properly. As a consequence, two different sets of simulations were conducted in order to compare the speed and efficiency of the algorithms in static designs, and the efficiency of the criteria developed for adaptive designs. In the following, a description of each scenario is presented together with the corresponding results.

7.1 Comparison of Modified Fedorov and Coordinate Exchange implementations

The main reason to make simulations comparing the Modified Fedorov and the Coordinate Exchange algorithm was to be completely sure that the implementation of both of them in the package was correct. As it was mentioned before, it is well known that the CEA is way faster than the Modified Fedorov algorithm; however, nothing has been said about the efficiency of the final design.

For this reason, four different scenarios were considered in this section. The first two correspond to the same designs applied by Kessels et al. (2009) to show that a new algorithm, proposed by them, performed better than the Modified Fedorov algorithm. The structure of the first design is $3^2 \times 2/2/12$ and the second is $3^2 \times 2/3/8$. Although both designs have the same number of attributes and levels, the difference is the number of profiles per choice set and the number of choice sets. Effects-type coding was used in all attributes, resulting in a parameter vector β with 5 entries. The prior distribution of β was a multivariate normal distribution with $\boldsymbol{\mu} = [-1, 0, -1, 0, -1]^{'}$ and $\Sigma = \boldsymbol{I}_5$. From this distribution 100 samples were drawn corresponding to the number of respondents. To obtain the optimal design, 50 different initial random designs were used, and this process was repeated 100 times.

The third and fourth scenario were taken from Crabbe et al. (2014), where the designs are related to air travel choice in the San Francisco bay area and with the choice of

flights connecting the Canary Islands to the Iberian Peninsula. The structure of the design in the third scenario is $5\times4^2/2/15$, with all factors being continuous. The parameter vector β has three elements and a multivariate normal distribution is assumed to be its prior distribution with $\boldsymbol{\mu} = [-0.068, -0.083, 1.623]'$ and $\boldsymbol{\Sigma} = diag(0.0009, 0.0015, 0.5622)$. The structure of the design in the fourth scenario is $3 \times 2 \times 3/3/15$, with all factors being categorical and dummy-coded. The parameter vector $\boldsymbol{\beta}$ has five elements and a multivariate normal distribution is assumed to be its prior distribution with $\boldsymbol{\mu} = [0.419, 0.700, 1.355, 1.638, 1.690]'$ and $\boldsymbol{\Sigma} = diag(0.0466, 0.9055, 2.6322, 0.5684, 1.1071)$. From these distributions 100 samples were drawn corresponding to the number of respondents. To obtain the optimal design, 50 different initial random designs were used, and this process was repeated 100 times. Table 7.1 shows an overview of this information.

Scenario	**Structure**	**Coding**	**Prior**
I	$3^2 \times 2/2/12$	Effect	$\boldsymbol{\mu} = [-1, 0, -1, 0, -1]'$ $\boldsymbol{\Sigma} = \boldsymbol{I}_5$
II	$3^2 \times 2/3/8$	Effect	$\boldsymbol{\mu} = [-1, 0, -1, 0, -1]'$ $\boldsymbol{\Sigma} = \boldsymbol{I}_5$
III	$5 \times 4^2/2/15$	Continuous	$\boldsymbol{\mu} = [-0.068, -0.083, 1.623]'$ $\boldsymbol{\Sigma} = diag(0.0009, 0.0015, 0.5622)$
IV	$3 \times 2 \times 3/3/15$	Dummy	$\boldsymbol{\mu} = [0.419, 0.700, 1.355, 1.638, 1.690]'$ $\boldsymbol{\Sigma} = diag(0.0466, 0.9055, 2.6322, 0.5684, 1.1071)$

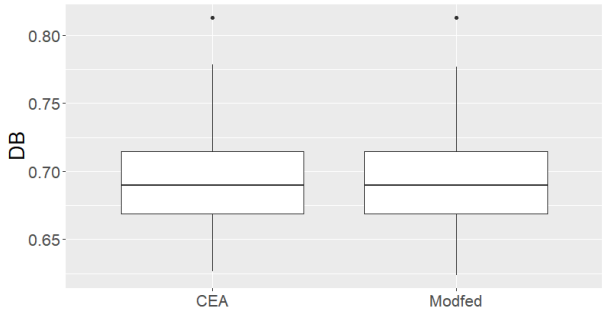
Table 7.1: Simulation scenarios static designs

To measure the difference between both algorithms, the D_B-optimality criterion of the obtained design and the processing time needed to get it were used [1]. The same samples obtained from the prior distribution in each scenario were used for each algorithm, in order to be sure that any difference obtained can be attributed to the algorithm itself. The same convergence criterion was used in both algorithms, which consisted in doing an entire iteration without conducting any exchange. Figure 7.1 presents these results, where it can be seen that the efficiency of the design is pretty much the same in all scenarios. In general, the median D_B-optimality of the Coordinate Exchange Algorithm is slightly higher because the search for best the alternative is not as exhaustive as in the Modified Fedorov. In terms of processing time, the Coordinate Exchange algorithm outperforms the other algorithm in orders of magnitude in all scenarios considered.

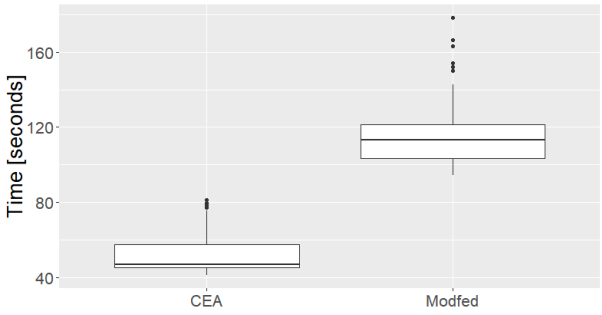
Having these results, where the Coordinate Exchange produces designs faster with the same quality than the other algorithm, raise the question of why is the Modified Fedorov algorithm kept in the package. It turns out that there are specific problems where constraints on the levels are needed to get an optimal design. These constraints are easy to handle in the Modified Fedorov by just removing those profiles that are restricted from the candidate list. On the contrary, the implementation of the Coordinate Exchange does not allow to put any constraints in the design. Therefore, it can be

[1] All computations were performed using a Lenovo Thinkpad personal computer with a 2.5 GHz Intel Core-i7 processor and 16 GB RAM.

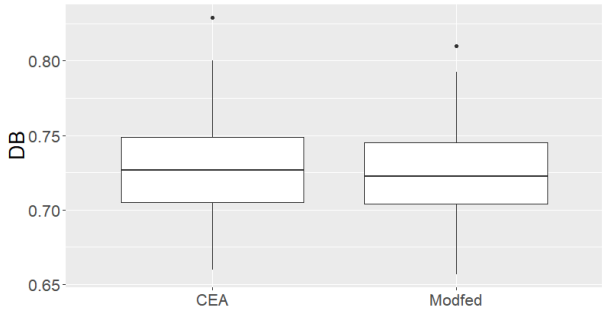
concluded that the Coordinate Exchange algorithm should be used in cases where no constraints are needed and the Modified Fedorov is useful when restrictions are absolutely necessary.



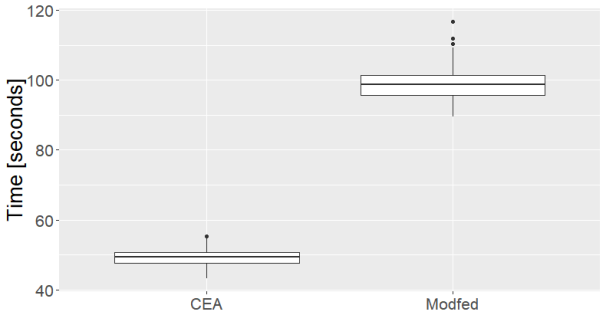
(a) Scenario I: D_B-optimality



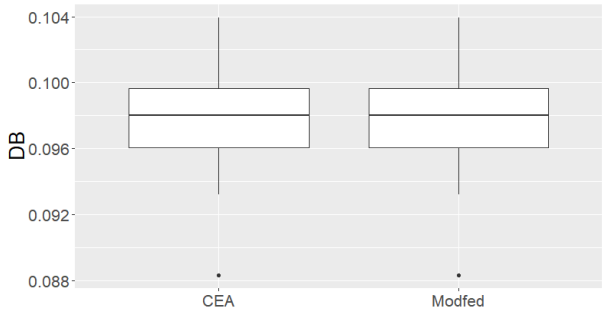
(b) Scenario I: Processing time



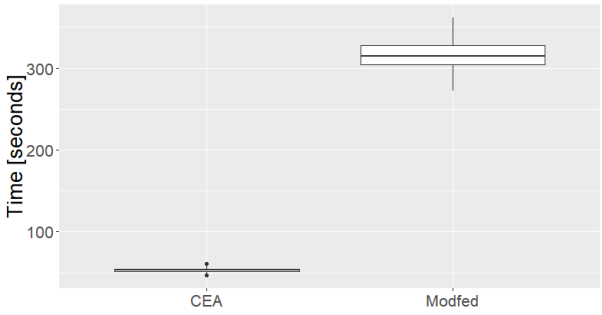
(c) Scenario II: D_B-optimality



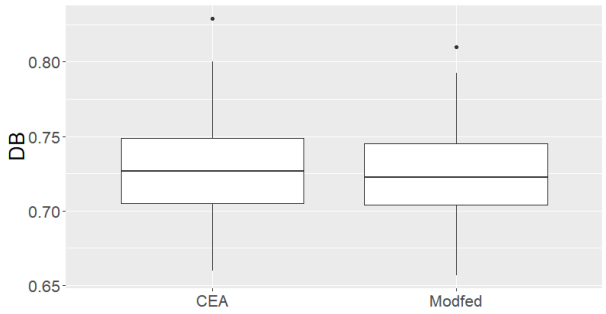
(d) Scenario II: Processing time



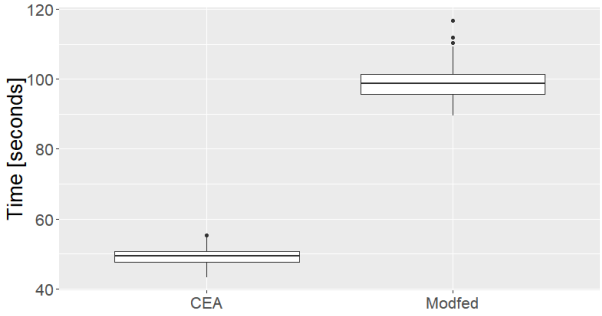
(e) Scenario III: D_B-optimality



(f) Scenario III: Processing time



(g) Scenario IV: D_B-optimality



(h) Scenario IV: Processing time

Figure 7.1: Comparison between algorithms

7.2 Comparison of DB-optimality and Kullback-Leibler criteria

The main reason to make simulations comparing the Kullback-Leibler criterion and the D_B-optimality criterion was to be sure that the KL indeed generated equally efficient designs. As it was mentioned before, the results obtained from the implementation of this criterion in the package were not consistent with the results presented by Crabbe et al. (2014), so it was important to check whether the revisited and complemented implementation of the criterion produced similar results.

For this reason, two different scenarios were considered in this section. The structure of both designs is $3^3/2/12$, which using dummy coding results in a parameter vector β with 6 elements. The prior distribution of β in each scenario was a multivariate normal distribution with $\boldsymbol{\mu} = [0.4, 0.8, -0.8, -1.6, 1.6, 3.2]'$ and different covariance matrix, defined as:

$$\Sigma_1 = \begin{bmatrix} 0.0100 & 0.0066 & -0.0044 & -0.0088 & 0.0044 & 0.0088 \\ 0.0066 & 0.0400 & -0.0088 & -0.0176 & 0.0088 & 0.0176 \\ -0.0044 & -0.0088 & 0.0400 & 0.0352 & -0.0176 & -0.0352 \\ -0.0088 & -0.0176 & 0.0352 & 0.1600 & -0.0352 & -0.0704 \\ 0.0044 & 0.0088 & -0.0176 & -0.0352 & 0.1600 & 0.1760 \\ 0.0088 & 0.0176 & -0.0352 & -0.0704 & 0.1760 & 0.6400 \end{bmatrix}$$

and

$$\Sigma_2 = \begin{bmatrix} 0.1600 & 0.1056 & -0.0704 & -0.1408 & 0.0704 & 0.1408 \\ 0.1056 & 0.6400 & -0.1408 & -0.2816 & 0.1408 & 0.2816 \\ -0.0704 & -0.1408 & 0.6400 & 0.5632 & -0.2816 & -0.5632 \\ -0.1408 & -0.2816 & 0.5632 & 2.5600 & -0.5632 & -1.1264 \\ 0.0704 & 0.1408 & -0.2816 & -0.5632 & 2.5600 & 2.8160 \\ 0.1408 & 0.2816 & -0.5632 & -1.1264 & 2.8160 & 10.2400 \end{bmatrix}$$

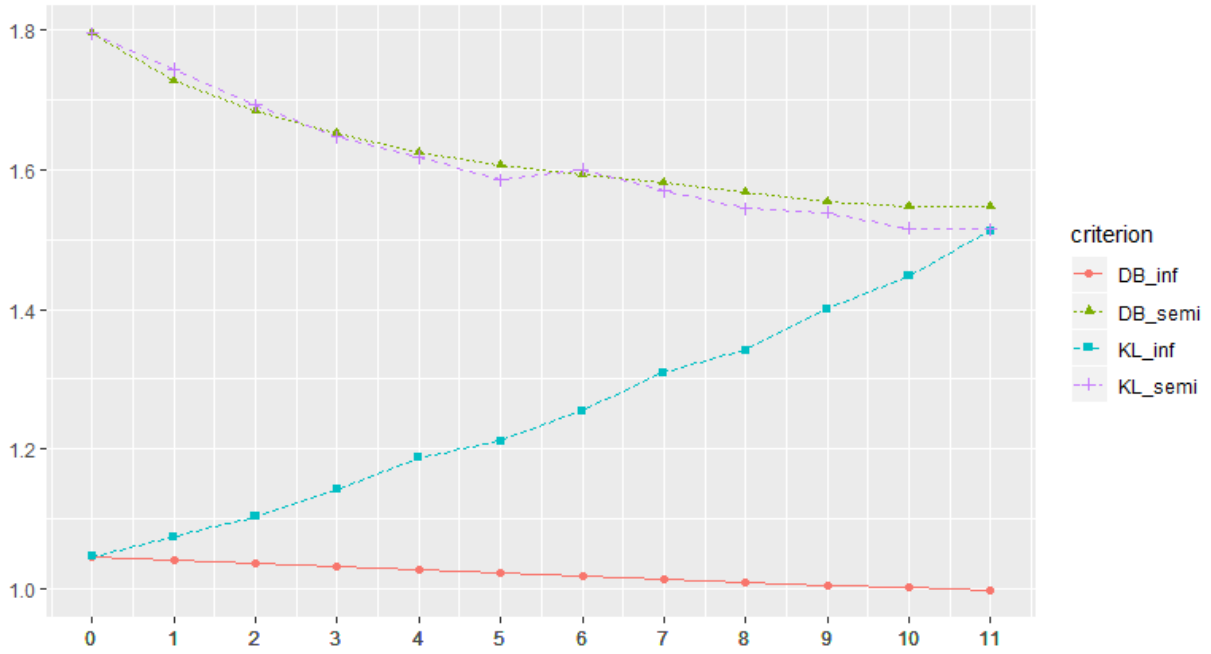
Note that the difference between these scenarios is mainly influenced by the variance of the parameters. In the first case, there is low heterogeneity between respondents, represented as small variances, whereas in the second there is high heterogeneity. It is important to mention that in both scenarios the correlation matrix was exactly the same, even though different covariance matrices were defined. From these distributions 100 samples were drawn corresponding to the number of respondents.

Within each scenario, 12 different kinds of designs were obtained corresponding to the number of adaptive choice sets considered. When zero adaptive choice sets were considered, the resulting design is not adaptive but static. When one adaptive choice set was considered, the first eleven choice sets were static and the remaining was adaptive. When two adaptive choice sets were considered, the first ten choice sets were static and the remaining two were adaptive. This pattern continues until eleven adaptive choice sets were considered in the design and just one choice set was static. The goal of specifying the simulations in this way was to be sure that the KL criterion works fine regardless the number of choice sets that were in the initial static design.

For each kind of design within each scenario, 100 different designs were generated corresponding to the number of respondents, and in each of them, the responses for each choice set were randomly assigned. Then, the individual D-errors were obtained and summarized by taking the mean over all designs[2]. This process was repeated 100 times because of the randomness introduced in the simulation. To obtain the adaptive choice sets in all designs two different variants were considered: a semi-informative prior, where $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\Sigma} = \mathbf{I}_6$, and an informative prior where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ were equal to the true values defined above.

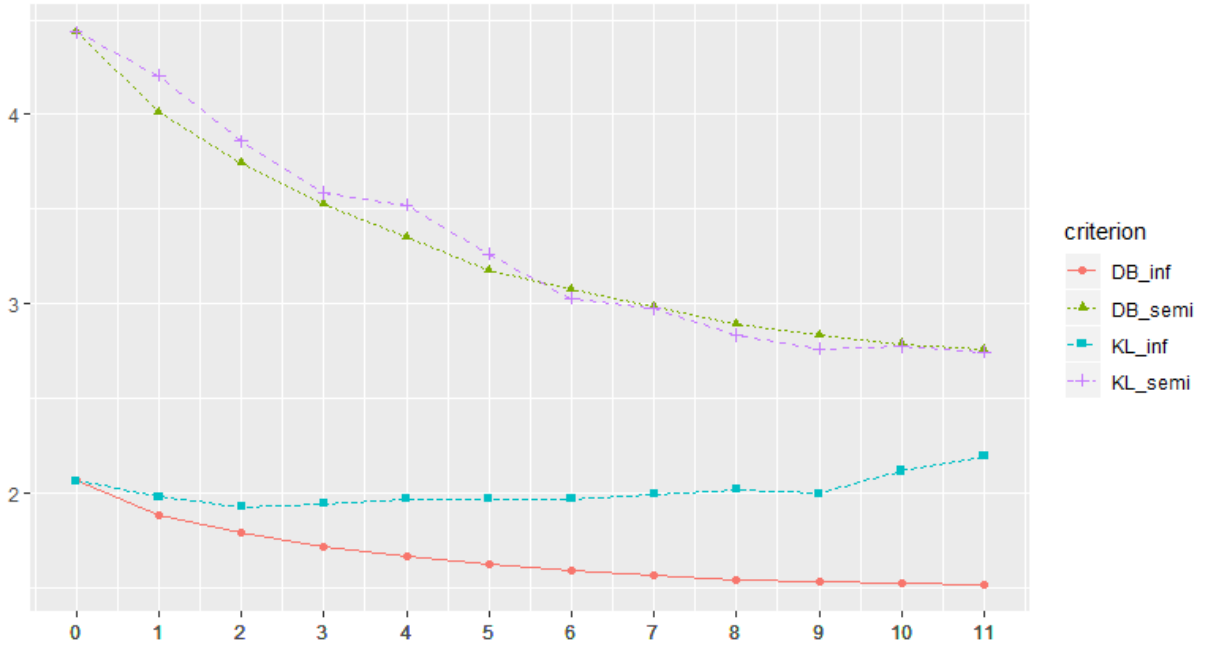
To illustrate this process, let's say that the first scenario, low heterogeneity, is analyzed by using six choice sets in the initial static design. These six choice sets are randomly obtained for all 100 respondents in the simulation. Then, random responses are assigned to each choice set in each respondent. Next, the remaining six choice sets of the design are adaptive by taking into account the prior information. At the end of this process, 100 different designs are obtained and the corresponding D-error is computed in each of them. Because the initial static design and the responses are random, this whole process is repeated 100 times. As a result, ten thousand D-errors are summarized by taking the mean over all of them.

Figure 7.2 shows the resulting D-error for each scenario summarized over all respondents and replicates. In these plots, the horizontal axis represents the number of adaptive choices per design and the vertical axis represents the averaged D-error. If the KL criterion works, the summarized D-error should decrease as the number of adaptive choice set increases.



(a) Scenario I. Low heterogeneity

[2]Note that D-error refers to the D_B-optimality criterion of the final design. In this section, this terminology is used to avoid confusion with respect to the criterion that is begin compared.



(b) Scenario II. High heterogeneity

Figure 7.2: Comparison between criteria

In the case where low heterogeneity in the respondents was assumed (plot 7.2a), the D-error of the D_B-optimality criterion decreases slightly when the informative prior was employed, and it decreases when the semi-informative prior was used. These trends are taken as a reference for the comparison with the KL criterion. When the semi-informative prior was used in the KL criterion, the D-error decreases at the same rate that the D_B-optimilaty does. On the other hand, when the informative prior was used, the D-error increases as the number of adaptive choice set increases. A totally unexpected behavior because this trend should be similar to the D_B-optimality criterion.

Regarding the case where high heterogeneity in the respondents was assumed (plot 7.2b), both D_B-optimality and KL criteria decrease when the semi-informative prior was employed. In fact, in some cases the designs generated with the KL criterion are more efficient than those designs generated with the D_B-optimality criterion. On the other hand, when the informative prior is used, the D-error of the D_B-optimality criterion decreases slightly, whereas the trend for the KL criterion remains constant and increases when the number of adaptive choice sets is larger than ten.

In summary, the implementation of the KL criterion in the package seems to be working well because in specific scenarios, like in semi-informative priors, the efficiency of the designs is similar to the D_B-optimality criterion. However, there are still doubts about the efficiency of the criterion under different scenarios, like in informative priors. For this reason, further simulations are needed to determine under which circumstances the KL criterion produces equally efficient designs, or better, than the D_B-optimality criterion.

Chapter 8

Improvement in coding practices

The last objective of the thesis consisted in reorganizing functions, removing redundancy and improving in general the way the package was coded. In this section, some of the changes made are enumerated:

- To improve the overall way the package was coded, the suggestions given by Wickham (2015a), in the chapter about *Style guide*, were followed:
  - Spacing: Placing spaces around all infix operators ($<, >, =, +, \dots$), conditionals and loops make the code look cleaner and it is easier to understand.
  - Indentation: Although this was not a generic problem in the package, there were some cases where it was needed.
  - Line length: The maximum number of characters per line allowed was set to 80. Every time a line was longer than this number, it was divided into multiple lines.
  - Assignment: Although this did not happen in the package very often, there were some cases where = was replaced by <-. Note that one option is not better than the other, but it is better to stick to only one of them.
- In terms of redundancy, two different situations were found in the package:
  - Redundancy inside a function: There were some functions where the same conditional was used more than once to run a specific part of the code. For instance, when an alternative specific constant is given in a design, there was a conditional in the function that handles errors associated with the format of it and then the same conditional was applied to merge these constants to the final model. This kind of redundancy was removed, making the code easier to debug.
  - Redundancy between functions: This only happened once, and it was associated with the computation of the probability of the alternatives in a specific choice set given a sample from the prior distribution of the coefficients. There were two different functions that compute these probabilities, but the only difference was the object that was returned. Hence, they were merged,

and a condition was put at the end of the function to decide what object to return.

- Automatizing tests in the development of new functions: As it is suggested by Wickham (2015b), every time a new function is being developed, it is also tested to see whether the desired results are indeed obtained. An easy way to automate these tests is by using the package `testthat` (Wickham, 2011). It creates a folder in the package called *Tests* where all possible tests can be set and run easily. This was used in the development of the Coordinate Exchange algorithm for static and adaptive designs.

Chapter 9

Conclusion

As in the introduction of the package, the best way to show the process developed in this thesis is through Figure 9.1. As before, this flow chart represents the capabilities of the package with respect to the type of design the user wants, but it also presents the modifications made in this process, colored in green.

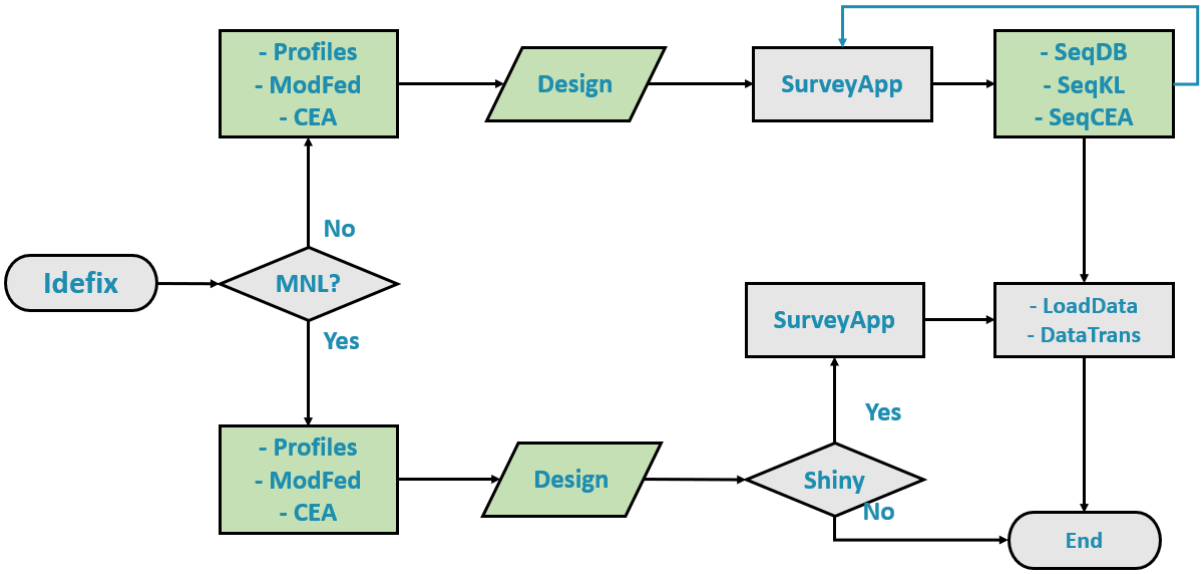


Figure 9.1: Modified status of the package after this thesis

With respect to those functions needed to create static designs, the processing time of the `Modfed` function was improved by orders of magnitude. The `CEA` function was implemented from scratch, using also `C++` functions, and it was shown that in regular scenarios, it produces designs as efficient as the Modified Fedorov algorithm in significantly less time.

Regarding the functions related to adaptive designs, the processing time of the `SeqDB` function was improved by orders of magnitude, implementing some parts of the code in `C++`. The `SeqKL` function, which involves the Kullback-Leibler criterion, was revisited,

modified and updated in order to give similar results as any other function in the package. And the `SeqCEA` function was implemented from scratch to speed the process of getting the next choice set in an adaptive environment.

There are still different ways to improve this package even further. First, by trying to revisit the way the Information Matrix is implemented and checking whether any fast update formula can be used to improve the processing time even more. Second, by adding more capabilities to the package in terms of optimality criteria, with special emphasis is those related to obtain precise predictions, such as G- and V- optimality criteria. Third, by allowing the user to generate optimal designs with partial profiles, which is useful when the number of attributes or levels is large. Finally, by being more user friendly in terms of documentation and help files: using `vignettes` to explain how to the package should be used, may increase the number of users.

The whole working process associated to the development of this thesis can be found in `https://github.com/danielgils/thesis`. The forked repository of the package and the whole history of commits can be found in `https://github.com/danielgils/idefix` and the last version of the package can be easily found in `https://github.com/traets/idefix`. At the moment this thesis was written, the last version was not available on CRAN yet, but the official version can be found at `https://cran.r-project.org/web/packages/idefix/index.html`.

Bibliography

Arnouts, H. and Goos, P. (2010). Update formulas for split-plot and block designs. *Computational Statistics and Data Analysis*, 54:3381–3391.

Chang, W., Cheng, J., Allaire, J., Xie, Y., and McPherson, J. (2018). *shiny: Web Application Framework for R*. R package version 1.2.0.

Chang, W. and Luraschi, J. (2018). *profvis: Interactive Visualizations for Profiling R Code*. R package version 0.3.5.

Cook, R. D., Nachtsheim, C. J., and Cook, R. D. (1980). A Comparison of Algorithms for Constructing Exact D-Optimal Designs Linked references are available on JSTOR for this article : A Comparison of Algorithms for Constructing Exact D-Optimal Designs. *American Society for Quality*, 22(3):315–324.

Crabbe, M., Akinc, D., and Vandebroek, M. (2014). Fast algorithms to generate individualized designs for the mixed logit choice model. *Transportation Research Part B: Methodological*, 60:1–15.

Eddelbuettel, D. (2013). *Seamless R and C++ Integration with Rcpp*. Springer, New York. ISBN 978-1-4614-6867-7.

Eddelbuettel, D. and Balamuta, J. J. (2017). Extending extitR with extitC++: A Brief Introduction to extitRcpp. *PeerJ Preprints*, 5:e3188v1.

Eddelbuettel, D. and François, R. (2011). Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18.

Eddelbuettel, D. and Sanderson, C. (2014). Rcpparmadillo: Accelerating r with high-performance c++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063.

Jones, M. (2017). *Quick Intro to Parallel Computing in R*. `https://nceas.github.io/oss-lessons/parallel-computing-in-r/parallel-computing-in-r.html`. [Online; accessed 10-May-2019].

Kessels, R., Goos, P., and Vandebroek, M. (2006). A comparison of criteria to design efficient choice experiments. *Journal of Marketing Research*, 43(3):409–419.

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

Mersmann, O. (2018). *microbenchmark: Accurate Timing Functions*. R package version 1.4-4.

Meyer, R. K. and Nachtsheim, C. J. (1995). The Coordinate-Exchange Algorithm for Constructing Exact Optimal Experimental Designs. *Technometrics*, 37(1):60–69.

R-core (2018). *Package 'parallel'*. `https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf`. [Online; accessed 10-May-2019].

R Core Team (2019). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

RStudio (2019). *Profvis — Interactive Visualizations for Profiling R Code*. `https://rstudio.github.io/profvis/`. [Online; accessed 10-May-2019].

Sanderson, C. and Curtin, R. (2016). Armadillo: a template-based C++ library for linear algebra. *Journal of Open Source Software*, 1:26.

Sanderson, C. and Curtin, R. (2018). A User-Friendly Hybrid Sparse Matrix Class in C++. *Lecture Notes in Computer Science (LNCS)*, 10931:422–430.

Tian, T. and Yang, M. (2017). Efficiency of the coordinate-exchange algorithm in constructing exact optimal discrete choice experiments. *Journal of statistical theory and practice*, 11:254–268.

Train, K. E. (2009). *Discrete Choice Methods with Simulation*. Cambridge University Press, Cambridge.

Wickham, H. (2011). testthat: Get started with testing. *The R Journal*, 3:5–10.

Wickham, H. (2015a). *Advanced R*. CRC Press, Boca Raton, FL, 2 edition.

Wickham, H. (2015b). *R Packages*. O'Reilly Media, Inc., 1st edition.

Yu, J., Goos, P., and 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.

Leuven Statistics Research Centre (LStat)
Celestijnenlaan 200 B bus 5307
3001 HEVERLEE, BELGIUM
tel. +32 16 32 88 75
www.kuleuven.be