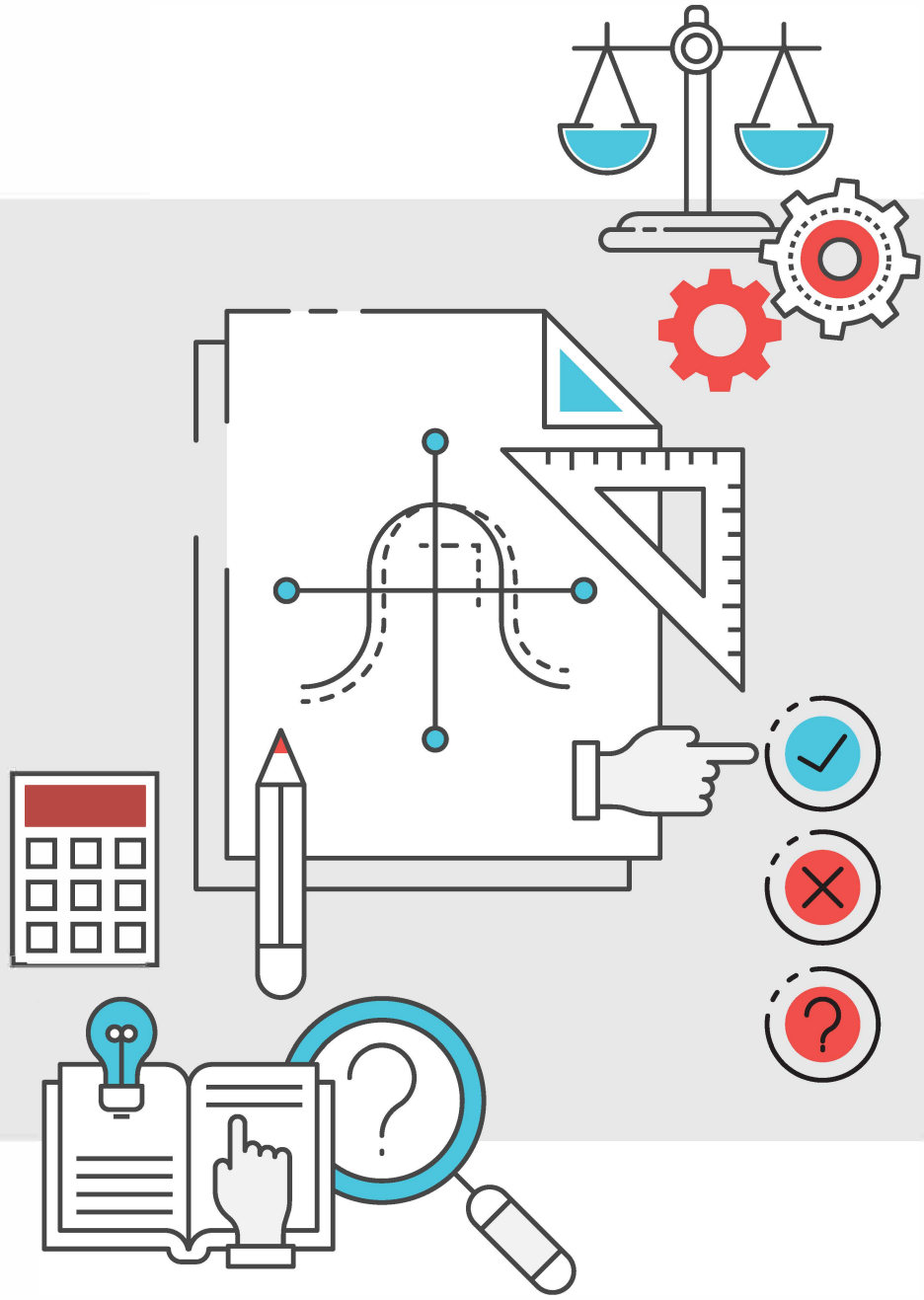
Ngene

MANUAL



ChoiceMetrics

Last updated: July 29, 2025

Contents

1 Designing and conducting choice experiments **10**

1.1 Step I: Labelled versus unlabelled experiments 13

1.2 Step II: Determine alternatives and attributes 17

1.3 Step III: Determine attribute levels and their coding 19

1.4 Step IV: Determine experimental design size 22

1.5 Step V: Choose experimental design strategy 24

1.5.1 Efficient designs 25

1.5.2 Orthogonal designs 28

1.5.3 Random designs 30

1.5.4 Agent- or segment-specific designs 31

1.6 Step VI: Conduct pre-testing and pilot-testing 32

1.7 Step VII: Conduct main study 37

1.8 Further remarks 39

2 Ngene user interface **41**

2.1 Main screen and project screen 41

2.2 Writing and editing scripts 42

2.3 Running scripts 43

2.4 Inspecting results 47

2.5 Exporting and importing designs 50

3 The basics of writing scripts **54**

3.1 Introduction to Ngene scripts and syntax 54

3.2 Defining alternatives 56

3.3 Defining design size and blocking 58

3.4 Defining design strategy 59

3.5 Generating full factorial designs 59

3.6 Generating random designs . . . 60
3.7 Defining model utility functions . . . 60
3.8 Defining attribute level preference order . . . 66
3.9 Interaction effects . . . 68

4 Orthogonal designs **72**
4.1 Orthogonal design generation . . . 72
4.2 Existence of orthogonal designs . . . 77
4.3 Blocking of orthogonal designs . . . 77
4.4 Interactions in orthogonal designs . . . 77
4.5 Limitations of orthogonal designs . . . 81

5 Efficient designs **82**
5.1 Efficient design generation . . . 82
5.1.1 Model type . . . 83
5.1.2 Efficiency criteria . . . 83
5.1.3 Efficiency statistic . . . 84
5.1.4 Decision rule . . . 85
5.2 Specifying noninformative priors . . . 86
5.3 Specifying informative local priors . . . 89
5.4 Algorithms to generate efficient designs . . . 93
5.5 Specifying Bayesian priors . . . 97
5.6 Specifying functions of attributes . . . 99
5.7 Designs for mixed logit models . . . 100
5.7.1 Random parameter models . . . 101
5.7.2 Error component models . . . 102
5.8 Evaluating or blocking existing designs . . . 102

6 Constrained choice tasks **105**
6.1 Strategies for applying constraints . . . 105
6.2 Logical expressions . . . 106
6.3 Conditional constraints . . . 106
6.4 Check constraints . . . 108
6.5 Status quo alternatives . . . 110
6.6 Scenario variables . . . 113

6.7 Attribute level overlap . . . 117
6.8 Availability of alternatives . . . 120
6.9 External candidate sets . . . 121

7 Multiple model specifications **125**
7.1 Different model specifications . . . 125
7.1.1 Different utility specifications . . . 126
7.1.2 Different model types . . . 126
7.1.3 Different alternatives . . . 127
7.2 Auxiliary model specification . . . 129

8 Considering population segments **133**
8.1 Defining population segments . . . 133
8.2 Homogeneous designs . . . 134
8.3 Heterogeneous designs . . . 136

9 Agent-specific attribute levels **139**
9.1 Homogeneous design . . . 139
9.2 Heterogeneous design . . . 143
9.3 Library of designs . . . 143
9.4 Pivot designs . . . 146
9.4.1 Pivot designs optimised around a single set of reference levels . . . 149
9.4.2 Pivot designs optimised around multiple sets of reference levels . . . 151

References **155**

List of Figures

1.1 Laptop choice task . . . 11
1.2 Treatment choice task . . . 11
1.3 Laptop choice task with images . . . 12
1.4 Depiction of full and fractional factorial designs . . . 14
1.5 Algorithms to find efficient designs . . . 28

2.1 Main screen . . . 41
2.2 New project screen . . . 42
2.3 Demo project . . . 43
2.4 Syntax help . . . 44
2.5 Running a script . . . 45
2.6 Search graph . . . 46
2.7 Design results . . . 47
2.8 More extensive design results depending on script . . . 48
2.9 Inspecting choice tasks . . . 49
2.10 Formatting options for choice tasks . . . 50
2.11 Choice task with alternative-specific attributes . . . 51
2.12 Import design . . . 51
2.13 Exported design in Excel . . . 52

3.1 Laptop choice task with identical profiles . . . 57
3.2 Two choice tasks that capture the same information . . . 57
3.3 Treatment choice task with identical profiles . . . 58
3.4 Random design and graphical depiction . . . 61
3.5 Laptop choice task with strictly dominant alternative . . . 67

4.1 Orthogonal design and graphical depiction . . . 74
4.2 Choice task for labelled car purchase example . . . 75

4.3 Choice task for unlabelled car purchase example 75

5.1 Efficient design and graphical depiction 87

6.1 Mode choice tasks with varying scenarios 114
6.2 Treatment choice task with multiple scenario variables 115
6.3 Laptop choice task with 2 overlapping attributes 118
6.4 Mode choice task with 3 available alternatives 121

7.1 Laptop choice task with unforced and forced choice 131

9.1 Query about distance class in survey 140
9.2 First choice task for different distance classes 142
9.3 Query about recent trip characteristics in survey 146
9.4 Choice task with attribute levels pivoted around reported reference levels 150

List of Tables

1.1 Data types and measurement scales 20
1.2 Attributes in laptop choice example 21
1.3 Types of priors and examples 28
1.4 Optimal orthogonal design 33
1.5 Attribute level balanced D-efficient design based on noninformative local priors . . 35
1.6 D-efficient design based on noninformative local priors using modified Fedorov algorithm 35
1.7 D-efficient design based on informative local priors 37

2.1 Keyboard shortcuts in the script editor 44
2.2 Design status 46

4.1 Sequential optimal orthogonal design for unlabelled car purchase example 76
4.2 Sequential randomised orthogonal design for unlabelled car purchase example . . . 76
4.3 Blocked orthogonal experimental design 78
4.4 Blocked simultaneous orthogonal design for mode choice example 79
4.5 Orthogonal design with mirror-image foldover 80
4.6 Optimal orthogonal design for laptop choice example 81

5.1 D-efficient design based on noninformative local priors for laptop choice example . 88
5.2 D-efficient design for laptop choice example when all attributes are dummy coded . 89
5.3 D-efficient design based on informative local priors for laptop choice example . . . 90
5.4 Choice probabilities in D-efficient design for laptop choice example 91
5.5 D-efficient design based on informative local priors for mode choice example 92
5.6 Choice probabilities in D-efficient design for mode choice example 93
5.7 D-efficient design using the modified Fedorov algorithm for laptop choice example 95
5.8 D-efficient design using modified Fedorov algorithm with balanced attributes . . . 97

6.1 D-efficient design based with conditional constraints for laptop choice example . . 108

6.2 D-efficient design based with check constraints for laptop choice example 110
6.3 D-efficient design with status quo alternative for treatment choice example 112
6.4 D-efficient design with scenario variable for mode choice example 114
6.5 Explicit partial profile design with 2 overlapping attributes in each choice task . . . 118
6.6 Implicit partial profile design with 2 overlapping attributes in each choice task . . . 120
6.7 Partial choice set design with 3 available alternatives in each choice task 123

7.1 Efficient design for multiple model specifications 128
7.2 Efficient design for models with different alternatives 130

8.1 Homogeneous design optimised across multiple population segments 135
8.2 Heterogeneous design with multiple population segments 137

9.1 Predefined attribute levels for different distance classes 140
9.2 Relabelling attribute levels for specific distance classes 141
9.3 Homogeneous design for three distance classes 144
9.4 Heterogeneous design for three distance classes 145
9.5 Library of designs for three distance classes 148
9.6 Pivot design based on single set of reference levels 150
9.7 Attribute levels after applying pivots to reference levels in Figure **9.3** 150
9.8 Homogeneous pivot design based on multiple sets of reference levels 153
9.9 Heterogeneous pivot design based on multiple sets of reference levels 154

List of Scripts

1.1 Optimal orthogonal design . . . 33
1.2 D-efficient design without strictly dominant alternatives . . . 34
3.1 Example script . . . 55
3.2 Less readable script . . . 55
3.3 Full factorial design . . . 59
3.4 Random design . . . 62
3.5 Design with opt-out alternative . . . 65
3.6 Design for labelled experiment . . . 66
3.7 Indicating preference order to exclude strictly dominant alternatives . . . 68
3.8 No clear preference order for most attributes . . . 69
4.1 Orthogonal design . . . 73
4.2 Sequential orthogonal design . . . 75
4.3 Simultaneous orthogonal design with three blocks . . . 78
4.4 Orthogonal design cannot avoid strictly dominant alternatives . . . 81
5.1 Efficient design . . . 86
5.2 Efficient design for unlabelled experiment . . . 88
5.3 Efficient design for labelled experiment . . . 92
5.4 Using modified Fedorov algorithm . . . 95
5.5 Assuming Bayesian priors . . . 98
5.6 Design with functions of attributes . . . 100
5.7 Design for panel random parameter logit model . . . 101
5.8 Design for panel error component logit model . . . 103
6.1 Conditional constraints . . . 107
6.2 Check constraints . . . 109
6.3 Status quo alternative . . . 111
6.4 Specifying an identifiable model . . . 112
6.5 Scenario variable with default swapping algorithm . . . 113

6.6 Scenario variable with modified Fedorov algorithm . . . 115
6.7 Status quo alternative and multiple scenario variables . . . 116
6.8 Scenario variable in unlabelled experiment . . . 117
6.9 Design with two overlapping attributes . . . 119
6.10 Partial choice set design with three available alternatives . . . 122
7.1 Different utility function specifications . . . 127
7.2 Omitted attribute . . . 128
7.3 Different model types and prior types . . . 129
7.4 Different alternatives . . . 130
7.5 Unforced and forced choice . . . 131
7.6 Auxiliary model to define variables . . . 132
8.1 Homogeneous design with multiple population segments . . . 135
8.2 Design with more population segments . . . 138
9.1 Single design for all distance classes using design coding . . . 140
9.2 Homogeneous design with informative priors . . . 143
9.3 Library of designs . . . 147
9.4 Pivot design around single set of reference levels . . . 149
9.5 Pivot design without showing reference alternative . . . 151
9.6 Homogeneous pivot design around multiple sets of reference levels . . . 152

1

Designing and conducting choice experiments

This chapter describes the theory behind experimental design for choice experiments and the process of designing and conducting choice experiments. It is recommended that Ngene users first familiarise themselves with this theory, as well as with discrete choice theory in general, before designing choice experiments and generating experimental designs. Excellent books in discrete choice theory are Train (2009) and Hensher et al. (2015).

Let us first provide a definition of a choice experiment.

> **Choice experiment.** A stated preference method in which agents are asked to choose their preferred alternative (most or least) in a series of hypothetical choice tasks. Also referred to as *stated choice experiment*, *discrete choice experiment*, or *choice-based conjoint*.

Choice experiments have a long history in both academia and practice. Originally designed to empirically test a range of economic theories, such as the existence of indifference curves (Thurstone, 1931; Mosteller and Nogee, 1951; Rousseas and Hart, 1951; May, 1954; MacCrimmon and Toda, 1969), stated choice experiments have since gained widespread acceptance across a range of fields in applied economics, including transportation (e.g., Bliemer and Rose, 2011; Hess et al., 2020; Ortúzar et al., 2021), health (e.g., De Bekker-Grob et al., 2013; Determann et al., 2014; Hansen et al., 2019), marketing (e.g, He and Oppewal, 2018; Wu et al., 2019; Burke et al., 2020), and environmental and resource economics (e.g., Scarpa et al., 2003; MacDonald et al., 2011; Greiner et al., 2014). Despite their prevalence, the design and implementation of a choice experiment requires far more nuance than most other survey methods, insofar as the technique requires that the analyst provide respondents a detailed set of choice situations with which they are expected to interact and respond. A choice experiment therefore does not simply ask agents what they did in some real-life situation (such data are called revealed preference data) or how they feel about some statement (as with attitudinal-type questions), but rather creates hypothetical situations that agents are expected to react to. The purpose of this chapter is to describe the processes required to generate these hypothetical situations.

> **Agent.** An individual making decisions or a representative of an entity in charge of making decisions. Also referred to as *decision-maker*.

You are looking to buy a new laptop *for at home*. Which of the following laptops would you prefer?	
Laptop A	Laptop B
Intel Core i7 processor 512 GB hard-disk drive $2100	Intel Core i5 processor 256 GB hard-disk drive $1500
○	◉

Figure 1.1: Laptop choice task

Consider a *70 year old* patient with *advanced prostate cancer*. As his doctor, what treatment would you recommend?		
Radiotherapy	Surgery	Active surveillance
Low risk of permanent side effects	High risk of permanent side effects	No side effects
50% probability of curing patient	70% probability of curing patient	0% probabilty of curing patient
○	○	◉
◉	○	

Figure 1.2: Treatment choice task

Examples of agents are consumers choosing a product to purchase, patients choosing medication, travellers choosing a mode of transport, farmers indicating a preference for a certain environmental policy, company directors choosing a financial strategy, etc.

Choice task. A situation presented to an agent that describes a *choice scenario*, a *choice set*, and a mechanism to capture one or more choice responses.

Choice scenario. A description of the context in which an agent is making a choice.

Choice set. A finite collection of alternatives that an agent can choose from.

Examples of choice tasks are shown in Figures 1.1 and 1.2, where agents are asked to select their preferred option by radio buttons. Although instructing agents to select their preferred alternative is most common, there exist other *choice response mechanisms*. In addition to the single-response mechanism in Figure 1.3, a dual-response could be asked when a status quo or opt-out alternative is present in the choice set, see the choice task in Figure 1.2. This choice task first captures the unforced choice where the choice set includes the status quo alternative 'Active surveillance'. If the status quo alternative is chosen, it also asks the agent to make a forced choice (e.g., in case the patient insists on being treated) where the choice set excludes the status quo alternatives. Other multi-response mechanisms include instructing agents to select the best and worst alternative or to select the first, second, and third best alternative. In the remainder of this chapter, it is assumed that each choice task asks a single question that captures only the first-best choice.

Each alternative in the choice set is described by a profile.

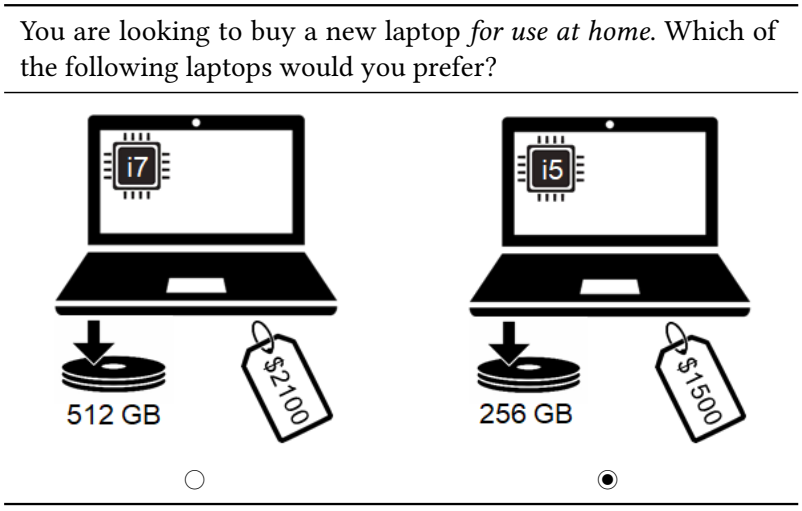


Figure 1.3: Laptop choice task with images

Profile. A complete representation of an alternative using specific levels of attributes.

Attribute. A quantitative or qualitative factor with which an alternative can be characterised.

Attribute level. A specific numerical or categorical value of an attribute.

For example, the price of the laptop is an attribute in the profiles shown in Figure 1.1, where \$2100 is a specific attribute level. Laptop A is characterised by a profile consisting of an Intel Core i7 processor, a 512 GB hard-disk drive, and a price of \$2100. Profiles vary from choice set to choice set, while the choice scenario is often fixed over choice sets but may also vary (e.g., the scenario in the choice task in Figure 1.2 may be varied to investigate the impact of patient age and illness type on choice behaviour). Although profiles are often shown in table format with text, different formats exist; see, for example, a graphical representation in Figure 1.3. Images may assist agents in imagining the alternatives and attributes, although one should be careful not to accidentally influence agents with factors that are present in the image but not the subject of study (e.g., symbolic colours or mood in a photo).

Choice experiments are usually part of a larger *questionnaire* or *survey* consisting of several parts. Although the survey flow differs from questionnaire to questionnaire, the first part of a survey typically involves agents being asked screening questions to judge their eligibility. In the second part, agents might be asked questions related to their current situation and behaviour related to the specific study. This information can be used to tailor choice tasks in a choice experiment in the third part of the survey. The fourth part typically concludes by asking additional questions, such as questions about general attitudes and perceptions, sociodemographic questions, and open-ended qualitative questions. Although sociodemographic questions could also be asked earlier in the survey, attitudinal questions should be asked after the choice experiment to avoid influencing choice behaviour (Liebe et al., 2021).

Designing and conducting choice experiments can be somewhat complex, consisting of several steps. The typical steps involved in designing a choice experiment are:

I. Determine whether an experiment is labelled or unlabelled depending on research questions;

II. Determine alternatives and attributes to include in the experiment;
III. Determine attribute levels and their coding;
IV. Determine experimental design size;
V. Choose experimental design strategy;
VI. Conduct pilot study;
VII. Conduct main study.

Each step is discussed in more detail in the following sections.

1.1 Step I: Labelled versus unlabelled experiments

The profiles of the alternatives shown in each choice task are based on an underlying full or fractional factorial experimental design.

> **Experimental design.** A matrix of attribute levels where each row contains a choice set with profiles that describe the alternatives. The number of rows represents the size of the collection of choice tasks from which the analyst can sample and give to an agent.

> **Full factorial design.** An experimental design that contains all possible choice tasks (that is, all possible combinations of attribute levels).

> **Fractional factorial design.** An experimental design that contains a subset of the full factorial.

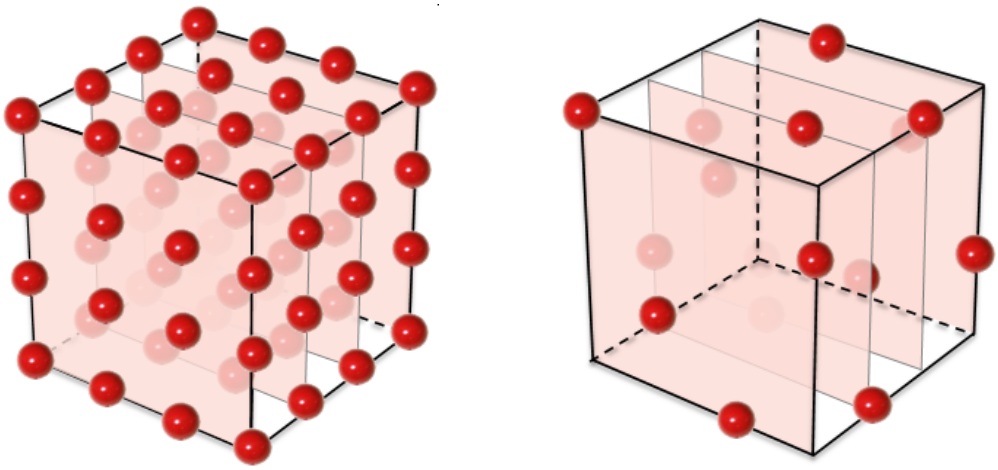
Let S denote the set of choice tasks where $|S|$ is referred to as *design size*. Assume that a subset of these choice tasks $S_n \subseteq S$ is given to agent $n \in \{1, \ldots, N\}$, where N is the *sample size* of the responding agents. In each choice task, agents are asked to choose among alternatives in the set J, where $|J|$ is the number of alternatives in the set of choices. Each choice task $s \in S_n$ is based on profiles for each alternative $j \in J$, described by attribute levels in the row vector $\mathbf{x}_{nsj}$. The experimental design matrix $\mathbf{X}$ contains $|S|$ rows where each row contains the profiles of all $|J|$ alternatives, $(\mathbf{x}_{ns1}, \ldots, \mathbf{x}_{ns|J|})$.

The size of a full factorial design is obtained by multiplying the number of attribute levels over all attributes in all alternatives. For example, suppose that there are two alternatives: the first alternative is characterised by three attributes, and the second alternative is an opt-out alternative without any attributes. If each attribute has four levels, then the total number of possible attribute level combinations is $4 \times 4 \times 4 = 64$, hence the full factorial consists of 64 unique choice tasks. A depiction of this full factorial design with 64 unique choice tasks is shown in Figure 1.4(a) where each side of the cube represents one of the attributes and each orb represents a unique choice task. Figure 1.4(b) shows an example of a fractional factorial design with 16 choice tasks. The selection of these 16 choice tasks depends on the experimental design strategy (random, orthogonal, or efficient, see Section 1.5).

Alternatives in a choice set can be of the same type or of different types, commonly described by a label.

> **Label.** A descriptor that indicates the type of alternative.

A label can, for example, be a product category, a brand name, but can also refer to specific alternative types such as a status quo or opt-out alternative.



(a) Full factorial design (b) Fractional factorial design

Figure 1.4: Depiction of full and fractional factorial designs

Status quo alternative. An existing alternative described by a fixed profile.

Opt-out alternative. An alternative without a profile that represents the option to choose none of the alternatives presented. Also referred to as *no-choice alternative*.

The label of an alternative generally impacts choice behaviour because an agent may have positive or negative associations attached to that label. However, if all alternatives have the same label, then the label does not play a role in the choice process. Examples where alternatives have the same generic label can be found in route choice (Route A, Route B, Route C), medication choice (Medication 1, Medication 2), policy choice (Policy I, Policy II), laptop choice (Laptop 1, Laptop 2), etc. An example is shown in Figure 1.1, where the alternatives are the same type (i.e., laptops). Such an experiment is often referred to as an *unlabelled experiment*.

Unlabelled experiment. A choice experiment where all alternatives have the same *generic* label and choice would not be affected if labels were swapped between alternatives.

If some or all of the alternatives have different labels, then choice may be influenced by these labels. Examples can be found in mode choice (Car 1, Car 2, Train, Bus), treatment choice (Surgery, Radiation Therapy), smartphone choice (Apple iPhone, Samsung Galaxy, Google Pixel), policy choice (Current policy [status quo], Policy A, Policy B), activity choice (Activity 1, Activity 2, Neither [opt-out]), etc. An example is shown in Figure 1.2. This type of experiment is referred to as *labelled experiment*.

Labelled experiment. A choice experiment where some alternatives have different labels and choice would be affected if labels were swapped between alternatives.

Based on observations from labelled or unlabelled choice experiments, one can build and estimate choice models. When modelling choices, the analyst needs to decide which *decision rule* to adopt as the theoretical foundation of the model.

Decision rule. Criteria that agents are assumed to use when evaluating alternatives and selecting their preferred option from a choice set.

In this chapter, we focus on *random utility maximisation* (RUM) (McFadden, 1973) as the dominant decision rule considered in the choice modelling literature, although other decision rules such as *random regret minimisation* (RRM) have also been considered (Chorus, 2012; Van Cranenburgh and Collins, 2019). Under RUM, it is assumed that agents choose the alternative with the highest utility when faced with multiple choice options, while RRM assumes that agents aim to minimise regret after making a decision. Choice models based on RUM and RRM are similar in an unlabelled experiment with exactly two alternatives but are different otherwise. From here on, we consider RUM, which requires the specification of *utility functions*.

Utility function. A mathematical calculation that measures how much an agent values an alternative as defined by its profile.

The utility functions for all alternatives in an *unlabelled experiment* are identical, i.e.,

$$V_{nsj} = f\left(\mathbf{x}_{nsj}\right), \qquad \forall n \in \{1, \ldots, N\}, \forall s \in S_n, \forall j \in J, \tag{1.1}$$

where V_{nsj} is the systematic utility that agent n attaches to alternative $j \in J$ in choice task $s \in S_n$ depending on the profile of alternative j defined by the attribute levels in profile $\mathbf{x}_{nsj}$ and a generic function f. This function depends on a vector of unknown generic preference parameters $\boldsymbol{\beta}$ that describe trade-offs between attributes and attribute levels and are subject to estimation. Function f can be linear or nonlinear in the attributes, for example

$$f(x_1, x_2) = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2, \tag{1.2}$$

where the first two terms describe the *main effects* of the attributes and the third term describes the *interaction effects*. To make model estimation easier, it is common to assume that function f is linear in the parameters (e.g., one would avoid terms like $\beta_1\beta_2$) such that each parameter is associated with exactly one main or interaction effect.

While not relevant at the experimental design stage, in model estimation one would add constants in $|J| - 1$ alternatives to account for *presentation order effects* of alternatives, also known as *left-to-right bias* (in countries where one reads from left to right), where alternatives shown on the left (or top) in the survey may have a higher propensity of being chosen than alternatives shown on the right (or bottom) (see e.g., Ryan et al., 2018).

The utility function for each alternative in a *labelled experiment* can be different for each label $h \in H$,

$$V_{nsj} = f_h\left(\mathbf{x}_{nsj}\right), \qquad \forall n \in \{1, \ldots, N\}, \forall s \in S_n, \forall j \in J_h, \tag{1.3}$$

where $J_h \subset J$ is the subset of alternatives with label h, where $\sum_h |J_h| = |J|$ (since each alternative has a single label) and each alternative within this set has the same linear or nonlinear label-specific utility function f_h. These functions have preference parameters $\boldsymbol{\beta}_h$, which can be label-specific or generic across labels. The functions also include label-specific constants, where for identifiability purposes one of them needs to be normalised to zero for a chosen reference label. That is, one would estimate $|M| - 1$ label-specific constants. As an example, in the mode choice situation with alternatives Car 1, Car 2, Train, and Bus, one would specify four alternatives across three labels (Car, Train, Bus), where Car 1 and Car 2 have identical utility functions. With three labels, the model identifies two label-specific constants. Note that an opt-out alternative can only have a label-specific

constant (which may be normalised to zero), while a status quo alternative is described by a regular utility function using fixed attribute levels. In some experiments, the levels of the status quo may be agent-specific, taking on values captured earlier in the survey.

In contrast to estimating models using data from an unlabelled experiment, one cannot simply add alternative-specific constants to account for presentation-order effects of alternatives in a labelled experiment since such constants would be confounded with (some or all) label-specific constants. How to account for presentation order effects of alternatives in labelled experiments will be discussed in Step VI in Section **1.6**.

Whether a labelled or unlabelled experiment is suitable for a certain study depends on the research questions being addressed. If one is interested in determining the *willingness-to-pay* (WTP) for certain attribute levels or in determining the *relative importance of attributes* in decision making, then it often suffices to consider an unlabelled experiment in which two or more alternatives are shown as variants of the same label. On the other hand, a labelled experiment is suitable if one would like to determine *market shares* of a product type or *demand elasticities*. An opt-out option would be included if one is interested in predicting unconditional absolute demand in the market using unforced choice tasks, while it can be left out if one is only interested in relative market shares or conditional demand between products that ask for a forced choice (it is worthwhile noting that evidence suggests that for the same empirical context, the results one obtains from forced and unforced choice tasks can vary dramatically; see Dhar and Simonson, 2003). A status quo alternative is often added to determine willingness to deviate from an existing policy or simply to make the choice task look more familiar to agents. Labelled experiments can also be used to determine WTP values, particularly if the WTP values are expected to vary across different labelled alternatives (e.g., the willingness to pay for travel time savings differs for bus and car use). If, however, WTP values are expected to be the same across alternatives, then given that labelled experiments generally require more complex choice tasks and the estimation of a larger number of coefficients, there is no reason to use a labelled experiment if the sole purpose of the study is to determine WTP values.

Significant differences in the results of choice experiments have been found within the literature with and without the presence of status quo alternatives (see e.g., Dhar, 1997), the recommendation being in general that status quo alternatives should be used in such experiments where applicable (e.g., Adamowicz and Boxall, 2001; Bennett and Blamey, 2001; Bateman et al., 2002). Dhar (1997) found that the decision to defer the choice (and hence select an opt-out option) is influenced by the absolute difference in attractiveness among the alternatives. That is, the overall utility of the alternative is the main driver of selecting a no choice option as opposed to the complexity of attribute trade-offs necessary when choosing between different alternatives. Boxall et al. (2009) report similar findings to Dhar (1997), suggesting that increasing task complexity, related to how similar the alternatives are as described by the attribute levels shown, leads to increased choice of the status quo alternative, whilst at the same time, the age and level of education of a responding agent may also influence this choice.

Dhar and Simonson (2003) found that if a forced choice is followed by an unforced choice in a dual response task, then some alternatives tended to lose proportionally more share than others, violating the assumption of the independent and identically distributed (IID) model. As such, it may be necessary to estimate more sophisticated discrete choice models that relax the IID assumption when data are collected using both forced and unforced choice responses. Brazell et al. (2006) failed to locate IID violations in a similar experiment, hypothesising that the failure to detect such effects was likely the result of using a more complex choice experiment involving more attributes than was used by Dhar and Simonson (2003), concluding that the increased complexity of their design decreased the prevalence of possible compromise alternatives appearing within the experiment. Rose and Hess (2009) also explored the use of dual forced/unforced response mechanisms, however, unlike

the Dhar and Simonson (2003) and Brazell et al. (2006) studies, made use of respondent reported status quo alternatives as opposed to a simple no-choice alternative. Like Brazell et al. (2006), Rose and Hess (2009) found no evidence of IID violations between forced and unforced tasks. Rose and Hess (2009) also reported no differences between the WTP estimates obtained in the dual forced / unforced response data.

Kontoleon and Yabe (2003) compared a 'do not buy' response format to a 'buy / choose my current brand' format. Keeping everything else equal, they found that the relative choice share of the opt-out alternative was higher in the 'own brand' treatment as opposed to the treatment that received the 'no purchase' treatment. They further found differences in parameter estimates for the more important attributes, while little difference was observed for less salient attributes.

1.2 Step II: Determine alternatives and attributes

Once the study objectives are known and the choice of a labelled or unlabelled experiment has been made, the analyst must determine which alternatives and attributes to include in the choice experiment. This is different for each study and while for some studies determining the alternatives and attributes is straightforward, for other studies, it requires careful consideration of how the outcomes will be used.

For any experiment, the minimum number of alternatives shown in a choice task is two, that is, $|J| \geq 2$, one of which may be a status quo or opt-out alternative. The larger the number of alternatives, the more information is captured in each choice task, but also the larger the cognitive burden placed on the responding agent. In case of an unlabelled experiment, there is generally no need to go beyond two or three generic alternatives. If the number of attributes is small, then three or four alternatives may be fine, but with a large number of attributes, one typically restricts the number of alternatives to two. In case of a labelled experiment, the number of alternatives in each choice task depends on the number of relevant labels to include, since each label requires at least one alternative, i.e., $|J_m| \geq 1$, which means that the number of alternatives needs to be larger than or equal to the number of labels, $|J| \geq |M|$. For example, in a mode choice experiment, one may need to include labels for Car, Metro, Train, Bus, Bicycle, and Walk, such that the number of alternatives in a choice tasks is at least six. If there is a risk that a certain label is dominant, e.g., if some agents will always choose Car no matter what the attribute levels are, then one can consider including two Car alternatives, Car 1 and Car 2, to ensure that all agents make trade-offs across alternatives. If the number of labelled alternatives is considered too large, one could show only a subset of labelled alternatives in each choice task, a so-called *partial choice set* (Bliemer et al., 2018).

> **Partial choice set.** A choice set that only contains a subset of relevant (labelled) alternatives.

An experimental design with partial choice sets is referred to as a *partial choice set design* or an *availability design* whereby the implicit assumption is that the alternatives that are not shown in a choice task are simply not available. This reduces the complexity of each choice task and hence the cognitive burden on an agent, but requires an increase in the number of choice tasks per agent, or an increase in sample size, to capture the same amount of information.

Extensive research has been conducted on the impact of the number of alternatives shown in choice experiments. For example, Adamowicz et al. (2006) found that respondents assigned to a three-alternative version of a choice experiment were more likely to choose a status quo option than a two-alternative version. Rolfe and Bennett (2009) report similar findings when comparing two- and three-alternative versions of a choice experiment. Caussade et al. (2005) found that the number of alternatives shown to the respondent had the second largest influence on the error variances of

all the design dimensions they tested and concluded that showing four alternatives is better than showing three or five alternatives in terms of the impact of scale effects. DeShazo and Fermo (2002) found a quadratic relationship between the number of alternatives and the variance, suggesting that error variance first decreases and then increases with the number of alternatives. In contrast, Arentze et al. (2003) found no error variance differences between the versions of the choice experiments that used two versus three alternatives. Hensher (2004) found that as the number of alternatives increases, there exists a differential impact on the WTP measures for different attributes of the design, while Rose et al. (2009) found different impacts on the mean WTP estimates obtained from the same survey conducted in different countries. Using eye tracking technology, Meißner et al. (2020) report that respondents tend to increase the amount of information they process as the number of alternatives increases, while simultaneously filtering out more pieces of information when choice tasks include more alternatives. Interestingly, Meißner et al. found that the respondents almost immediately changed their adopted search strategies when the number of alternatives changes dramatically (say, from two to five alternatives) from one choice task to another. Weng et al. (2021) found differences in the WTP results obtained for an unlabelled choice experiment with two alternatives compared to one with more than two alternatives. They also found that the ability of agents to identify their preferred alternative improves for experiments consisting of a status quo and a single additional alternative as the number of attributes increases, but becomes harder when more alternatives are added.

With respect to attributes, if the objective of the study is to determine specific WTP estimates in an unlabelled experiment, one could simply include only the attributes under investigation. For example, it is common in transport to determine the value of travel time using only two attributes, namely travel time and travel cost (see e.g., Batley et al., 2019), although it is necessary to be careful to avoid endogeneity bias[1]. However, if the objective of the study is to forecast demand or market shares, you would generally include all the attributes that are deemed relevant to make the choice. Relevant attributes can be identified by reviewing the literature, conducting a series of qualitative interviews such as focus groups involving a small number of agents (typically less than ten) from the target population, or personal interviews with experts.

> **Focus group.** A qualitative research technique in which one asks a group of agents about their rationale for making decisions in the choice context of interest.

Focus groups are often held face-to-face, but can also be conducted online. Although focus groups may include individual tasks such as writing down the most relevant attributes and ranking them in order of importance, open-ended group discussions guided by a moderator are at the core. Group discussions allow participants to agree or disagree and provide a way to identify a range of opinions and experiences that would be difficult to obtain through surveys.

While considering only a small number of attributes assists in reducing cognitive burden on agents, it has been argued that relevance is more important than quantity. If a large number of attributes is deemed relevant, then one can consider showing only a subset of attributes in each choice task. Such an incomplete profile is typically referred to as a *partial profile* (see e.g., Chrzan, 2010; Kessels et al., 2011).

[1] Endogeneity bias may occur if the true decision calculus used by agents involves interactions between omitted attributes and attributes used as part of the study. For example, one agent may imagine travel time seated in an empty bus, while another may imagine travel time standing in a crowded bus, and hence attach more disutility to travel time. In this case, the omission of crowding as an attribute and its interaction with travel time results in endogeneity bias, invalidating the assumption that the error term is independent of the systematic component of utility.

Partial profile. A profile that contains only a subset of relevant attributes.

Showing partial profiles simplifies the choice task, but one will need to increase the number of choice tasks per agent, or increase the sample size, to ensure that the same amount of information is obtained. An experimental design that contains partial profiles is referred to as a *partial profile design*. Instead of completely omitting some attributes in a choice task, one could instead show full profiles whereby some attribute levels are overlapping across alternatives. This *attribute level overlap* also reduces the number of trade-offs an agent needs to make for each choice task. A design that is specifically designed to have a given number of overlapping attribute levels is sometimes referred to as an *overlap design* or an *explicit partial profile design*, whereby overlapping attribute levels are explicitly shown in contrast to an *implicit partial profile design* that omits attributes and implicitly assumes that they are overlapping (without showing the attribute level).

Research has tended to show that the number of attributes present within the experiment has an impact on the behavioural responses provided. Caussade et al. (2005) and DeShazo and Fermo (2002) report that the number of attributes has a significant impact on the error variance of the models estimated using the choice experiment data. DeShazo and Fermo (2002) found that, on average, an increase in the number of attributes leads to an increase in the variance of the error component in utility of choice experiments, while Caussade et al. (2005) concluded that the number of attributes used had the largest influence on error variances of all design dimensions. In a similar vein, Arentze et al. (2003) found that increasing the number of attributes from three to five led to increased error variances and parameter differences. In support of this argument, Green and Srinivasan (1990) argued that respondents are incapable of processing many attributes simultaneously and become tired, and consequently ignore or address attributes in random and uncontrolled ways, or tend to use heuristics that lead to biased preference measures. Hensher (2006) found that the number of attributes has a significant influence on parameter estimates and WTP measures, which was also confirmed by Rose et al. (2009) who found statistically significant differences in WTP measures as the number of attributes increases. Nevertheless, Rose et al. (2009) report directional differences in the mean WTP in data sets collected from different countries.

The number of alternatives and attributes shown in each choice task also depends on the survey instrument. When using a computer-aided personal interviewer (CAPI), one can generally present more complex choice tasks to each agent given that a personal interviewer can explain the choice task and answer any questions that the responding agent may have about what they are presented with. In case of a typical online survey, completed on a computer or smartphone, one would generally keep the number of alternatives and attributes shown in each choice task limited as agents may be less engaged with the experiment and therefore spent less time on each choice task.

1.3 Step III: Determine attribute levels and their coding

Attributes can be classified as *qualitative* (also known as *categorical*), or *quantitative* (also referred to as *numerical*), and can further be distinguished according to their measurement scale; see Table **1.1**.

Attributes with nominal or ordinal scale describe qualitative/categorical data. If an attribute has nominal scale, then its levels do not have a specific ordering, whereas an attribute with ordinal scale has levels that describe a certain order. Attributes with an interval or ratio scale describe quantitative/numerical data, which can be discrete or continuous. Such attributes have an order in which absolute differences between levels are meaningful, and attributes with a ratio scale also have an absolute zero point.

Data type	Measurement scale	Example attributes with example levels
Qualitative / Categorical	Nominal	Colour (red, blue, yellow, green, purple) Warranty (yes, no) Livestock (cattle, sheep, pigs, horses)
	Ordinal	Comfort (low, medium, high) Side-effects (none, moderate, severe) Education (primary, secondary, tertiary)
Quantitative / Numerical	Interval	Temperature (5°C, 10°C, 15°C) Time of day (9am, noon, 5pm, midnight) Elevation (200 m, 700 m, 1500 m)
	Ratio	Cost ($20, $30, $40, $50) Travel time (15 min, 20 min, 25 min) Distance (1 km, 2 km, 5 km, 10 km)

Table 1.1: Data types and measurement scales

The attribute levels in profile $\mathbf{x}_{nsj}$ for each alternative j shown to agent n in choice task s in experimental design matrix $\mathbf{X}$ can be expressed using *design coding* or *estimation coding*.

Design coding. A data coding scheme where attribute levels are represented by values $0, \ldots, L$, where L is the number of attribute levels.

Estimation coding. A data coding scheme in which attribute levels are converted to numerical values that are used to calculate utilities in model estimation.

Although design coding is useful in the experimental design phase, attribute levels need to be converted to meaningful values for model estimation. This estimation coding is needed when evaluating utility functions in Equations (1.1) and (1.3). Estimation coding is also required when generating efficient designs; see Section 1.5. Various estimation coding schemes exist for qualitative attributes, where *dummy coding* and *effects coding* are the most widely used schemes, but there are other schemes, such as *orthogonal polynomial coding*. In each of these coding schemes, an attribute with L levels is represented by $L-1$ variables in the utility function. Each of these coding schemes results in the same behavioural model and the same model fit, only the interpretation of the values of the associated parameters differs (Daly et al., 2016). Dummy coding is most easy to interpret since each parameter corresponding to a dummy coded attribute level corresponds to the contribution to utility of this level relative to a chosen base level. The interpretation of parameters when using effects coding is similar except that they express differences with the average utility contribution instead of differences with the base level. When using orthogonal polynomial coding, the parameters describe the relationship between the attribute levels and utility in terms of linear effects, quadratic effects, cubic effects, etc.

Estimation coding for quantitative attributes is typically based on the actual numerical values of the attribute levels and can enter the utility function as a continuous linear effect, e.g., βx, or a non-linear effect, e.g., $\beta \ln(x)$ or βx^2. The unit in which a quantitative attribute is expressed can be chosen by the analyst and has no influence on the behavioural model (a parameter associated with an attribute measured in hours will simply be 60 times larger than a parameter associated with the same attribute measured in minutes, resulting in the same utility). Although it is possible to use dummy, effects, or (orthogonal) contrast coding for quantitative attributes using discrete levels, this

Attribute	Level	Preference order	Design coding	Estimation coding	
Processor	Intel Core i3	3	0	1	0
	Intel Core i5	2	1	0	1
	Intel Core i7	1	2	0	0
Hard-disk storage	256 GB	3	0	1	0
	512 GB	2	1	0	1
	1 TB	1	2	0	0
Price	\$1500	1	0	1500	
	\$1800	2	1	1800	
	\$2100	3	2	2100	

Table 1.2: Attributes in laptop choice example

makes it more difficult to interpolate/extrapolate beyond these levels in forecasting. Nevertheless, in some fields of applied economics, such as marketing, it is common practice to do so.

Once the measurement scale of each attribute has been identified, the number of levels can be determined. For nominal attributes, one typically needs to include all relevant levels (which can be asked in a focus group discussion, see Step II described in Section 1.2). In case of an ordinal attribute, one can often choose the number of levels, for example 'quality' can be described as low–high, or as low–medium–high, or as low–medium–high–very high. In case of ordinal attributes, one may want to be careful not to cause ambiguity as different agents will understand something different with respect to 'medium quality'. If possible, it is best to describe these levels in terms of specific characteristics, e.g., in terms of durability or referring to standards.

For attributes with interval or ratio scale, the analyst has full flexibility in choosing the number of attribute levels. For estimating linear effects, two levels are sufficient; however, for nonlinear effects, one would need more than two levels. Using (orthogonal) polynomial functions, three levels would allow us to estimate linear and quadratic effects, while four levels would also allow us to estimate cubic effects. The attribute level range has a large influence on the reliability of the parameter estimates. In general, a wide attribute level range (e.g., \$10 to \$50) leads to smaller standard errors than a narrow range (e.g., \$25 to \$30), but one should always make sure that the attribute levels are realistic and appropriate relative to other attributes. Furthermore, in choosing the exact values of the quantitative levels, one should prefer rounded values (e.g., \$5, \$10) over values that increase cognitive burden (e.g., \$4.75, \$9.90). Finally, one generally prefers equidistance attribute levels that cover the range equally (e.g. \$5, \$10, \$15) over levels that are not equidistant (e.g., \$5, \$8, \$15), unless the latter provides a more realistic representation of an attribute.

As an example, consider an unlabelled laptop choice experiment with three attributes, namely processor, hard-disk storage, and price. Each attribute is assumed to have three levels, as given in Table 1.2. The processor is measured on an ordinal scale, while hard-disk storage and price have a ratio measurement scale. The levels have a clear preference order, where 1 is the most preferred level and 3 is the least preferred level. This ordering allows us to assess whether there exists a strictly dominant alternative in a choice task. The last two columns in Table 1.2 illustrate the difference between design coding and estimation coding, where dummy coding is used, selected for the qualitative variables assuming that the last level is the base level. If effects coding is preferred then coding $(0, 0)$ for the last level should be replaced by $(-1, -1)$.

Empirically, the number of attribute levels has been found to have a significant impact on the behavioural outcomes of choice experiments by several authors. Wittink et al. (1990) found that

adding an intermediate level to a two-level attribute resulted in increasing the relative importance of an attribute, and in a subsequent study, Wittink et al. (1992) found that the number of levels influences the relative importance of an attribute, an effect that was magnified in the presence of dominated alternatives. Van der Waerden et al. (2004) concluded that the number of attribute levels can influence choice outcomes, finding that the number of attribute levels present in an experiment influences the scale of utility. Hensher (2006) found mixed evidence that the number of attribute levels affects the probability that respondents ignore an attribute when completing the tasks of the choice experiment, affecting some but not all the attributes contained in the experiment. Caussade et al. (2005) report that the number of attribute levels employed has a statistically significant impact on the degree of error variance present within the data; however, they conclude that the impact is marginal, having the second-lowest effect out of all the design dimensions they varied. Rose et al. (2009) found that the number of attribute levels used has a significant impact on WTP estimates; however, these differences depend on the country the data were collected from. Meyerhoff et al. (2015) found the impact that the number of attributes, alternatives and choice tasks has on modelled outputs differs according to the socio-demographic profile of the agents, with the biggest impact being on the drop-out rate of the survey itself. Finally, Oehlmann et al. (2017) found that as the attribute level range increases, the probability of selecting a status quo alternative increases, likely due to signals sent to the respondents about the certainty in the options shown throughout the experiment.

A further dimension of experimental design that has received attention in the past is the effect that the attribute level range plays on behavioural responses. Meyer and Eagle (1982) and Eagle (1984) found that attributes with larger ranges produced larger effects than those with smaller relative ranges, all else being equal. Ohler et al. (2000) on the other hand, found that attribute range differences affect experimental results in terms of the complexity of functional forms, model fit, power to detect non-additivity, and between-subject response variability. No effect was found on model parameters, within-subject response variability, or error variance. In contrast to Ohler et al., Caussade et al. (2005) concluded that the attribute range significantly impacts upon error variances, and that changes to the range that attribute levels take had the third largest influence on the error variances out of all the design dimensions tested. Hensher (2004) found that increasing the range of attribute levels resulted in lower mean WTP values, while Rose et al. (2009) found significant impacts on WTP estimates given changes to attribute level ranges; however, the directions of the impacts varied across different data sets.

1.4 Step IV: Determine experimental design size

The minimum required experimental design size $|S|$ depends on the total number of parameters to estimate in the choice model. Let K denote the total number of parameters, including label-specific constants and coefficients of attributes that are dummy, effects, or contrast coded. There must be sufficient variation in the design matrix $\mathbf{X}$ to estimate these K parameters. When an agent makes a choice among $|J|$ alternatives in a certain choice task s, this provides the information that the chosen alternative is preferred over each of the other $|J|-1$ alternatives shown to the agent. In other words, a design $\mathbf{X}$ consisting of $|S|$ choice tasks provides $|S| \cdot (|J|-1)$ pieces of information. To be able to estimate K parameters, it must hold that $|S| \cdot (|J|-1) \geq K$, in other words, the minimum size of the design can be determined by finding the smallest integer $|S|$ that satisfies:

$$|S| \geq \frac{K}{|J|-1}. \tag{1.4}$$

The difference between the actual number of choice tasks in the design and the minimum required design size is called the degree of freedom.

As an example, consider the laptop choice example in Figure 1.1 with the two alternatives, that is, $|J| = 2$, each with three attributes and attribute levels as shown in Table 1.2. Assume that the processor attribute is dummy coded so that it has two associated parameters, whilst storage and price are assumed to be continuous variables, each with a single parameter such that $K = 4$. Then according to (1.4) it should hold that $|S| \geq 4$. Although a design matrix of size 4 would be sufficient, increasing the degrees of freedom (and hence increasing variety in the design data) is recommended to improve identification of the parameter estimates. It is often recommended to use a design size $|S|$ that is at least two or three times the minimum required size to have sufficient degrees of freedom.

In choosing $|S|$ one may also want to consider attribute level balance.

> **Attribute level balance.** An experimental design is *attribute level balanced* if each attribute level appears the same number of times across all choice tasks.

Considering three levels in our laptop choice example in Table 1.2, attribute level balance could be guaranteed if the design size is a multiple of three, i.e., 6, 9, 12, etc. If the price attribute has four levels, then attribute level balance would require $|S|$ to be divisible by three and four, that is, 12, 24, 36, etc. Attribute level balance is not a requirement, but a high degree of attribute level balance is often desirable to obtain a good coverage over the data space.

If each agent $n \in \{1, \ldots, N\}$ is shown the same choice tasks, that is, $S_n = S$ such that each agent is subject to all choice tasks in the design matrix, then $\mathbf{X}$ is referred to as a *homogeneous design*. If the number of choice tasks $|S|$ is too large to show a single agent, then one can move from a homogeneous design to a heterogeneous design (where $S_n \subset S$) by blocking the design.

> **Blocking.** A process to split an experimental design matrix into two or more (equal) subset of choice tasks, known as *blocks*, where each block preferably has a high degree of attribute level balance.

For example, suppose that $|S| = 24$ and one would like to block this design in B equal parts. Then the design can be divided into blocks of six choice tasks each if $B = 4$, or blocks of eight choice tasks each if $B = 3$, or blocks of 12 choice tasks each if $B = 2$. The best number of choice tasks to show to each agent, $|S_n| = |S|/B$, depends on the complexity of each choice task and how many the analyst believes that an agent can handle without significant fatigue (which is a bigger issue with online surveys than with face-to-face interviews). Each block essentially represents a different version of the choice experiment, in which agents are distributed among these blocks (as evenly as possible). Survey instruments for conducting choice experiments can often automatically assign blocks to agents from a given design matrix or can randomly select a subset of choice tasks; therefore, implementing a heterogeneous design is not necessarily complicated. Heterogeneous designs are generally considered a good choice because they provide more information (Sándor and Wedel, 2005), although a homogeneous design can be justified if the number of parameters to be estimated is small relative to the number of choice tasks (Kessels, 2016). Instead of first creating an explicit (large) design matrix, one can also generate random choice tasks on the fly for each agent n, in which case the design matrix $\mathbf{X}$ is *implicit*.

Mixed evidence exists as to the impact the number of choice tasks has empirically upon choice experiments. Caussade et al. (2005) and Hensher (2004, 2006) found that the number of choice tasks acts on the error variance of discrete choice models; however, the effects reported by both Caussade et al. (2005) and Hensher (2004) were only marginal. Interestingly, Caussade et al. (2005), keeping the choice context constant while systematically varying all possible design dimensions in a sample of respondents, found that the number of choice tasks a respondent saw had the least influence of any of the design dimensions on the error variance of the choice data. Brazell and

Louviere (1996), keeping all other design dimensions constant, varied only the number of choice tasks shown to each respondent to be between 16 and 120. In their study, they found evidence of learning and fatigue effects, however they concluded that there exist no significant differences in either internal reliability or model variability for models estimated from survey questionnaires with varying numbers of choice tasks. Likewise, Hensher et al. (2001) reported finding that increasing the number of choice tasks had only a marginal impact on model elasticities; however, differences in elasticities were observed when agents were presented with 24 and 32 choice tasks compared to fewer. Hensher et al. recommend using more than four choice tasks, with 16 being sufficient for most modelling efforts. Bech et al. (2011) found only minor impacts on the mean WTP estimates obtained from choice experiments with different numbers of choice tasks, while Rose et al. (2009) found mixed evidence for the impacts of the number of choice tasks on WTP estimates, with differences observed between different countries. In this later study, the authors found that the number of choice tasks had almost no impact on a data set collected within an Australian context, a limited impact on the same survey collected in Taiwan and a very large impact using the same survey in Chile. More recently, Czajkowski et al. (2014) report that many observed discrepancies in modelled outcomes over choice tasks can be mitigated if error variance differences are properly taken into account, while Campbell et al. (2015) found that failing to account for the effects of learning and fatigue present within the choice data can significantly affect the WTP outputs. Oehlmann et al. (2017) report that all else being equal, increasing the number of choice tasks increases the probability that a status quo alternative will be chosen. Finally, Oehlmann et al. (2017) recommend that, everything else being equal, between 10 and 15 choice tasks is optimal in practice.

1.5 Step V: Choose experimental design strategy

In this section, we consider design strategies for determining a fractional factorial design (since a full factorial design is generally too large or undesirable). We assume that the aim is to determine a design matrix $\mathbf{X}$ for the estimation of a conditional logit model, also referred to in the literature as a multinomial logit model[2], which is the workhorse of discrete choice models. Choice probabilities in the conditional logit model are given by (McFadden, 1973)

$$p_{nsj} = \frac{\exp(V_{nsj})}{\sum_{j \in J} \exp(V_{nsi})}, \tag{1.5}$$

where utilities V_{nsj} are computed using generic or label-specific functions (**1.1**) or (**1.3**) where profiles $\mathbf{x}_{nsj}$ are represented using estimation coding. The Fisher information matrix for the conditional logit model is a $K \times K$ matrix $\mathbf{F}$ that can be computed as (McFadden, 1973)

$$\mathbf{F} = \sum_{n=1}^{N} \sum_{s \in S_n} \sum_{j \in J} \left(\mathbf{x}_{nsj} - \bar{\mathbf{x}}_{ns}\right)' p_{nsj} \left(\mathbf{x}_{nsj} - \bar{\mathbf{x}}_{ns}\right), \qquad \text{with} \quad \bar{\mathbf{x}}_{ns} = \sum_{i \in J} \mathbf{x}_{nsi} p_{nsi}. \tag{1.6}$$

Different types of choice models result in different matrices $\mathbf{F}$, for example, Sándor and Wedel (2002) derived the Fisher information matrix for the cross-sectional mixed logit model, Bliemer et al. (2009) for the nested logit model and Bliemer and Rose (2010) for the panel mixed logit model. It is possible to design data specifically around more advanced choice models, but this may come at a significant computational cost and may even be practically infeasible. Therefore, at the design stage it is common

[2]McFadden (1973) made a distinction between a multinomial model and a conditional logit model. In his definition, a multinomial logit model only contains variables related to the respondent (i.e., sociodemographics), whereas a conditional logit model only contains variables related to the alternatives (i.e., attributes). Therefore, according to these definitions, conditional logit is the appropriate term when we refer to data in a stated choice experiment. However, in practice, both sociodemographics and attributes appear in utility functions, and in the literature the term multinomial logit became the dominant term to indicate this type of model.

to design the data while having a conditional logit model in mind. Note that this generally does not prohibit the estimation of more advanced models at a later stage. As noted in Bliemer and Rose (2010), data that are designed to estimate a conditional logit model will generally also work well to estimate a panel mixed logit model.

The (asymptotic) variance-covariance matrix of parameter estimates, $\Omega = \text{var}(\hat{\boldsymbol{\beta}})$ is the inverse of the Fisher information matrix, i.e., $\Omega = \mathbf{F}^{-1}$. The diagonal elements of matrix Ω are directly related to the standard errors of the parameter estimates, namely, the standard error of parameter β_k equals $\sqrt{\Omega_{kk}}$ where Ω_{kk} is the k^{th} diagonal element of matrix Ω. A good design matrix $\mathbf{X}$ ensures that each parameter receives (non-zero) Fisher information such that they can all be estimated and that parameter estimates are reliable (i.e., small standard errors).

From Eqn. (1.6) we can make the following observations. First, Fisher information for the conditional logit model depends only on attribute levels and choice probabilities, not on choice observations, therefore, Fisher information can be determined based on experimental design $\mathbf{X}$ and best guesses of the choice probabilities for each alternative and each choice task. The same holds for the cross-sectional mixed logit model and the nested logit model, but the panel mixed logit model unfortunately requires simulated choice observations. Second, no Fisher information is obtained for choice tasks with a strictly dominant alternative (since Fisher information is zero if $p_{nsj} \rightarrow 1$ for a certain alternative j). Third, more Fisher information is obtained if the levels of quantitative attributes are further apart (wide range). And fourth, in the case of a homogeneous design where all agents face the same choice tasks, Fisher information increases linearly with the sample size N, which means that Ω is proportional to $1/N$ so that standard errors decrease at a rate of $\sqrt{N}$. And finally, Fisher information is generally reduced if there exists *overlap*.

> **Overlap.** The situation in which the levels of a generic attribute are the same for two or more alternatives in a choice task such that no trade-offs are made with respect to this attribute across these alternatives.

Although in principle more information is captured if overlap in the design is minimal, sometimes some overlap may be desirable. For example, to reduce the complexity of choice tasks using partial profiles (see Section 1.2) or when a dominant attribute level exists. To explain the latter, consider comparing two alternative laptops having brand as an attribute with two levels, Apple and Dell. Zero overlap means that agents are always forced to choose between a laptop of brand Apple and a laptop of brand Dell on all choice tasks. Depending on the agent's preference for an operating system (MacOS or Windows) they may always choose the alternative with a specific brand and ignore the other attributes. In this situation, it would be better to allow some overlap such that agents are also asked to choose between laptops of the same brand (thereby making trade-offs on the other attributes).

In this section, we discuss three main types of design strategies, namely *efficient designs*, *orthogonal designs*, and *random designs*, and we discuss the advantages and disadvantages of each strategy.

1.5.1 Efficient designs

Efficient designs have become the state-of-the-art in experimental design in the past decade. Let us first define efficiency.

> **Efficient design.** An experimental design is *efficient* if it captures a large amount of Fisher information. More Fisher information means more precise/reliable parameter estimates with the same sample size.

Since it is generally not possible to determine the most efficient design, the typical aim is to generate a design that is efficient without claiming that it is optimal. To maximise Fisher information, the volume of matrix $\mathbf{F}$ can be maximised, which is equal to minimising the volume of variance-covariance matrix $\boldsymbol{\Omega}$.

A $K \times K$ matrix can be represented as a hypercube in K dimensions. The lengths of the edges of a matrix are given by its eigenvalues $\boldsymbol{\lambda}$, where λ_k is the eigenvalue for dimension k that in matrix $\mathbf{F}$ corresponds to parameter β_k, $k \in \{1, \ldots, K\}$. The eigenvalues are determined via an eigen decomposition where matrix $\mathbf{F}$ is decomposed as $\mathbf{F} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^{-1}$ where $\mathbf{Q}$ is a matrix of eigenvectors that span the hypercube and $\boldsymbol{\Lambda} = \text{diag}\{\lambda_1, \ldots, \lambda_K\}$ is a diagonal matrix with eigenvalues of $\mathbf{F}$, and the volume can be calculated by multiplying the lengths of the edges of the hypercube. If $K = 2$ one multiplies the length and width to obtain the volume of a square, if $K = 3$ one multiplies the length, width and height to obtain the volume of a cube, etc. The volume of Fisher information is therefore given by the determinant of $\mathbf{F}$,

$$\det(\mathbf{F}) = \prod_{k=1}^{K} \lambda_k. \tag{1.7}$$

A related measure to the volume of Fisher information is the *D-error*, which is defined as the determinant of the variance-covariance matrix to the power $1/K$ to normalise the measure and account for the number of parameters,

$$\text{D-error} = \left(\det\left(\boldsymbol{\Omega}\right)\right)^{1/K} = \left(\frac{1}{\det(\mathbf{F})}\right)^{1/K}. \tag{1.8}$$

As a result, minimising the D-error equals maximising the volume of Fisher information. The literature commonly refers to D-efficient designs to indicate a low D-error. There does not exist a general threshold for a 'good' D-error value since this is case-specific and cannot be compared across studies, so all that can be said is that lower is better. It is generally also not possible to compute the lowest D-error value since this requires an exhaustive evaluation of all possible experimental designs, which is not practically feasible. To illustrate, consider our simple laptop choice example with two alternatives with the attribute and levels shown in Table **1.2**. This means that each alternative has $3^3 = 27$ unique profiles, such that there exist $27^2 = 729$ choice tasks in a full factorial design (although not all choice tasks would be sensible). Suppose that one is interested in determining the most efficient (fractional factorial) design consisting of six choice tasks. Choosing the best six choice tasks out of 729 possible choice tasks (without replacement) would require the evaluation of $729!/(729-6)! \approx 147,030,187,802,098,000$ unique designs, which would take even the fastest computer a very long time to complete.

To compute the efficiency of a design, utility functions need to be fully specified, including any interaction effects, nonlinearities, and estimation coding scheme (e.g., dummy coding). If an analyst tries to optimise the data for a choice model where one or more parameters are not identifiable (e.g., due to overspecification, due to lack of variation in attribute levels, or due to self-imposed multicollinearity via constraints), then the volume of Fisher information will be zero and the D-error will be infinite/undefined. Therefore, the D-error informs the analyst whether the model as specified can be estimated based on the specified attribute levels and constraints; a finite D-error (usually smaller than 1) gives confidence that the data can be used for model estimation.

In addition to D-efficient designs, other design types such as A-efficient designs (see e.g., Huber and Zwerina, 1996), or C-efficient designs exist (see e.g., Scarpa and Rose, 2008). An A-efficient design minimises A-error related to the circumference, instead of volume, of the Fisher information matrix, and a C-efficient design is used when optimisation of some function of parameters is of

interest, such as WTP estimates. Many other efficient design types exist (see Kessels et al., 2006), all measuring information in a slightly different way, but D-error is by far the most widely used information criterion and is recommended in most cases.

The main advantage of using an efficient design is that it captures (near) maximum information for a specific model, which means that it enables significant and/or reliable parameter estimates at smaller sample sizes that other design strategies. This makes efficient designs particularly useful if one is restricted either by budget or by a limited population of specific agents (e.g., pilots, physicians, patients with a certain disease, managers in a firm, etc.). In addition, efficient designs are very flexible and can be used in conjunction with various constraints on attribute levels (Collins et al., 2014), for example to avoid attribute levels that are unrealistic or impossible, and can avoid strictly dominant alternatives (Bliemer et al., 2017). The main disadvantages of an efficient design strategy are that efficient designs cannot be determined manually and require the use of optimisation algorithms, and that efficiency is sensitive to prior information about the expected choice probabilities in each choice task. To determine these expected choice probabilities, so-called priors are needed.

Different types of priors can be used to generate efficient designs. Although priors in experimental design have a somewhat different meaning than priors in Bayesian statistics, we use similar terminology to indicate the various types of priors. Two main types of prior can be distinguished, namely *noninformative priors* and *informative priors*.

Noninformative prior. An expression of the analyst's belief regarding the value of an unknown parameter value based on minimal information. This generally means assuming zero as a *local* prior or assuming a flat distribution around zero as a *Bayesian* prior. If knowledge of the sign of the parameter is available, then one can use a value close to zero with the correct sign or a flat distribution bounded by zero.

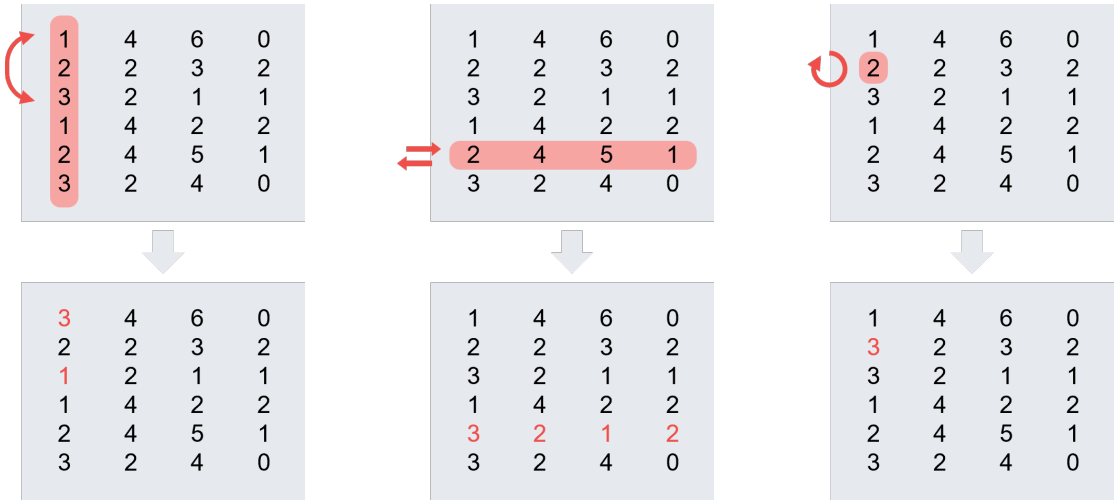
Informative prior. A best guess of an unknown parameter value based on information from a previous experiment (that is, a pilot study or a similar study described in the literature) or via subjective assessment of an expert. This generally means assuming the mean value provided by available evidence as a *local* prior or assuming a probability distribution (*Bayesian prior*) where the density is concentrated around this mean value.

Informative priors help to increase the efficiency of data collection, but may not be available. A pilot study is the best source for prior information; one should be careful adopting parameter values from other studies since parameter estimates may not be transferable due to scale, culture, and country effects. For information on expert judgement, we refer for example to Bliemer and Collins (2016). In practice, one would typically not mix informative and noninformative priors when generating an efficient design, but it is fine to mix local and Bayesian priors. Table **1.3** shows examples of the various types of priors. The more these priors (set when generating an efficient design) deviate from the true parameter values (obtained via model estimation after the data collection), the more efficiency will be lost. Choosing bad priors can also lead to *in*efficient designs (see for example the simulation study described in Walker et al., 2018), therefore choosing appropriate priors needs to be done deliberately, and if uncertain, it is best to choose noninformative (zero) priors or conservative (close to zero) priors.

Several algorithms exist to locate efficient designs. These algorithms minimise the D-error by modifying columns, rows, or cells within the experimental design matrix; see Figure **1.5**. A coordinate-exchange algorithm such as proposed by Meyer and Nachtsheim (1995) changes a single cell at the same is mainly useful for generating efficient designs without constraints. Column-based algorithms (e.g., Huber and Zwerina, 1996) are particularly useful for designs with attribute level balance

	Local	Bayesian
Informative priors	$\beta_k = -0.5$, $\beta_k = 0.8$, $\beta_k = 1.2$,	$\beta_k \sim \text{Normal}(-0.5, 0.2)$, $\beta_k \sim \text{Normal}(0.8, 0.5)$, $\beta_k \sim \text{Lognormal}(1.2, 0.9)$
Noninformative priors	$\beta_k = 0$, $\beta_k = -0.00001$, $\beta_k = 0.00001$,	$\beta_k \sim \text{Uniform}(-1, 1)$, $\beta_k \sim \text{Uniform}(-1, 0)$, $\beta_k \sim \text{Uniform}(0, 0.5)$

Table 1.3: Types of priors and examples



(a) Column-based (b) Row-based (c) Cell-based

Figure 1.5: Algorithms to find efficient designs

constraints as they swap attribute levels within a column, and row-based algorithms such as the modified Fedorov algorithm (Cook and Nachtsheim, 1980) select and exchange entire rows from a candidate set and are particularly useful for designs with many dominance constraints or other restrictions on attribute level combinations within a choice task.

Candidate set. A list of possible choice tasks from which a subset is chosen to form an experimental design. Such a *candidate set* could be a large random fractional factorial design or could be externally created by the analyst.

1.5.2 Orthogonal designs

Orthogonal designs have been used for choice experiments since the 1980s and have been the default design approach for several decades. Let us first define orthogonality.

Orthogonal design. An experimental design is *orthogonal* if for each two attributes, each pair of attribute levels appears equally across the choice tasks.

The above definition of orthogonality is based on *strength 2* orthogonality, where pairs of attribute levels across two attributes are considered. There also exist orthogonal designs with a higher strength where attribute level combinations across more than two attributes appear equally across the choice tasks, but such designs are rare. Based on this definition of orthogonality, attribute level balance

can be considered as strength 1 orthogonality. An orthogonal design matrix $\mathbf{X}$ is often referred to in statistics as an *orthogonal array*. If attributes have different numbers of levels, then such arrays are referred to as *mixed* orthogonal arrays, in contrast to conventional *fixed-level* orthogonal arrays (Hedayat et al., 1999).

The main advantages of orthogonal designs are that they cover the attribute space consistently, and no skills or running algorithms are required since they can be found in lookup tables in books (e.g., Hahn and Shapiro, 1967) or in online libraries (simply conduct a web-search for 'orthogonal array' to find the most recent sets of (mixed) orthogonal arrays as new arrays are being found and added over time). Furthermore, orthogonal arrays allow for blocking of the design matrix in such a way that it maintains perfect attribute level balance within each block. Several disadvantages of orthogonal designs exist. First, orthogonal arrays only exist for specific combinations of the number of attributes and attribute levels. If attributes have a varying number of levels where some have more than four levels, then an orthogonal array may not exist. Secondly, orthogonal arrays have a very rigid structure, which means that it is generally not possible to impose constraints on attribute levels or avoid strictly dominant alternatives. One could manually remove choice tasks from the orthogonal design that violate certain constraints or contain strictly dominant alternatives, but that would mean that the design is no longer orthogonal. Orthogonality is also lost in the data when considering interaction effects in the utility function that were not considered when locating an orthogonal array, when using dummy or effects coding, or when there are missing observations such as unequal representation of blocks in the data or unanswered choice tasks due to fatigue.

There exists a relationship between orthogonality and correlation. If the levels of two attributes are orthogonal, then they have zero correlation.

> **Attribute level correlation.** A measure between the levels of two or more attributes that indicates their level of dependence, where zero correlation means complete independence and a correlation of 1 or -1 means perfect (positive or negative) dependence.

It is easy to compute correlations between attribute levels and if the correlation in the levels across some attributes is not zero, then the design is not orthogonal. Although zero correlation is a necessary condition for orthogonality, it is not a sufficient condition. This means that zero correlation does not necessarily imply orthogonality.

Attribute levels in (fixed-level or mixed) orthogonal designs are uncorrelated; therefore, multicollinearity is avoided by definition. Independent estimation of parameters has often been claimed as a benefit of using orthogonal designs, but it should be noted that this benefit holds for estimating linear regression models and does not hold for estimating choice models. To explain this, we first introduce the concept of utility balance.

> **Utility balance.** A choice task is *utility balanced* if the utility of each alternative is the same, which implies that each alternative has the same choice probability. This is also referred to as *utility neutral*.

If the design matrix $\mathbf{X}$ is orthogonal and all choice tasks are utility balanced, i.e., $V_{nsj} \neq V_{nsi}$ for all alternatives $j \neq i$ such that $p_{nsj} = 1/|J|$, then $\mathbf{F}$ becomes a diagonal matrix. This implies that the variance-covariance matrix $\boldsymbol{\Omega}$ is also diagonal, which means that parameter estimates are uncorrelated and can be independently estimated. However, it is impossible in practice to satisfy both orthogonality and utility balance at the same time unless all parameters are equal to zero (which is typically the null hypothesis one wishes to reject). Hence, in practical applications it is not possible to independently estimate parameters in a choice model.

Street et al. (2001), Burgess and Street (2003), Street and Burgess (2004) and Street et al. (2005) introduced so-called *optimal* (orthogonal) designs in cases where alternatives have generic attributes (such as in unlabelled experiments). An optimal orthogonal design is a specific type of orthogonal design that seeks to maximise the Gramian matrix (which is an algebraic characterisation of the equivalent statistical Fisher information matrix, up to a scale) of the conditional logit model, thereby combining efficiency and orthogonality. Street et al. (2005) showed that generating such designs by hand is relatively easy using so-called *generators* that ensure minimum overlap in the design.

> **Generator.** A sequence of numbers using design coding that can be applied to the profiles of one alternative to generate profiles for another alternative.

The procedure described by Street et al. (2005) involves generating an orthogonal array (using design coding) to describe the profiles of the first alternative and then sequentially applying generators to create the profiles for the other alternatives. For example, consider an unlabelled experiment with three alternatives, each with three attributes having three levels as described in Table 1.2. Suppose that the generators 211 and 122 are applied to create the profiles for the second and third alternative. Consider profile 011 for the first alternative, which means a laptop with an Intel Core i3, 512 GB hard disk drive and a price of $1800. Then the profile for the second alternative becomes $011 + 211 = 222$ using element-wise addition, which means a laptop with an Intel Core i7, a 1 TB hard disk drive and a price of $2100. For the third alternative, the profile becomes $011 + 122 = 100$ where modulo 3 is applied to cycle back to level 0, that is, $(1 + 2) \bmod 3 = 0$. Application of a generator essentially re-assigns design codes to attribute levels, a process that is also referred to as *relabelling*. A generator of 122 means a relabelling of $0 \rightarrow 1$, $1 \rightarrow 2$, and $2 \rightarrow 0$ for the first attribute and a relabelling of $0 \rightarrow 2$, $1 \rightarrow 0$, and $2 \rightarrow 1$ for the second and third attribute. Since generators 211, 122, and 000 (associated with the first alternative) have no overlap, the resulting design will have zero overlap (and therefore optimal). Note that optimal orthogonal designs are also subject to the disadvantages mentioned above for orthogonal designs.

Under the (very strict) assumption of utility balance and assuming that all attributes are coded using orthogonal polynomial contrasts, it is possible to analytically compute the lowest possible D-error and therefore express D-efficiency as a percentage, where 100 percent indicates an optimal orthogonal design. These D-efficiency percentages are somewhat misleading since a 100% optimal design is unlikely to be optimal in practice, since the underlying assumptions are typically violated.

1.5.3 Random designs

Although efficient and orthogonal design strategies are systematic approaches in determining a fractional factorial design matrix $\mathbf{X}$ that contains a specific subset of choice tasks, an alternative strategy is simply to use randomly generated choice tasks for each agent by selecting choice tasks from an explicitly generated full factorial design (if the full factorial is sufficiently small), or by randomly generating choice tasks on the fly for each agent (if the full factorial is large). This experimental design strategy also allows for the application of constraints and can avoid strictly dominant alternatives. Random designs do not suffer from multicollinearity unless the analyst imposes constraints that perfectly correlate attribute levels.

As mentioned earlier, heterogeneous designs generally contain more information than a homogeneous design. A random design can be considered an extreme version of a heterogeneous design. While individual choice tasks in random designs may not capture a large amount of information, variation in the data is where random designs excel. The fact that each randomly generated choice task may capture different information allows random designs to decrease standard errors at a rate

larger than $\sqrt{N}$. Therefore, for a large enough sample size N, the amount of information captured with a random design can approach that of a fixed efficient design.

The main advantages of a random design strategy are that no experimental design skills are required (unless attribute level constraints or dominance checks need to be imposed), and the analyst does not need to formulate utility functions in advance since the data will be sufficiently rich to estimate any model. The main disadvantage is that it is an inefficient data collection strategy for small sample sizes and therefore should only be considered sample size is sufficiently large (typically at least 1,000 responding agents).

1.5.4 Agent- or segment-specific designs

Hypothetical bias is a well-known concern in choice experiments, e.g., due to the absence of consequences in hypothetical choice tasks or the difficulty in imagining alternatives that may not yet exist. We refer to Penn and Hu (2018) for a meta-analysis of hypothetical bias and to Haghani et al. (2021a) for an extensive overview of empirical evidence of hypothetical bias in choice experiments.

> **Hypothetical bias.** The deviation in estimated preferences due to choice data collected in hypothetical settings instead of a more realistic setting.

To make choices more realistic and incentive compatible, one could simulate experiences (e.g., Fayyaz et al., 2021) or introduce consequences (MacDonald et al., 2016). Several other methods exist to reduce hypothetical bias, including cheap talk, solemn oath, honesty priming, indirect questioning, time-to-think, and certainty scales; see Haghani et al. (2021b) for an overview.

Another way to reduce hypothetical bias in choice experiments is to create familiar choice tasks based on real experiences of agents instead of using a fixed design across the entire population (e.g., Hensher, 2010). This can be done through a so-called *pivot design* in which attribute levels are absolute or relative pivots around reference attribute levels reported previously by an agent (Rose et al., 2008). Another way is to create a *library of designs* containing separate designs for different agents; see, for example, Merkert et al. (2022). Both methods can be applied in conjunction with any experimental design strategy (efficient, orthogonal, or random) and are briefly explained below.

Using route choice as a common application in transport, consider asking agents about a recent trip they have made and wanting to tailor the choice tasks around their reported trips. An agent may report a recent trip to work by car that took 25 minutes and where \$5 toll was paid. Then in the choice experiment, the same agent would be asked to imagine making the same trip to work again and choose between two or more route alternatives where route travel times and toll costs vary around the reported travel time and toll cost. A pivot design is a fixed matrix $\mathbf{X}$ consisting of pivot levels. In case relative pivots are used, the matrix contains, for example, levels -25%, 0%, and +25%, which means that for this specific agent the levels shown in the choice tasks would be 25, 30, and 35 minutes for travel time and \$4, \$5, and \$6 for toll costs. Using relative pivot levels, attribute levels automatically scale to make sense for short and long trips. However, relative pivots do not always work; for example, if an agent reports to have paid \$0 in tolls, then the levels shown would be zero toll only. In such cases, one may want to revert to absolute levels, such as +\$1, +\$2, +\$3. Pivoting is generally not needed around qualitative attributes, but it is possible to pivot around attributes with an ordinal measurement scale by showing levels close to the reference input. Implementing a pivot design in a survey instrument typically requires programming rules and logic to deal with all kinds of user input, which may be challenging in certain survey tools.

An alternative to using a fixed pivot design is to generate different designs $\mathbf{X}^{(g)}$ for different groups of situations g, $g = 1, \ldots, G$, and have them available in a library within the survey instrument. In

our route choice experiment, we may, for example, create $G = 24$ different designs based on four categories of trips (work, business, shopping, leisure), two modes of transport (car, public transport), and three distance categories (short, medium, long). Using the same agent as described above, for this agent we would look up and use the design with characteristics 'work', 'car', and 'medium' from the library. The advantage of this approach is that all experimental designs can be generated and checked in advance, although it may require generating many experimental designs.

1.6 Step VI: Conduct pre-testing and pilot-testing

Once a draft survey has been developed, it needs to be pre-tested and pilot tested. This can be done qualitatively through focus groups or personal interviews and / or quantitatively through a pilot study (Mariel et al., 2021).

Pre-testing. A small-scale preliminary qualitative study to evaluate the clarity, validity, and reliability of the survey.

Pilot testing. A preliminary quantitative study to assess the feasibility of the survey and to improve the survey design for the main survey.

A *pre-test* aims to find out whether the information in the survey is sufficient and well understood by the target audience (using familiar concepts and terminology), noting that agents have different backgrounds and levels of education (Mariel et al., 2021). Pre-testing identifies and fixes problems or errors in the survey content, format, or administration. Johnston et al. (2017) recommends a minimum of four to six focus groups in survey pre-testing. Other ways to pre-test a survey is through face-to-face interviews or cognitive testing via think-aloud protocols.

A *pilot test* typically involves approximately 10 per cent of the total sample size (that is, $\frac{1}{10}N$) to verify that a choice model can be estimated before starting the main data collection. Pilot testing identifies potential problems throughout the entire survey procedure. One can ask agents for feedback about the survey, and the choice experiment in particular, at the end of the survey. This may include questions about the difficulty of choice tasks and how much they enjoyed it to get a sense of the complexity and engagement of choice tasks. A pilot study is also useful for obtaining parameter priors to generate a more efficient design for the main data collection, as further explained in Section **1.7**.

An efficient, orthogonal, or random design can be used for the pilot study. An orthogonal design could be useful if (i) most attributes have only two or three levels, (ii) if there is no real concern about strictly dominant alternatives (e.g., if the experiment is labelled with label-specific attributes, or if the attribute levels do not have an obvious preference ordering), and (iii) if there do not exist unrealistic attribute level combinations. In other cases, one could use an efficient design if sample size is small or a random design if sample size is large, while in both cases applying possible constraints and excluding choice tasks with strictly dominant alternatives. When using an efficient design in the pilot study, non-informative (zero) priors could be used to indicate that no prior information about the parameters is available.

As an example, Table **1.4** shows an optimal orthogonal design for our laptop choice example with two alternatives. This design was created by running script **1.1** in Ngene, which adopts the design generation technique proposed by Street et al. (2005). It can be seen in Table **1.4** that the generator 111 was used to create the profiles in the second alternative. Note that this syntax is provided merely for illustration purposes; explanations of the various parts of this syntax are given in subsequent chapters (although brief comments are included in the syntax).

```
design
;alts = LaptopA, LaptopB   ? two generic alternatives
;rows = 9                  ? design size of 9 choice tasks
;orth = ood                ? generate optimal orthogonal design
;model:                    ? using design coding for attribute levels
U(laptopA, laptopB) = proc * PROCESSOR[0,1,2]
                    + stor * STORAGE[0,1,2]
                    + cost * PRICE[0,1,2]
? PROCESSOR: 0(Core i3), 1(Core i5), 2(Core i7)
? STORAGE:   0(256 GB),  1(512 GB),  2(1 TB)
? PRICE:     0($1500),   1($1800),   2($2100)
$
```

Script 1.1: Optimal orthogonal design

	Laptop A			Laptop B		
Choice task	Processor	Storage	Price	Processor	Storage	Price
1	Core i5	256 GB	$1,500	Core i7	512 GB	$1,800
2	Core i7	512 GB	$1,500	Core i3	1 TB	$1,800
3	Core i3	1 TB	$1,500	Core i5	256 GB	$1,800
4	Core i7	256 GB	$1,800	Core i3	512 GB	$2,100
5	Core i3	512 GB	$1,800	Core i5	1 TB	$2,100
6	Core i5	1 TB	$1,800	Core i7	256 GB	$2,100
7	Core i3	256 GB	$2,100	Core i5	512 GB	$1,500
8	Core i5	512 GB	$2,100	Core i7	1 TB	$1,500
9	Core i7	1 TB	$2,100	Core i3	256 GB	$1,500

Table 1.4: Optimal orthogonal design

One can check that the attribute levels for Laptop A (and Laptop B) are orthogonal since each attribute level combination appears the same number of times, for example combination (Core i5, 256 GB) appears once, (1 TB, $1500) appears once, (Core i3, $2100) appears once, etc. This orthogonal design is optimal since there is no attribute level overlap, namely processor, amount of storage, and price are always different across the two alternatives. Despite being theoretically optimally efficient (under the assumptions of linear utility functions, orthogonality, orthogonal polynomial contrast coding, and utility balance or zero priors), it has two problematic choice tasks: Laptop B has a strictly dominant alternative in choice tasks 7 and 8 (indicated in red), where it is expected that Laptop B will always be chosen. We need to make a distinction between a dominance in probability (which can occur in labelled and unlabelled experiments) and dominance in attribute levels (which can only occur in unlabelled experiments).

Dominant alternative. An alternative that has a very high likelihood of being chosen. Also referred to as *dominance in probability*.

Strictly dominant alternative. An alternative that is better than (or equal to) any other alternative in the choice set with respect to all attributes. Also referred to as *dominance in attribute levels*.

```
design
;alts = (LaptopA, LaptopB) ? avoids strictly dominant alternatives
;rows = 9                  ? design size of 9 choice tasks
;eff = (mnl,d)             ? minimise D-error for the multinomial logit model
                           ? uses default column-based swapping algorithm
;model:                    ? uses noninformative priors to indicate preference order
                           ? uses estimation coding for attribute levels
U(laptopA, laptopB) = proc.dummy[+] * PROCESSOR[1,2,0]
                    + stor[+]       * STORAGE[5.545,6.238,6.931]
                    + cost[-]       * PRICE[1500,1800,2100]
? PROCESSOR: 0(Core i3), 1(Core i5), 2(Core i7)
? STORAGE:   ln(256), ln(512), ln(1024) GB
? PRICE:     $1500, $1800, $2100
$
```

Script 1.2: D-efficient design without strictly dominant alternatives

Although a strictly dominant alternative is also dominant, a dominant alternative is not necessarily strictly dominant. Choice tasks with a dominant alternative generally capture little information, but mainly choice tasks with strictly dominant alternatives are problematic, since they can bias parameter estimates (Bliemer et al., 2017). Choice tasks with a strictly dominant alternative can easily be identified by substituting attribute levels with their preference order according to Table 1.2, for example, Laptop A has attributes with preference orders (3,3,3) in choice task 7, while Laptop B has a profile with preference orders (2,2,1), making it better in each attribute. In case there is no clear preference order for attribute levels (e.g., colour of a car), then strictly dominant alternatives are generally not a concern.

Script 1.2 was used to generate an attribute level balanced D-efficient design assuming noninformative (zero) priors (i.e., utility balance) for the laptop choice example using the default swapping algorithm in Ngene where constraints have been applied to avoid strictly dominant alternatives. Again, this syntax is for illustration purposes only and will be explained in subsequent chapters.

Table 1.5 shows the resulting efficient design. For the computation of the D-errors, the following utility function was assumed:

$$f(\mathbf{x}) = \beta_1 x_{\text{processor}}^{(\text{Core i5})} + \beta_2 x_{\text{processor}}^{(\text{Core i7})} + \beta_3 \ln(x_{\text{storage}}) + \beta_4 x_{\text{price}}, \tag{1.9}$$

where $\ln(\cdot)$ indicates the natural logarithm, $x_{\text{processor}}^{(\text{Core i5})}$ and $x_{\text{processor}}^{(\text{Core i7})}$ are dummy-coded binary variables using level 'Core i3' as the base level, x_{storage} is the hard-disk storage in GB (i.e., 256, 512, 1024), x_{price} is the price in dollars, and $\boldsymbol{\beta} = (\beta_1, \beta_2, \beta_3, \beta_4)$ are parameters to be estimated. In this example, we have applied a transformation via the natural logarithm on the storage variable under the hypothesis that there is diminishing benefit in additional storage space (i.e., at some point enough is enough).

The D-error of the design in Table 1.5 for the above model specification is 0.0272, which is slightly better than the D-error of 0.0287 that would result from the design in Table 1.4 (which imposes orthogonality constraints but no dominance constraints) despite some overlap in the storage and price attribute. The efficiency of the design can be further improved by using the modified Fedorov algorithm, which relaxes the attribute level balance constraint. This requires adding the following command to Script 1.2:

```
;alg = mfedorov    ? uses row-based modified Fedorov algorithm
```

Table 1.6 shows this design, which is clearly not attribute level balanced. It has a low D-error of 0.0255 and does not have strictly dominant alternatives and does not have overlap in the attribute

	Laptop A			Laptop B		
Choice task	Processor	Storage	Price	Processor	Storage	Price
1	Core i7	1 TB	$2,100	Core i3	256 GB	$1,800
2	Core i3	256 GB	$1,500	Core i7	1 TB	$2,100
3	Core i7	512 GB	$1,500	Core i5	1 TB	$2,100
4	Core i5	1 TB	$1,500	Core i7	256 GB	$2,100
5	Core i3	1 TB	$1,800	Core i5	512 GB	$1,800
6	Core i3	512 GB	$2,100	Core i7	256 GB	$1,500
7	Core i7	512 GB	$1,800	Core i5	512 GB	$1,500
8	Core i5	256 GB	$2,100	Core i3	1 TB	$1,500
9	Core i5	256 GB	$1,800	Core i3	512 GB	$1,800

Table 1.5: Attribute level balanced D-efficient design based on noninformative local priors

	Laptop A			Laptop B		
Choice task	Processor	Storage	Price	Processor	Storage	Price
1	Core i5	256 GB	$2,100	Core i3	1 TB	$1,500
2	Core i5	1 TB	$1,500	Core i7	256 GB	$2,100
3	Core i5	1 TB	$2,100	Core i7	256 GB	$1,500
4	Core i5	256 GB	$1,500	Core i3	1 TB	$2,100
5	Core i3	256 GB	$1,500	Core i7	1 TB	$1,800
6	Core i3	256 GB	$1,500	Core i5	1 TB	$2,100
7	Core i7	256 GB	$1,500	Core i3	512 GB	$2,100
8	Core i7	256 GB	$2,100	Core i3	1 TB	$1,500
9	Core i5	256 GB	$1,500	Core i7	1 TB	$2,100

Table 1.6: D-efficient design based on noninformative local priors using modified Fedorov algorithm

levels. Dummy (or effects) coded attributes will generally show a high degree of attribute-level balance across the two alternatives, since a low representation of a certain level would not capture much information for the corresponding parameter and, therefore, lead to a high D-error. However, for other attributes, it is typically more efficient to show the most extreme levels (at least when assuming zero priors), as this increases the trade-offs made in each choice task and hence increasing Fisher information. As a result, the medium storage level of 512 GB and the middle price level of $1800 only appear once within the nine choice tasks. If the levels of hard-disk storage were dummy coded, then the middle level of 512 GB would appear more frequently.

After having generated an experimental design (or a library of multiple segment-specific designs), one needs to choose a survey instrument. For the pilot study, one can simply use a pen and paper questionnaire or an Excel spreadsheet (e.g., Black et al., 2005), but in most cases, one would implement the choice experiment in an online (for web-based surveys) or offline (for CAPI surveys) software tool that will also be used in the main study. Tools that support choice experiments include SurveyEngine, Confirmit, Nebu, and Qualtrics (with a choice-based conjoint add-on module). Most free online survey tools do not support choice experiments, but for simple choice experiments, one may use tricks such as creating multiple-choice questions with images that are screenshots of profiles or whole choice tasks. A recommended tool for conducting choice experiments is SurveyEngine, which is specifically designed for this purpose and has the additional advantage that it can easily use experimental designs generated by Ngene.

As mentioned in Step I (see Section **1.1**, for labelled experiments, it is important to randomise (across agents, not within an agent) the arrangement of labelled alternatives shown in choice tasks to be able to account for possible presentation order effects of alternatives (e.g., left-to-right bias). In model estimation, one would include a generic dummy coded variable in the utility functions of all alternatives that indicates the order in which the alternative appeared in the choice task (essentially making presentation order an 'attribute' of each alternative).

To account for presentation order effects of *attributes*, one may also want to randomise (again between agents, not within an agent) the order in which attributes are shown to respondents, as their relative position (e.g., top or bottom) may have a significant impact upon the behavioural responses of agents completed choice tasks, also referred to as. For example, Kjær et al. (2006) varied the location of the price attribute, presenting it as the first attribute or the last attribute shown in the task. They found that the order of the price attribute led to statistically significant differences in price sensitiveness; however, they concluded that attribute presentation order did not result in different decision rules being used by the sampled respondents. In an earlier study, Scott and Vick (1999) reversed the order in which attributes were shown to responding agents and found statistically significant evidence of an attribute ordering effect on model outcomes. On the other hand, Farrar and Ryan (1999) found no evidence of this when they swapped the first two attributes with the last two attributes. Likewise, Boyle and Özdemir (2009) suggest that it is not a forgone conclusion that the ordering of attributes will affect the choices and statistical results; it is likely to be a study-specific issue. More recently, Logar et al. (2020) found that attribute order had no significant impact on WTP estimates in standard models, but did significantly impact attribute non-attendance (e.g., people ignoring certain attributes when making their choices). Interestingly, Weller et al. (2014), who did not explore attribute order effects, found that other design dimensions had no impact on attribute non-attendance.

After the pilot study, the analyst would use the collected choice data to estimate a conditional logit model and verify that the model parameters can be estimated resulting in parameter estimates $\hat{\beta}_k$, $k = 1, \ldots, K$, with corresponding standard errors ς_k that indicate the precision (reliability) of the estimates. In the pilot study, it is likely that some or all parameters are not statistically significant given the relatively small sample size. For parameters that are statistically significant, one can check whether they have the expected signs (e.g., price or cost coefficients are expected to be negative). If some parameters have an unexpected sign when using an efficient design, then one may want to check for strong correlations between certain attributes in profiles. For example, if in our laptop choice experiment the price attribute is always high (low) when storage space is large (small), then the parameter for price may become positive if agents generally prefer to have a large hard-disk. This can be remedied by including profiles with a low price and large storage space or high price and small storage space (while at the same time avoiding that this alternative becomes dominant via trade-offs on other attributes) or using an orthogonal design (which avoids such correlations by definition but may suffer from strictly dominant alternatives).

Since parameter estimates themselves are difficult to assess, one often looks at marginal rates of substitution (MRS) between attributes, of which WTP is a special case. The MRS represents the amount of attribute l (i.e., the cost attribute in the case of WTP) that one has to give up for the gain of one additional unit of attribute k such that the utility remains the same. For example, in our laptop choice experiment with utility function (**1.9**) the WTP to have a Core i7 processor instead of a Core i3 processor equals $-\beta_2/\beta_4$ dollars, and the WTP for an increase in hard-disk storage is $-\left(\beta_3/x_{\text{storage}}\right)/\beta_4$ dollars per GB, based on an initial storage level x_{storage}.

	Laptop A			Laptop B		
Choice task	Processor	Storage	Price	Processor	Storage	Price
1	Core i5	1 TB	$1,500	Core i7	256 GB	$1,500
2	Core i7	256 GB	$1,800	Core i3	1 TB	$2,100
3	Core i5	1 TB	$1,800	Core i3	256 GB	$1,500
4	Core i5	256 GB	$1,800	Core i3	1 TB	$1,500
5	Core i5	256 GB	$1,800	Core i7	1 TB	$2,100
6	Core i7	1 TB	$2,100	Core i3	256 GB	$1,800
7	Core i3	1 TB	$1,800	Core i5	256 GB	$2,100
8	Core i7	256 GB	$1,500	Core i3	1 TB	$1,800
9	Core i5	256 GB	$1,500	Core i7	1 TB	$2,100

Table 1.7: D-efficient design based on informative local priors

1.7 Step VII: Conduct main study

The main study can use the experimental design for the choice experiment as used in the pilot study (possibly after making some minor changes). However, one could improve the efficiency of data collection by generating a new experimental design using information from the pilot study. In particular, the parameter values $\hat{\beta}_k$ estimated using data from the pilot study can replace the noninformative zero priors used previously. Using these non-zero parameter values as priors means that we can no longer assume utility balance (i.e., equal choice probabilities) but rather use choice probabilities that are expected to be closer to the truth. This results in a more accurate measure of Fisher information, thereby allowing an improved experimental design.

Suppose that the parameter estimates obtained through a pilot study for our laptop choice experiment are given by $\hat{\beta}_1 = 0.35$ and $\hat{\beta}_2 = 0.5$ (for the dummy coded processor attribute), $\hat{\beta}_3 = 0.6$ (for the logarithmic storage attribute), and $\hat{\beta}_4$ (for the price attribute). Using these values as informative local priors (instead of zeros) we can generate a new D-efficient design by updating the specification of the utility function in Script 1.2:

```
U(laptopA) = proc.dummy[0.35|0.5] * PROCESSOR[1,2,0]
           + stor[0.6]            * STORAGE[5.545,6.238,6.931]
           + cost[-0.004]         * PRICE[1500,1800,2100]
```

This generates the experimental design shown in Table 1.7, which has a D-error of 0.0413. It is important to emphasise that this D-error is not comparable to D-errors of designs that were generated under different prior assumptions such as the designs generated in the previous section using zero priors. If the informative local priors equal the true parameter values, then the design in Table 1.7 captures maximum information. We observe that the price levels for the two alternatives in Table 1.7 are more balanced than in Table 1.6. This is a direct effect of using informative local priors. Since a prior value -0.004 for the price parameter indicates that price is relatively important in choosing a laptop (see discussion below), making comparisons only between extreme price points $1,500 and $2,100 would often result in choice tasks where price dominates. In such cases, little to no trade-offs are made with respect to processor and storage, and hence little information is captured with respect to these two attributes. Therefore, using informative priors when generating a D-efficient design assists in ensuring that agents make trade-offs across all attributes, especially when one or more dominant attributes exist.

The *relative importance of each attribute* in the experimental design can be determined by looking at the relative impact each attribute has on utility (Orme, 2019). Considering again the laptop choice example and the priors given, the processor attribute contributes between 0 (Core i3) and 0.5 (Core i7) to utility, the storage attribute contributes between $0.6\ln(256) = 3.33$ and $0.6\ln(1024) = 4.16$ to utility, and the price attribute contributes between $-0.004 \cdot 1500 = -6$ and $-0.004 \cdot 2100 = -8.4$ to utility. Looking at the range in utility contribution, in absolute terms, the processor makes a maximum difference of 0.5, storage makes a maximum difference of 0.83, and the price makes a maximum difference of 2.4 in utility. Expressing this in percentages, the relative importance of processor, storage, and price is 13 percent, 22 percent, and 64 percent, respectively. These percentages are also referred to as *partworth utilities*. In other words, price is the most important attribute in the choice experiment. We point out that the assessment of attribute importance can*not* be based on the size of the corresponding parameter values, since the measurement scales and units of attributes are different.

While a D-efficient design based on informative local priors would be able to capture maximum information under ideal circumstances where prior assumptions are correct, such priors are in practice merely a best guess and will often be considerably different from the final parameter estimates, resulting in some loss of information. The more accurate the informative local priors, the less information is lost in the data collection. If the informative local priors turn out to be entirely different from the actual parameter values, then data collection can, in fact, become very inefficient (Walker et al., 2018). To make a D-efficient design more robust against prior misspecification, informative Bayesian priors have been proposed (Sándor and Wedel, 2001). A Bayesian prior is different from a local prior in that it does not consider a single value for the prior but rather considers a range of values via a predefined probability distribution. For example, if one believes that the parameter value for the price attribute in our laptop example lies somewhere between 0 and -0.008, then one could consider a Bayesian prior with a uniform distribution between the two values. In other words, Bayesian priors take into account the inherent unreliability about prior parameter values. The degree of unreliability of each prior can be obtained via standard errors of the parameter estimates in a pilot study. Assuming parameter estimate $\hat{\beta}_k$ and its corresponding standard error ς_k that indicates the degree of unreliability of the parameter estimate, a natural choice for a Bayesian prior is to assume a normal distribution with mean $\hat{\beta}_k$ and standard deviation ς_k.

The Bayesian D-error of a design indicates the expected (mean) D-error over the given prior distributions and can be computed via Monte Carlo simulation by taking quasi-random draws from the prior distributions (Bliemer et al., 2008). The number of draws required to obtain stable Bayesian D-error values increases exponentially with the number of Bayesian priors (e.g., 2^K or 3^K for distributions with small standard deviations, 4^K or more for distributions with large standard deviations). It is therefore recommended to keep the number of Bayesian priors limited (typically not more than eight to ten) and to use local priors for the remaining parameters (if any). In choosing which parameters to allocate a Bayesian prior, it is advised to give priority to attributes with a high relative importance, as they will have the largest influence on utility and therefore are most sensitive to prior misspecification.

Continuing our laptop choice example, assume that the previously mentioned parameter estimates have standard errors $\varsigma_1 = 0.2$ and $\varsigma_2 = 0.3$ (associated with the dummy coded processor parameters), $\varsigma_3 = 0.4$ (associated with the storage parameter), and $\varsigma_4 = 0.0025$ (associated with the price parameter). A Bayesian D-efficient design (not shown here) with a mean D-error of 0.0497 can be generated by replacing the utility specification in Script **1.2** with the following:

```
U(laptopA) = proc.dummy[(n,0.35,0.2)|(n,0.5,0.3)] * PROCESSOR[1,2,0]
           + stor[(n,0.6,0.4)]                    * STORAGE[5.545,6.238,6.931]
           + cost[(n,-0.004,0.0025)]              * PRICE[1500,1800,2100]
```

The Bayesian D-error will always be larger than the D-error of a design that is optimised using corresponding local priors, but the associated Bayesian D-efficient design will result in less loss of information when the true parameter values deviate from the informative local priors. Therefore, it is recommended to use a Bayesian D-efficient design as a more robust design strategy, despite the increase in mean D-error.

An often asked question is “What sample size do I need?" The answer is that this is case-specific, where in some studies only 50 agents are needed to get statistically significant and reliable parameter estimates, whilst in other studies possibly thousands of respondents are needed. If alternatives include attributes that are all highly important (such as the cost attribute in most studies), then all parameters can be estimated with a smaller sample size. In contrast, if most attributes are only marginally relevant in making a choice, then a large sample size will be required to obtain statistically significant parameter estimates. Some rules of thumb have been discussed in the literature; see (Rose and Bliemer, 2013) for an overview, but one can make some specific minimum required sample size calculations if informative parameter priors are available. Using parameter estimates $\hat{\beta}_k, k = 1, \ldots, K$ from a pilot study as informative local priors, we can compute the Fisher information matrix and the related variance-covariance matrix $\boldsymbol{\Omega}$. The minimum sample size N_k^* for parameter k, such that it can be estimated at a given level of statistically significance, can be computed as (Rose and Bliemer, 2013; De Bekker-Grob et al., 2015):

$$N_k^* = \left(\frac{t_{\alpha/2}}{\hat{\beta}_k}\right)^2 \Omega_{kk}, \tag{1.10}$$

where Ω_{kk} is the variance of parameter k and $t_{\alpha/2}$ indicates the (two-sided) t-value at the desired level of significance α (e.g., 1.96 if $\alpha = 0.05$). The values N_k^* are also referred to as *S-estimates*, and the minimum sample size N^* required to estimate all K parameters at a statistically significant level, i.e., $N^* = \max_k \left\{N_k^*\right\}$ is also known as *S-error* (Rose and Bliemer, 2013). Given that the above minimum sample size computations rely heavily on prior parameter values, they should only be used when using informative priors that are sufficiently reliable and should only be used as ballpark figures (e.g., whether one needs tens, hundreds, or thousands of respondents). Note that if a design is blocked, these minimum sample size estimates need to be multiplied by the number of blocks.

1.8 Further remarks

This chapter aimed to outline the steps to follow in generating a choice survey. Although each study will differ in terms of research objectives, empirical application area, and sampling requirements, following the seven steps outlined here represents current best practice for most choice studies. In any case, six of the seven steps are required to collect any choice data, with only the possibility of not conducting pre-testing and pilot testing being feasible. However, this does not mean that one should not undertake some form of pilot study, and indeed it is highly recommended to do so. Unfortunately, some applied economic fields are better at this than others.

Of the seven steps, most are fairly straightforward and easy to complete. Of course, given the range of possible applications that choice experiments can be applied to, the ease of generating a stated choice experiment should never be taken for granted. Additionally, individuals planning to conduct a choice experiment should possess a solid understanding of discrete choice methods, particularly regarding the correct specification of utility functions to ensure all parameters can be accurately identified. It is often easy to make what appear to be small innocuous mistakes that can have significant ramifications that only become apparent after the data have been collected. For example, in a model with a status quo alternative containing a qualitative attribute (dummy or effects-coded), it is important that the fixed attribute level of the status quo alternative also appears in one or more other alternatives to avoid identification issues in model estimation (Cooper et al., 2012). Any person

attempting to design stated choice experiments is encouraged to first properly immerse themselves within the greater literature to fully understand the subtle nuances of discrete choice modelling.

Based on the data collected from a choice experiment, the analyst will typically estimate a large number of discrete choice models, starting with the multinomial logit model where covariates such as socio-demographics can be added into the utility functions. When choice set size varies in the dataset one may want to test for heteroscedasticity by allowing scale differences (e.g., via a nested logit model or error component mixed logit model) and account for preference heterogeneity (e.g., via a random parameter mixed logit model or a latent class model). Since stated choice data typically captures multiple choice responses from a single agent, it is important to also account for the panel nature of the data in model estimation. Various specialised software tools are available to estimate discrete choice models. A widely used tool for model estimation is Nlogit, which is specifically designed to estimate a wide range of discrete choice models using syntax that is very similar to that of Ngene. Effective and adaptable open-source tools for estimating discrete choice models include Apollo, which uses R (Hess and Palma, 2019), and Biogeme, which uses Python (Bierlaire, 2020).

2

Ngene user interface

This chapter provides an overview of the Ngene graphical user interface. It discusses the main screen and shows how to create a new project, how to open existing projects, and how to open demonstration projects. In addition, it explains the script editor, how to run scripts, and how to inspect results.

2.1 Main screen and project screen

Figure 2.1 shows the Ngene main screen. Clicking on the Ngene logo on the top left always returns to this main screen. There are buttons on the top right for *Notifications*, *Support*, and *Account*.

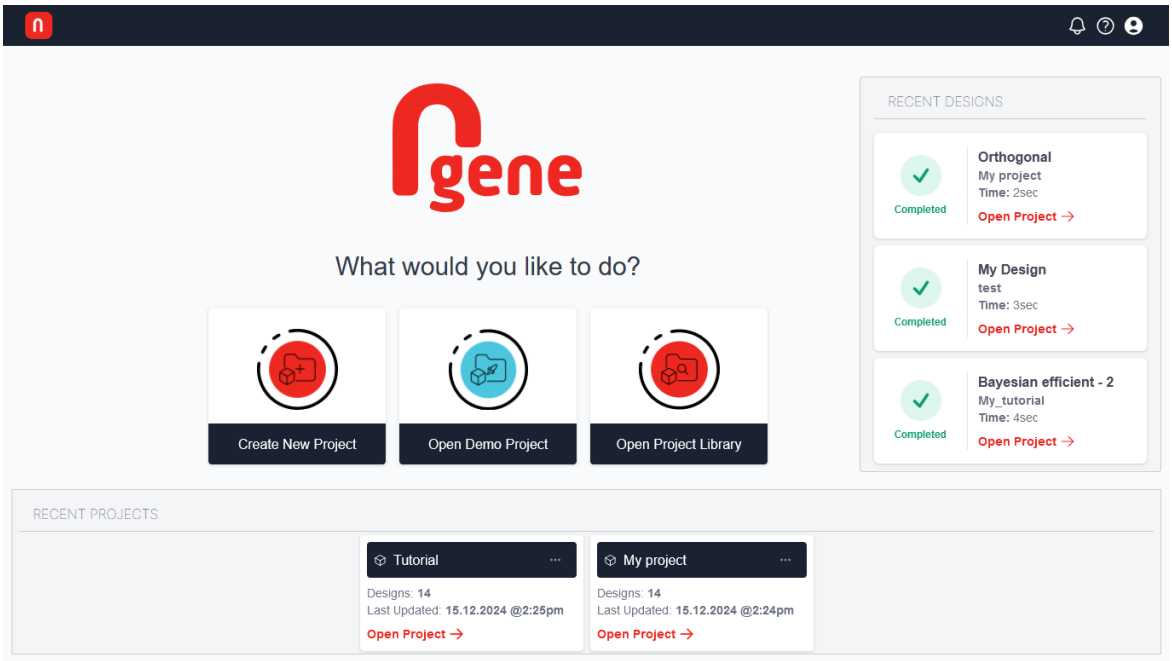


Figure 2.1: Main screen

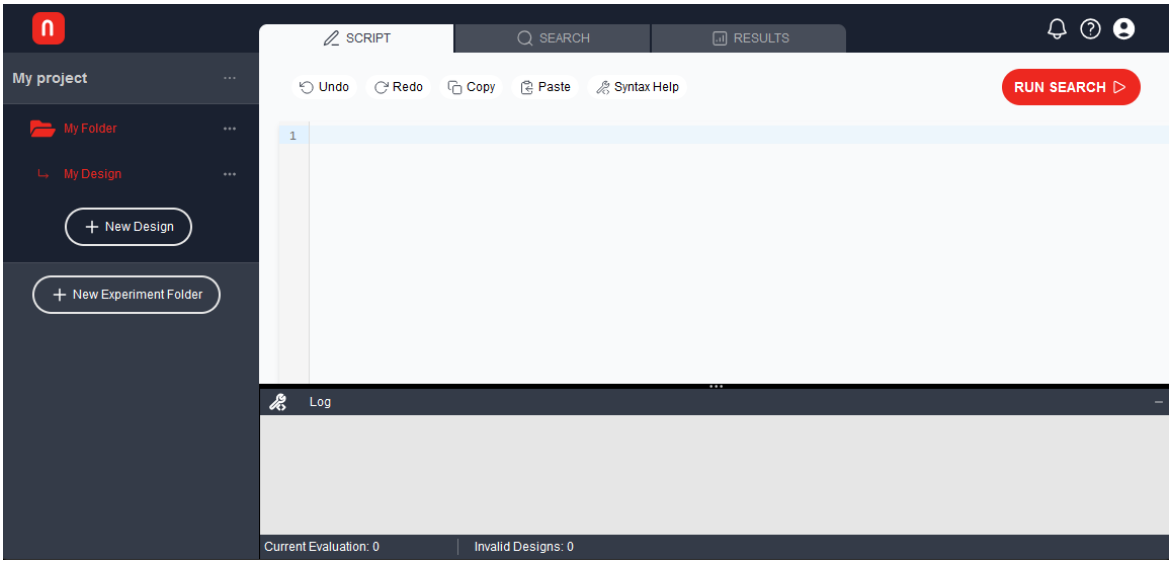


Figure 2.2: New project screen

A list of recently generated designs is presented on the right of the main screen is a, while on the bottom a list of recently opened projects appears. There are three main actions a user can take, namely *Create New Project*, *Open Demo Project*, and *Open Project Library*.

To create a new project, click the *Create New Project* button on the main screen. A pop-up window will ask you to name the project – for example, 'My project' – and then ask you whether to start with a *Blank script* or *Import design matrix*. Design import is discussed in Section 2.5. In most cases, you will want to start with a blank script. Figure 2.2 shows what the project screen looks like when starting with a blank script. By clicking on the ellipsis menu after the project name, indicated by three horizontal dots $(\cdots)$, it is possible to rename the project, close it, or delete it.

All experimental designs generated in Ngene are stored in experiment folders. *Folders* are useful for structuring a project. One could create, for example, separate folders for different phases in the study (e.g., pilot and main) or for different population segments. New folders can be added by clicking on the *New Experiment Folder* button. Folders can be opened and closed by clicking on the folder name. By clicking on its ellipsis menu, the folder can be renamed or deleted. Each folder can contain one or more designs. A *Design* can contain a script, a search graph, and results, which can be accessed via tabs at the top. To add a design, click on the *New Design* button. By clicking on its ellipsis menu, one can duplicate, rename, or delete it, or move the design to another folder.

Demonstration projects can be accessed by clicking on the *Open Demo Project* button on the main screen and then creating a copy of one of the available demo projects, see for example Figure 2.3. Existing projects can be opened by clicking on the *Open Project Library* button on the main screen. Recent projects can also be opened directly from the main screen.

2.2 Writing and editing scripts

Scripts to generate an experimental design can be written and edited in the *Script* tab in the project screen. Figure 2.3 shows an example of a script. The syntax for writing scripts is explained in Chapter 3 and further. The script editor uses syntax highlighting to indicate properties, reserved words, and comments. It also automatically adds closing brackets and parentheses.

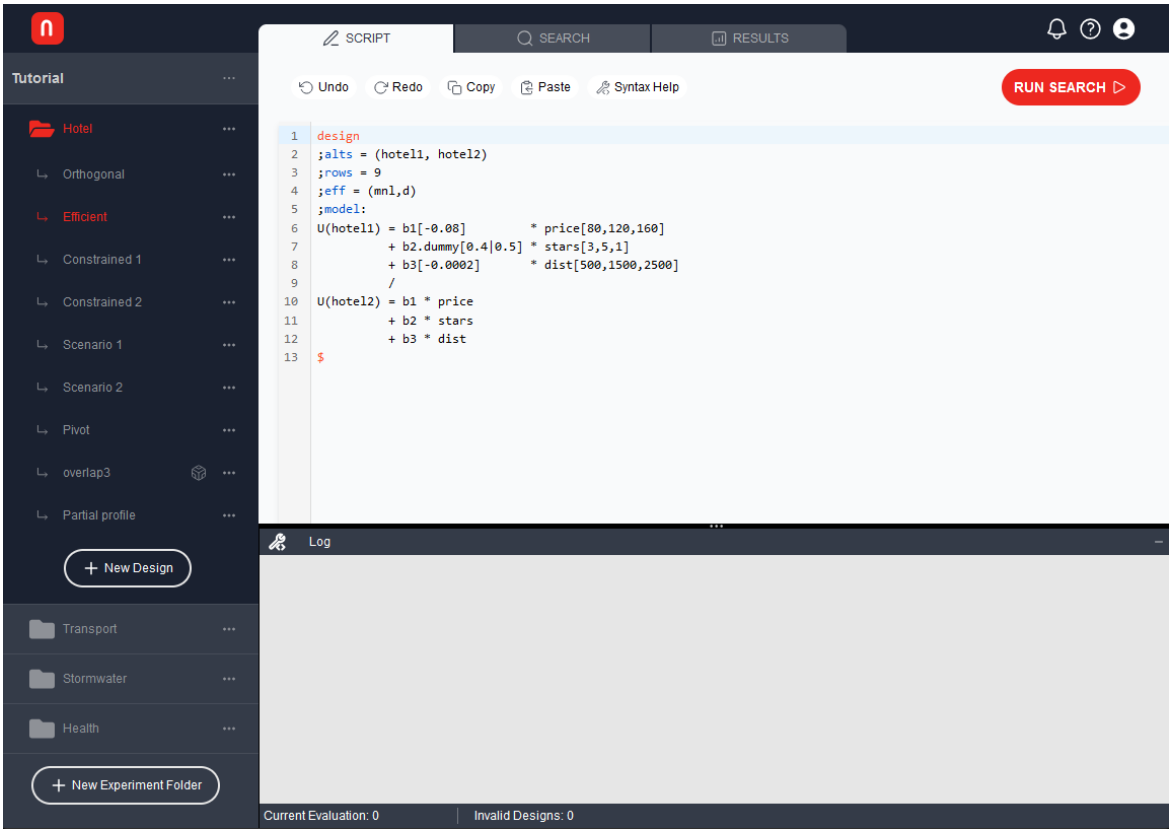


Figure 2.3: Demo project

Scripts are automatically saved, but the script editor has the *Undo* and *Redo* buttons above the script editor. There is also *Copy* and *Paste* functionality that can be accessed via buttons or by using typical keyboard shortcuts; see also Table 2.1. Text can be found (and replaced) using the keyboard shortcut Ctrl + F on a Windows computer or Cmd + F on a Mac. The script editor also supports multiple cursors to easily edit various parts of the script simultaneously. One can add another cursor by holding the Ctrl/Cmd button and clicking on another location in the script with the mouse, which is useful for adding or deleting text in multiple places. Similarly, to select multiple parts of the script, hold the Ctrl/Cmd button and select additional text with the mouse. Changing the name of an attribute in the entire script can be done with find and replace but also by selecting the attribute name (e.g., by double-clicking on the name) and then pressing Ctrl/Cmd + Shift + L to select all occurrences of this attribute name across the script. When multiple occurrences are selected, one can type and delete across multiple locations at the same time. A list of all keyboard shortcuts is shown in Table 2.1.

The syntax reference guide can be accessed by clicking the *Syntax Help* button. This Syntax Help pops up in a separate window and is useful for quickly looking up properties and syntax examples, see for example Figure 2.4.

2.3 Running scripts

After finishing writing a script, it can be run by clicking the *Run search* button on the upper right of the script editor, see, for example, Figure 2.3. The *Run search* button changes into a *Stop search* button, see Figure 2.5, and the Log screen underneath the script editor provides messages while the

Keyboard shortcut	Explanation
Basic editing	
Ctrl/Cmd + A	Select all.
Ctrl/Cmd + C	Copy.
Ctrl/Cmd + X	Cut.
Ctrl/Cmd + V	Paste.
Ctrl/Cmd + Z	Undo.
Ctrl/Cmd + Y	Redo.
Ctrl/Cmd + F	Find (and Replace).
Multi-cursor support	
Ctrl/Cmd + D	Select and add next occurrence.
Ctrl/Cmd + U	Undo latest cursor operation.
Ctrl/Cmd + Shift + L	Select all occurrences.
Ctrl/Cmd + Alt	Select multiple columns.

Table 2.1: Keyboard shortcuts in the script editor

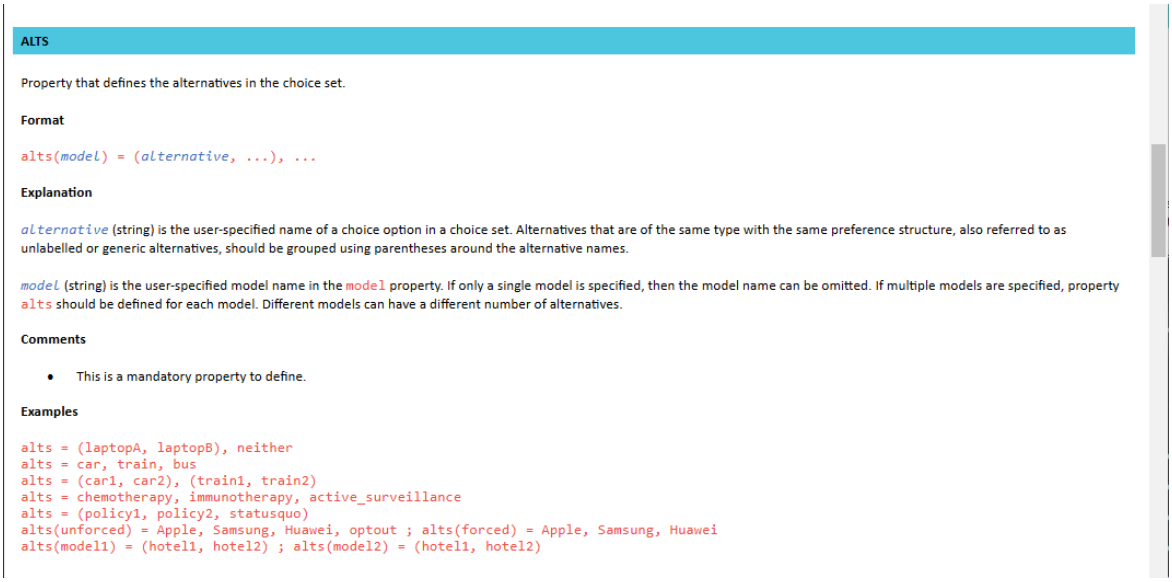


Figure 2.4: Syntax help

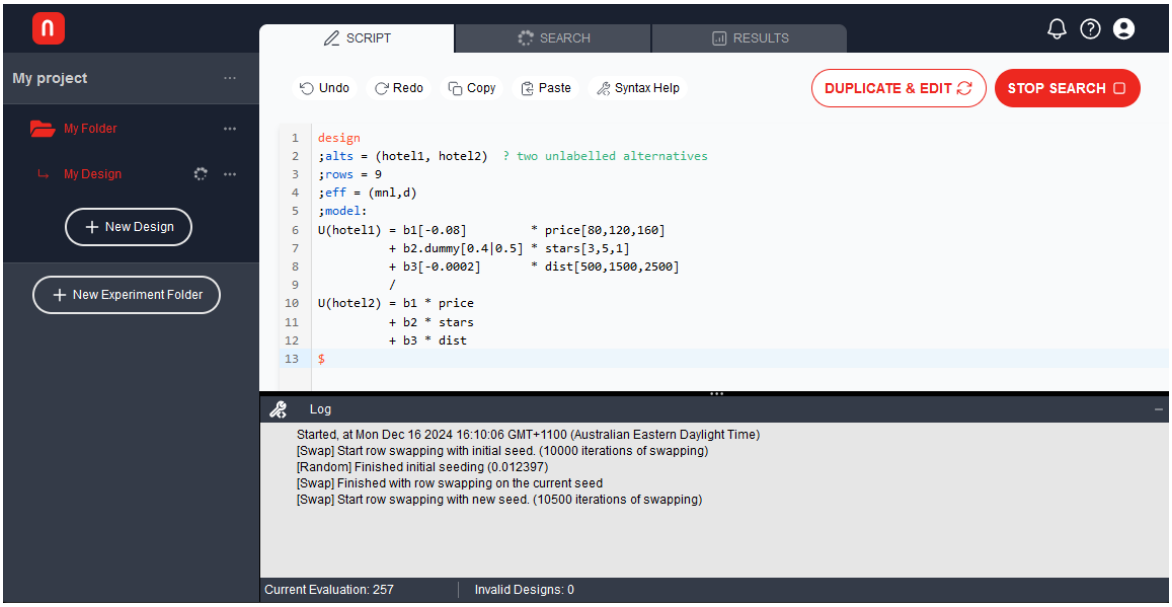


Figure 2.5: Running a script

script is running. In addition, at the bottom of the screen the *Current Evaluation* counter indicates how many experimental designs have been evaluated thus far.[1]

Some scripts will finish running within seconds, while other scripts may run for several minutes or longer. If the script does not automatically finish, it can be stopped manually once Ngene no longer generates designs that are substantially better (i.e., more efficient). This assessment can be made by looking at the graph in the *Search* tab, see Figure 2.6. In this example, a good design was generated after approximately 150 design evaluations, and designs found afterwards are only marginally better.

In almost all cases, scripts will start running immediately after clicking the Run search button. But occasionally, it may happen that the server is busy running many other scripts and that a script is put in the queue. In such cases, scripts will generally start running in one minute, but in rare cases it may take longer. We encourage all users not to let scripts run unnecessarily long so that other users can also run their scripts quickly. The search will typically automatically end if no better design is found after 5,000 consecutive design evaluations. All running scripts will time out upon reaching 10 hours of execution.

The status of each design is indicated with an icon. These icons are explained in Table 2.2. If there is an error in the script, more details about the error are provided in the log screen. The log screen may also display warnings. Warnings are not errors and will not prevent the script from running, but they often provide useful information and are worth taking note of.

After running a script, it becomes locked and can no longer be edited to ensure that no scripts or results are lost by mistake. To make changes to the script and run it again, simply click the *Duplicate & Edit* button, which creates a new design and copies the script. If the previous script / design is no longer needed, it can be deleted via its ellipsis menu.

[1] There is also a counter for *Invalid Designs*, which indicates the number of designs that were evaluated but rejected for various reasons (e.g., due to multi-collinearity or poor efficiency).

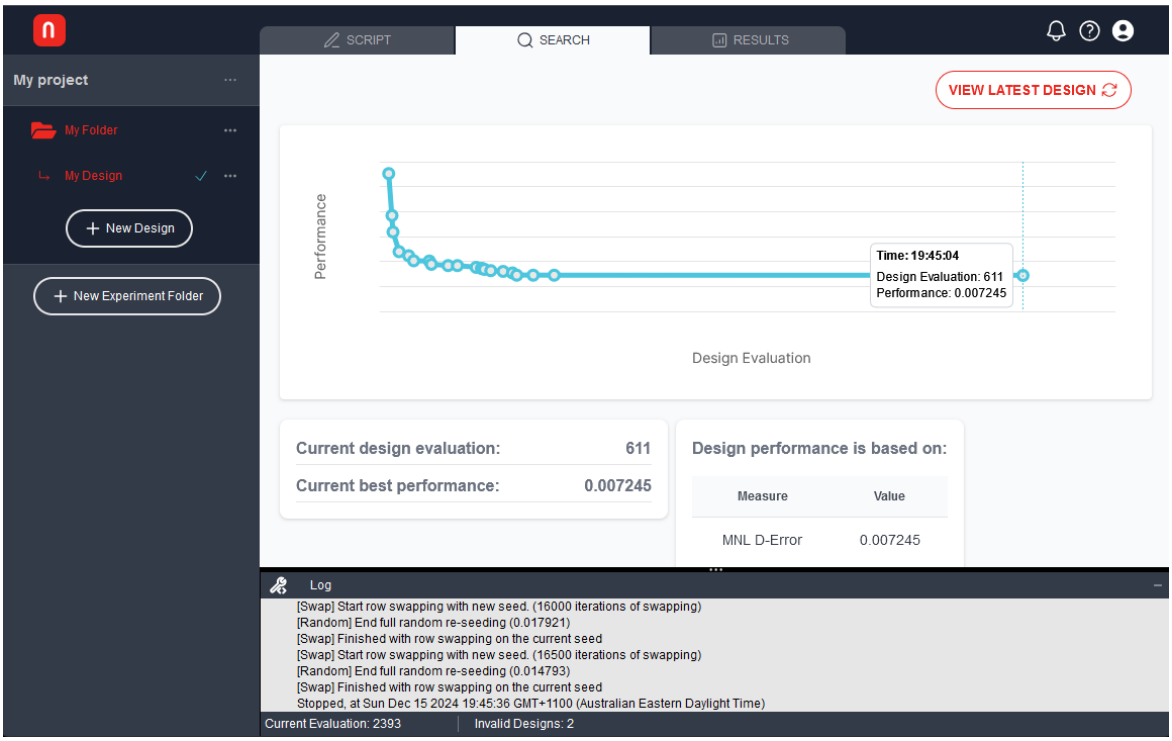


Figure 2.6: Search graph

Icon	Explanation
	Actively searching for designs.
	Finished design search.
	Favourite design (toggle via ellipsis menu).
	Error in script, no design generated.
	Imported design (no script).

Table 2.2: Design status

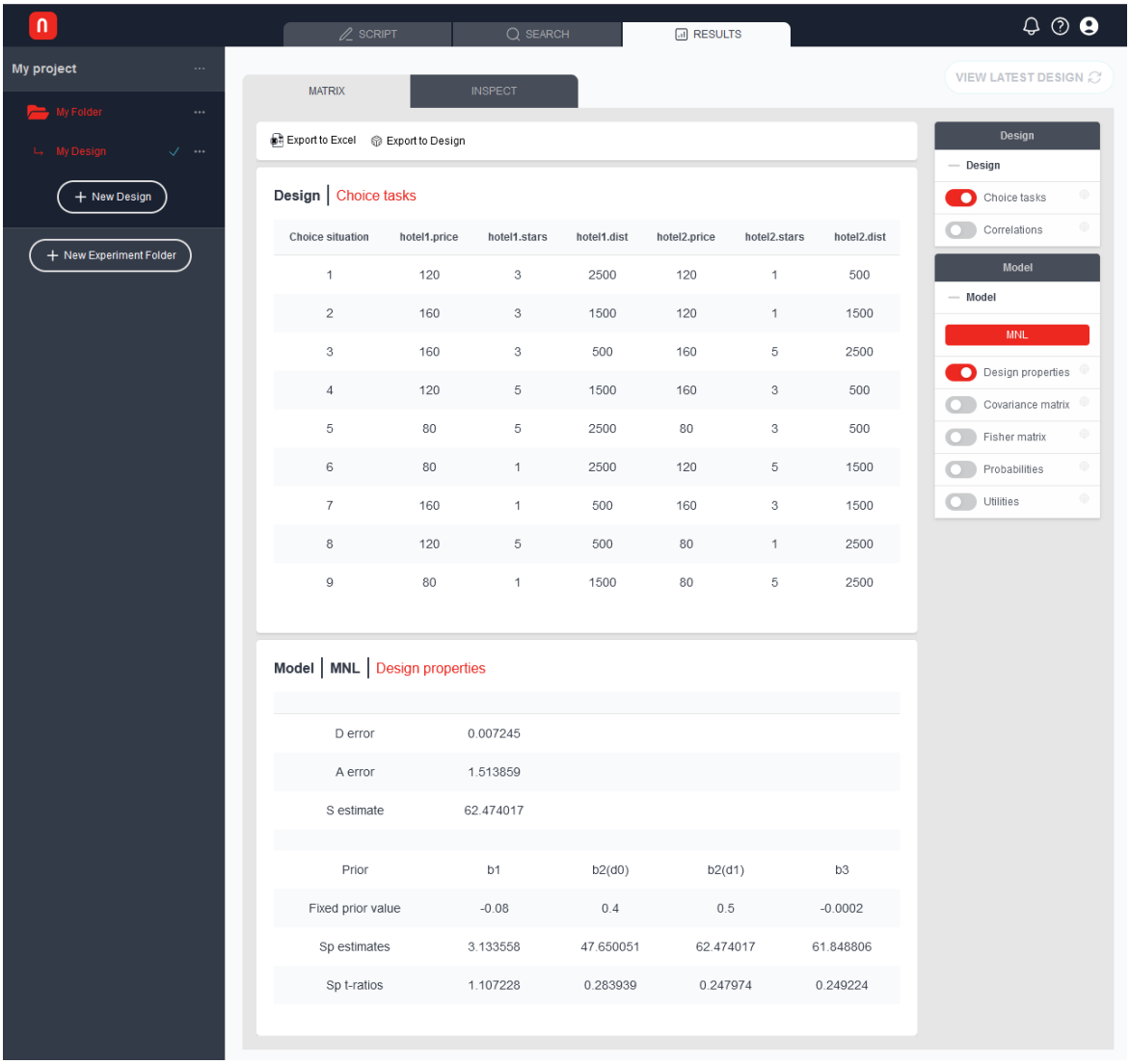
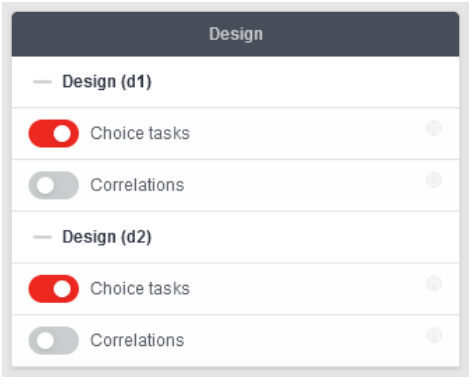


Figure 2.7: Design results

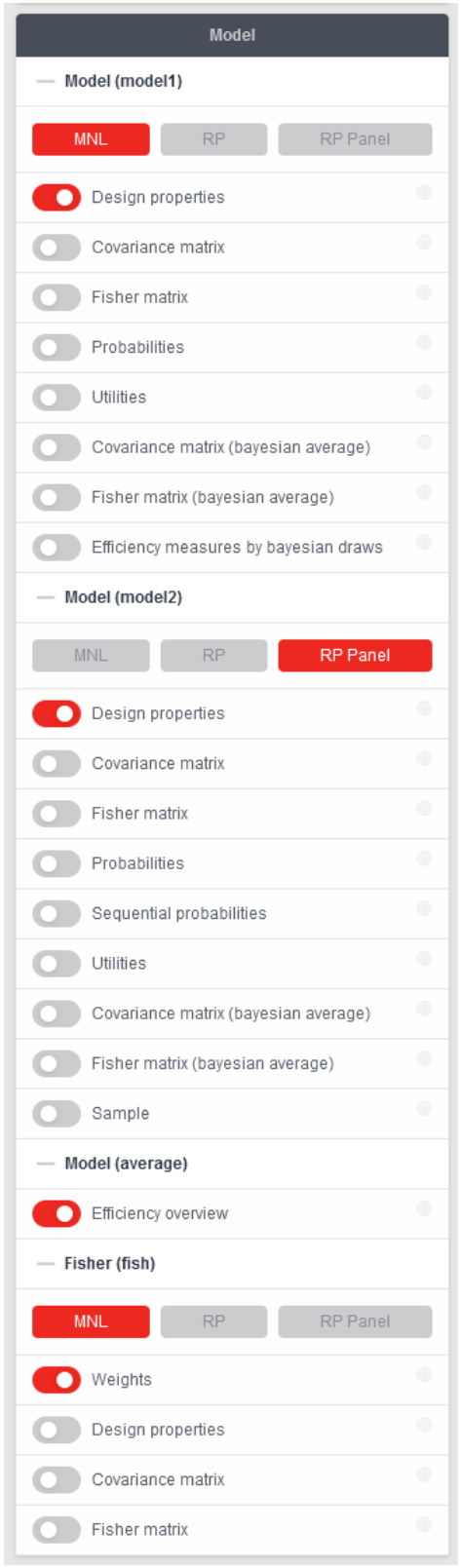
2.4 Inspecting results

The experimental designs generated by Ngene can be viewed in the *Results* tab of the project screen. Figure 2.7 shows a table with *Choice tasks* (choice situations) in the generated experimental design, as well as *Design properties*. The results are presented in two main sections. In the *Design* section one can also find *Correlations* based on Pearson product-moment correlation coefficients. In the *Model* section one can find further model-specific design properties, such as the *Covariance matrix* (assuming a single agent receives all choice tasks) and choice *Probabilities* in each choice task based on any specified priors. One can display or hide results by toggling various result items on or off. Although most scripts will only produce a single design and results for a single model, some scripts will generate multiple designs for multiple model types and the result options would automatically expand, see Figure 2.8. The available results may also depend on whether local or Bayesian priors were used when generating an efficient design.

Any information that is toggled on will be added to the matrix view and may require scrolling down to see them. To avoid the need for extensive scrolling to find results in a long list, one can immediately view a specific result by clicking on the finder button ( ) at the end of each result



(a) Multiple designs



(b) Multiple models

Figure 2.8: More extensive design results depending on script

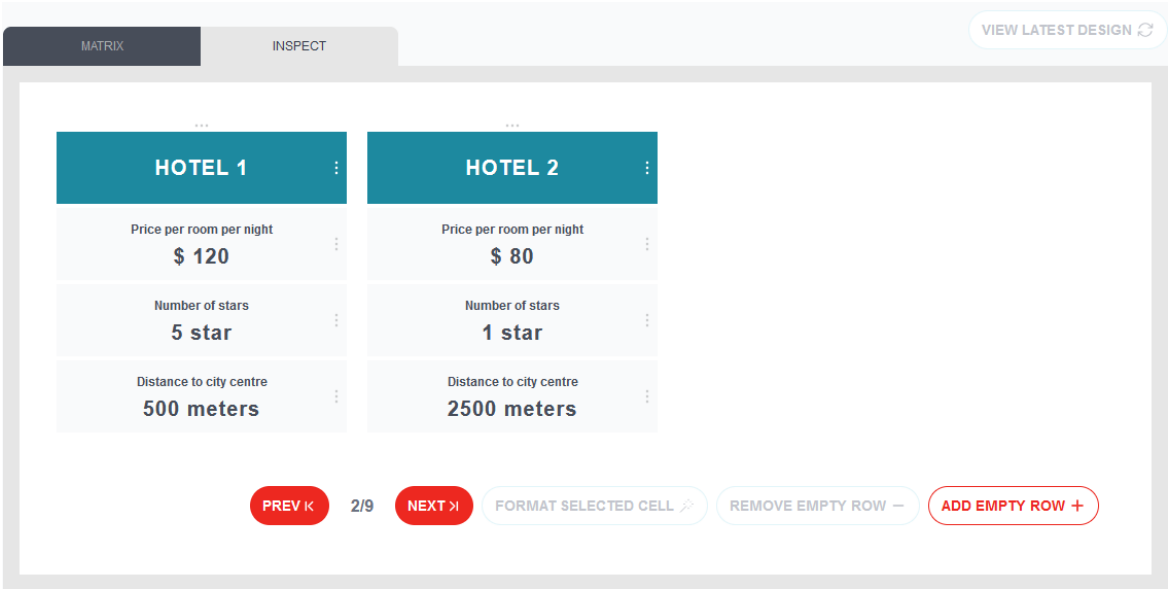


Figure 2.9: Inspecting choice tasks

item on the right-hand side. This finder button lights up when hovering the mouse over it, but only when the item is toggled on. To quickly scroll back to the top of the list, press the scroll up button (⏫) when it appears on the bottom right of the screen.

Most results are self-explanatory, but sample size estimates need more explanation. As shown in Equation (1.10), an S_p estimate refers to the estimated minimum number of respondents required to be able to estimate the parameter at a statistically significant level, based on a specified local informative prior and assuming a two-sided significance level of 0.05 (t-ratio of 1.96). An S_b estimate is its Bayesian counterpart that refers to the *average* minimum number of respondents across draws from a randomly distributed prior distribution. The overall S-estimate (also referred to as S-error) refers to the corresponding maximum S_p or S_b estimate across all parameters. For more information and caveats on the interpretation of sample size estimates, refer to Section 1.7.

To inspect and assess each individual choice task, it is easiest to do this in a choice task format rather than looking at rows in the design matrix. For this purpose, you can switch from the *Matrix* view to the *Inspect* view by clicking the respective tab. The Inspect view visualises each row in the design matrix as a choice task in a survey as shown in Figure 2.9. This screen also allows us to change the names of the alternatives and attributes and to convert the numbers in the design matrix into text that describes the meaning of each attribute level. Simply select any cell in the choice task and click on *Format Selected Cell* to open the pop-up window *Formatting Options*, see Figure 2.10. The formatting of the choice tasks is retained when clicking the *Duplicate & Edit* button in the script editor, but if attribute levels are modified in the script, then the formatting may also need updating.

It is important to point out that choice task formatting in Ngene is only meant for quick and easy inspection and evaluation by the analyst, it is not meant for data collection purposes. When transferring the experimental design to an online survey platform (such as SurveyEngine or Qualtrics), the final formatting suitable for respondents and data collection is applied in that platform, including replacing any attribute levels with images.

For choice experiments with labelled alternatives there may be different attributes per alternative; see, for example, Figure 2.11. Alternative-specific attributes have different variable names across alternatives in the script and would therefore be shown on separate rows when inspecting the choice

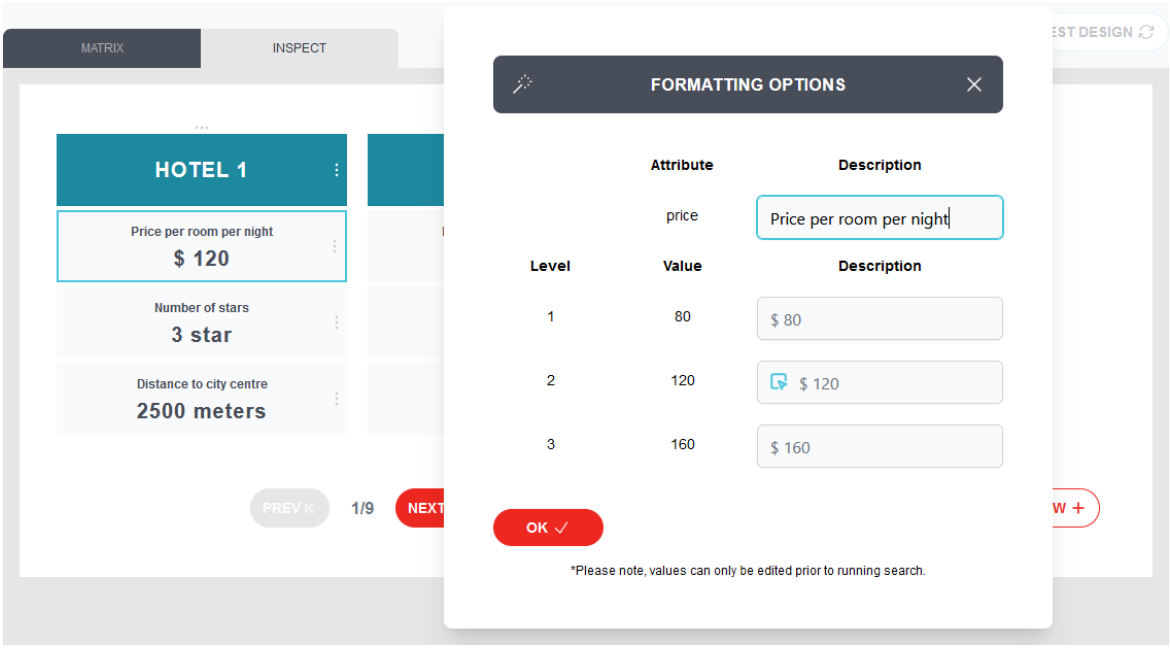


Figure 2.10: Formatting options for choice tasks

tasks. To change the location of an attribute within an alternative, simply drag the corresponding cell up or down within a column[2]. Columns can be dragged to the left and right to change order. In addition, any empty rows in the choice task can be removed by clicking on a cell in the empty row and then clicking the *Remove Empty Row* button. Similarly, an empty row can be added by clicking the *Add Empty Row* button.

2.5 Exporting and importing designs

Results can be exported to Excel by clicking the *Export to Excel* button near the top of the Results screen; see Figure 2.7. This exports results – but only the items that are *toggled on* – to an Excel spreadsheet (.XLSX) and saves the design and design properties in different worksheets tabs, see for example Figure 2.13. You can also click on the *Export to Design* button to export the results to an Ngene design file (.NGD), which can be opened in the desktop version of Ngene.

Any design that was exported from Ngene can also be imported again by selecting *Import design matrix* when creating a new project or after adding a new design to a folder, see Section 2.1. This is, for example, useful when sharing designs across Ngene users or when wanting to move a generated design from one project to another. Figure 2.12 shows the import design screen where files can be dropped and selected from your device. This screen also allows the import of Ngene design (.NGD) files that were created in the desktop version of Ngene.

A design that has been created with other software or that has been created manually can also be imported. This may, for example, be useful when using external candidate sets (see Section 6.9), when evaluating the efficiency of a design provided by someone else (see Section 5.8). The spreadsheet or CSV file must be in a specific format as outlined in the Example file that can be downloaded from the Import matrix screen; see Figure 2.12. One of the requirements is that it must contain a header row where the first cell contains the text 'Choice situation'. To be able to use this design in a script, it is important to put the attribute columns in the same order as they will appear in the utility functions

[2]It is possible to drag a cell above the alternative name, which is useful when the cell reflects to a scenario variable.

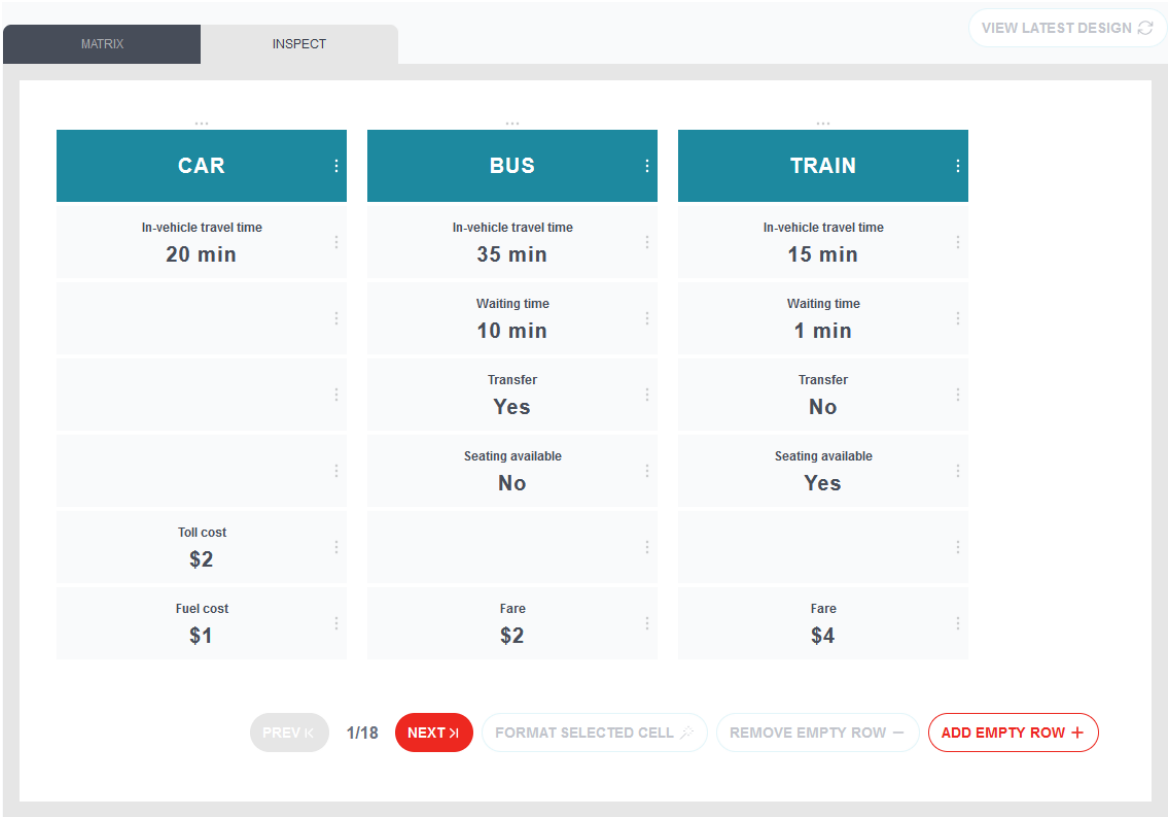


Figure 2.11: Choice task with alternative-specific attributes

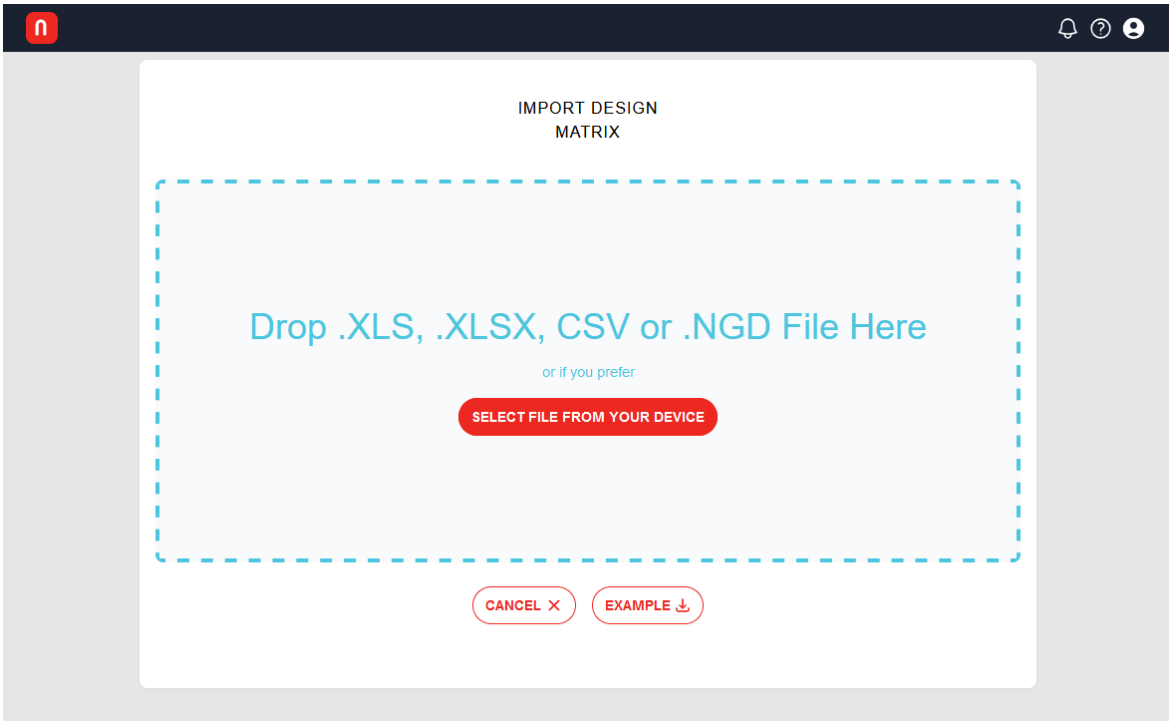
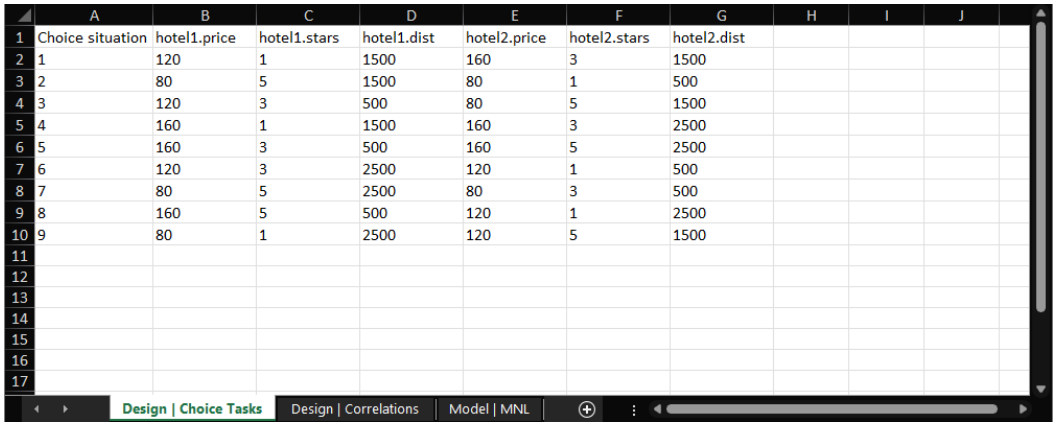


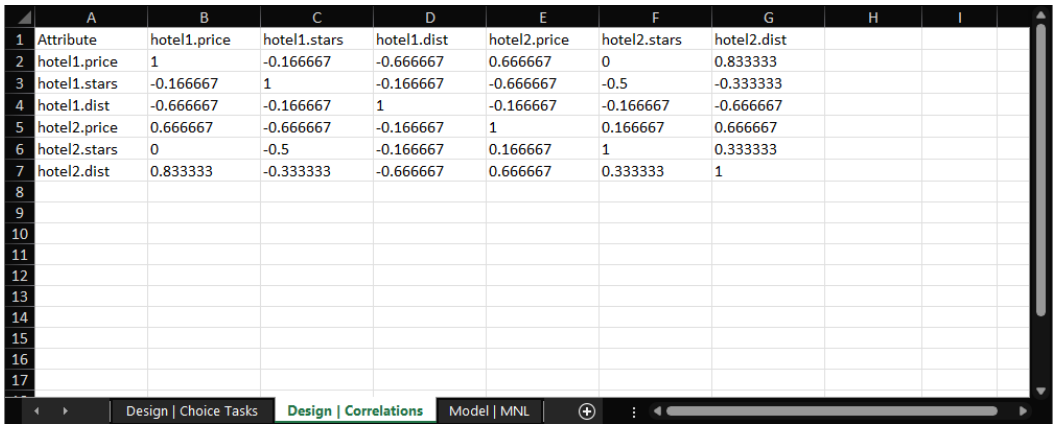
Figure 2.12: Import design



	A	B	C	D	E	F	G
1	Choice situation	hotel1.price	hotel1.stars	hotel1.dist	hotel2.price	hotel2.stars	hotel2.dist
2	1	120	1	1500	160	3	1500
3	2	80	5	1500	80	1	500
4	3	120	3	500	80	5	1500
5	4	160	1	1500	160	3	2500
6	5	160	3	500	160	5	2500
7	6	120	3	2500	120	1	500
8	7	80	5	2500	80	3	500
9	8	160	5	500	120	1	2500
10	9	80	1	2500	120	5	1500

Design | Choice Tasks | Design | Correlations | Model | MNL

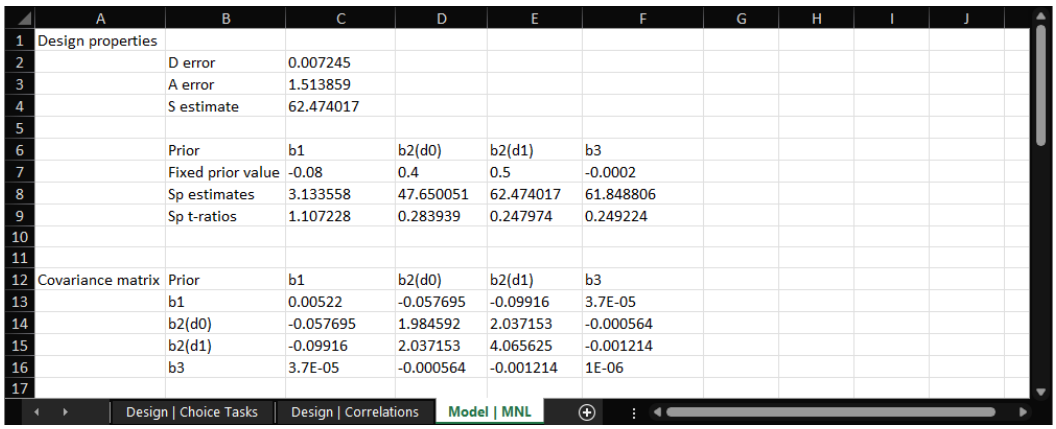
(a) Design matrix



	A	B	C	D	E	F	G
1	Attribute	hotel1.price	hotel1.stars	hotel1.dist	hotel2.price	hotel2.stars	hotel2.dist
2	hotel1.price	1	-0.166667	-0.666667	0.666667	0	0.833333
3	hotel1.stars	-0.166667	1	-0.166667	-0.666667	-0.5	-0.333333
4	hotel1.dist	-0.666667	-0.166667	1	-0.166667	-0.166667	-0.666667
5	hotel2.price	0.666667	-0.666667	-0.166667	1	0.166667	0.666667
6	hotel2.stars	0	-0.5	-0.166667	0.166667	1	0.333333
7	hotel2.dist	0.833333	-0.333333	-0.666667	0.666667	0.333333	1

Design | Choice Tasks | Design | Correlations | Model | MNL

(b) Correlations



	A	B	C	D	E	F
1	Design properties					
2		D error	0.007245			
3		A error	1.513859			
4		S estimate	62.474017			
5						
6		Prior	b1	b2(d0)	b2(d1)	b3
7		Fixed prior value	-0.08	0.4	0.5	-0.0002
8		Sp estimates	3.133558	47.650051	62.474017	61.848806
9		Sp t-ratios	1.107228	0.283939	0.247974	0.249224
10						
11						
12	Covariance matrix	Prior	b1	b2(d0)	b2(d1)	b3
13		b1	0.00522	-0.057695	-0.09916	3.7E-05
14		b2(d0)	-0.057695	1.984592	2.037153	-0.000564
15		b2(d1)	-0.09916	2.037153	4.065625	-0.001214
16		b3	3.7E-05	-0.000564	-0.001214	1E-06

Design | Choice Tasks | Design | Correlations | Model | MNL

(c) Model-specific design properties

Figure 2.13: Exported design in Excel

in the script (Ngene does not automatically match variable names in the script with header names in the file).

3

The basics of writing scripts

This chapter describes the basics of writing Ngene syntax to generate experimental designs for choice experiments. It shows how to generate full factorial experimental designs and random fractional factorial designs for unlabelled and labelled experiments. Further, it explains how to modify the script to avoid non-sensible choice tasks, namely choice tasks with identical profiles, repeated choice tasks, and choice tasks with a dominant alternative.

3.1 Introduction to Ngene scripts and syntax

In this section, we introduce the structure of Ngene scripts and how syntax is written. First, we introduce the terminology used throughout this manual.

Script. A text in a specific structured format, referred to as *syntax*, that contains instructions for the creation of an experimental design for a choice experiment.

Instruction. Setting a property of an experimental design using syntax.

Property. A design characteristic that can be represented by *property values* and possibly some parameters.

All scripts start with the command `design` and end with the dollar symbol (`$`). Instructions are to be written between this command and the end symbol, where each instruction is separated by a semicolon (`;`). The order in which these instructions appear is irrelevant. There are three properties that must be set in each script, namely

- `alts` to define the alternatives in each choice task, see Section 3.2
- `rows` to specify the design size, see Section 3.3
- `model` to specify the utility functions of all alternatives, see Section 3.7

In addition, the experimental design strategy/type must be specified by adding one of the following properties to the script (see Section 3.4): `fact`, `rand`, `orth`, or `eff`.

```
1  design  ? Laptop choice example
2  ;alts = (laptopA, laptopB)
3  ;rows = 20
4  ;block = 4
5  ;rand
6  ;model:     ? using design coding
7  U(laptopA, laptopB) = proc * PROCESSOR[0,1,2,3]
8                      + stor * STORAGE[0,1,2,3]
9                      + cost * PRICE[0,1,2,3]
10 ? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7), 3(Core i9)
11 ? STORAGE:    0(256 GB),  1(512 GB),  2(1 TB),    3(2TB)
12 ? PRICE:      0($1200),   1($1500),   2($1800),   3($2100)
13 $
```

Script 3.1: Example script

Depending on the property, a *property value* can be a *string* (defined by a sequence of characters) or a *number*. For example, `alts = laptopA, laptopB` assigns two user-defined strings to property `alts`, while `rows = 9` assigns a user-defined number to property `rows`.

There are a few rules when writing instructions:

- Syntax is not case-sensitive, hence using `ROWS`, `rows`, or `Rows` all yield the same result
- Spaces and empty lines between properties, values, labels, and parameters are ignored, hence `rows = 9` yields the same result as `rows=9`
- Spaces should not appear within a property name or property value, hence using `row s` instead of `rows` will generate an error
- User-defined strings can consist of combinations of letters, numbers, and symbols, for example `laptop_1`
- User-defined strings cannot only consist of numbers
- User-defined strings cannot include the following symbols: `? ; $ : = , . | ( ) [ ] * + -`
- User-defined strings cannot be a reserved name such as `rows`

Consider the choice between two generic laptops, namely Laptop A and Laptop B, each characterised by three attributes, namely processor type (with levels Core i3, Core i5, Core i7, and Core i9), amount of hard disk storage (with levels 256 GB, 512 GB, 1 TB, and 2 TB), and price (with levels $1200, $1500, $1800, and $2100). Script 3.1 shows an example of a complete script. In this script, five properties are defined, namely `alts`, `rows`, `block`, `rand` and `model`.

User-defined strings in this script are `laptopA`, `laptopB`, `proc`, `PROCESSOR`, `stor`, `STORAGE`, `cost`, and `PRICE`. Any text after a question mark (`?`), see lines 1, 6, and 14–16, is considered a comment and is ignored by Ngene. Including comments in the script is useful for the user to document design choices, explain attribute levels, etc. Strings chosen by the user should preferably be informative, and spacing in the script should preferably be functional to improve readability. Script 3.2 does exactly the same thing as Script 3.1. However, Script 3.1 is clearly more readable since it has informative strings, legible spacing, and comments.

```
1  design
2  ;alts=(alt1,alt2);rows=20;block=4;rand
3  ;model: U(alt1,alt2)=b1*x1[0,1,2,3]+b2*x2[0,1,2,3]+b3*x3[0,1,2,3]
4  $
```

Script 3.2: Less readable script

3.2 Defining alternatives

The property `alts` is mandatory in each script. It defines the number of alternatives and gives them a name. The property `alts` in Script 3.1 is defined by two strings, `laptopA` and `laptopB`, separated by a comma (`,`). This instructs Ngene that the design needs to have two alternatives, and their respective strings are used later to define the utility function of each alternative in the `model` property. For example, the following syntax defines three alternatives in a labelled experiment for cancer treatment.

```
;alts = chemotherapy, surgery, immunotherapy
```

If two or more alternatives are of the same generic label (i.e., unlabelled alternatives that have the same utility function), then this needs to be indicated, as it influences the design generation. Unlabelled alternatives can be grouped together by placing parentheses around them. For example, the alternatives in Script 3.1 are unlabelled and therefore are grouped as follows:

```
;alts = (laptopA, laptopB)
```

In a previous version of Ngene, unlabelled alternatives were indicated with an asterisk (`*`). Such syntax still works; for example, `alts = laptopA*, laptopB*` is equivalent to the syntax above, but has the limitation that it does not allow defining multiple groups of generic alternatives as shown below.

Alternatives can be a mix of labelled and unlabelled alternatives, for example, in the syntax below there are three labels, namely car, bus, and train, where two generic car alternatives (with different routes) are included in the choice set.

```
;alts = (car1, car2), bus, train
```

Multiple groups of unlabelled alternatives can also be specified. For example, this syntax defines two labels, car and train, each with two generic alternatives:

```
;alts = (car1, car2), (train1, train2)
```

An opt-out (no choice) alternative is a special type of alternative that can be added to an experiment. Since an opt-out alternative is always labelled, it should never be grouped with other alternatives; see the example syntax below.

```
;alts = (laptopA, laptopB), optout
```

Another special type of alternative is a status quo alternative. Such an alternative can share the same label or have a different label. Below an example where a generic status quo alternative is assumed. Status quo alternatives are discussed in more detail in Section 6.5.

```
;alts = (currentpolicy, policyA, policyB)
```

You are looking to buy a new laptop *for at home*. Which of the following laptops would you prefer?	
Laptop A	Laptop B
Intel Core i5 processor 256 GB hard-disk drive $2100	Intel Core i5 processor 256 GB hard-disk drive $2100
○	○

Figure 3.1: Laptop choice task with identical profiles

Choice task 1. You are looking to buy a new laptop *for at home*. Which of the following laptops would you prefer?	
Laptop A	Laptop B
Intel Core i3 processor 256 GB hard-disk drive $1500	Intel Core i5 processor 1 TB hard-disk drive $1800
◉	○

Choice task 2. You are looking to buy a new laptop *for at home*. Which of the following laptops would you prefer?	
Laptop A	Laptop B
Intel Core i5 processor 1 TB hard-disk drive $1800	Intel Core i3 processor 256 GB hard-disk drive $1500
○	◉

Figure 3.2: Two choice tasks that capture the same information

When generating an efficient or random design, Ngene performs additional checks when generic alternatives are present (as identified by a grouping with parentheses) to avoid undesirable choice tasks. First, Ngene identifies and removes choice tasks where the profiles of alternatives are identical. Figure 3.1 illustrates such a choice task with completely overlapping attribute levels. Secondly, Ngene detects and removes choice tasks in the design that capture the same information as other choice tasks. An example is shown in Figure 3.2 where it is clear that the two choice tasks are essentially the same since the profiles are merely swapped and at most one of these choice tasks should appear in the design. Thirdly and most importantly, Ngene automatically avoids choice tasks with strictly dominant alternatives when parameter priors are available that indicate the preference order of attribute levels, see Section 3.8. It is important to note that Ngene does *not* perform these checks for orthogonal designs, as it is generally not possible to avoid undesirable choice tasks without violating orthogonality.

Such undesirable choice tasks generally do not exist in experiments in which all alternatives are labelled. Even if a choice task has identical profiles across the alternatives, an agent would still making trading-offs on the labels; see, for example, Figure 3.3.

Consider a *70 year old* patient with *advanced prostate cancer*. As his doctor, what treatment would you recommend?		
Radiotherapy	Surgery	Neither
Medium risk of permanent side effects 70% probability of curing patient	Medium risk of permanent side effects 70% probability of curing patient	
◉	○	○

Figure 3.3: Treatment choice task with identical profiles

3.3 Defining design size and blocking

Property `rows` defines the design size, that is, the number of choice tasks in the experimental design, which equals the number of rows in the design matrix.

The number of rows is typically set to a number that reflects the desired number of choice tasks in a fractional factorial design. This number should not be too small; see the discussion on determining an appropriate design size in Section 1.4. If the number of rows in the design is not sufficient to capture information about all parameters, then Ngene will produce an error: "There are not enough degrees of freedom. Decrease the number of parameters or increase the number of rows to at least x", where x is the minimum required design size.

When generating an orthogonal design, the number of rows specified in the script may not be consistent with any available orthogonal arrays. In that case, Ngene will automatically increase the design size until it is able to locate an orthogonal design (if it exists).

In Script 3.1 the number of rows is set to 20, thereby generating 20 choice tasks.

```
;rows = 20
```

A special case of the number of rows occurs when using `fact` to generate a full factorial design. In this case, the default will be that all possible design rows are generated, which can be specified explicitly as

```
;rows = all
```

The number of choice tasks in an experimental design is often too large to give to a single agent, and hence the design will need to be blocked into smaller subsets. This can be performed by adding the optional property `block` to the script, for example:

```
;block = 4
```

In this case, the experimental design is blocked into four equal parts (or near-equal parts if the number of rows is not divisible by the number of blocks). In Script 3.1 where `rows = 20`, this means that each block contains five choice tasks. In the survey, each agent is assigned one block of five choice tasks. Although blocking aims to achieve (a high degree of) attribute level balance, this can only be guaranteed for orthogonal designs (see Chapter 4). For other design types, including random

```
1  design  ? labelled car purchase example
2  ;alts = car, no_car
3  ;fact
4  ;model:
5  U(car)    = con_c
6            + eng.dummy  * ENGINE[0,1,2,3]
7            + col.dummy  * COLOUR[0,1,2,3]
8            + type.dummy * TYPE[0,1,2,3]
9  ? ENGINE: 0(Diesel), 1(Petrol), 2(Hybrid),    3(Electric)
10 ? COLOUR: 0(White),  1(Red),    2(Black),     3(Blue)
11 ? TYPE:   0(Sedan),  1(Coupe),  2(Hatchback), 3(Station wagon)
12 $
```

Script 3.3: Full factorial design

designs, Ngene minimises the correlation of the block number with the attribute levels, where lower correlations generally result in a higher degree of attribute level balance within each block. However, results vary since such correlations are only a proxy for attribute level balance. Since blocking does not have any impact on other statistical properties of the design, the user can manually re-allocate choice tasks to blocks if desired.

3.4 Defining design strategy

An instruction in the script is needed that tells Ngene *how* to generate the design matrix. This allocation depends on the experimental design strategy. Each strategy requires the use of a specific property in the script.

There exist three main experimental design strategies; see also Section 1.5.

- Property `fact` creates a full factorial design, see Section 3.5
- Property `rand` create a random fractional factorial design, see Section 3.6
- Property `orth` creates an orthogonal fractional factorial design, see Chapter 4
- Property `eff` creates an efficient fractional factorial design, see Chapter 5

In each script, the selection of a design strategy is mandatory. It is possible to combine properties `eff` and `orth` in the same script, which instructs Ngene to find an orthogonal design with the highest efficiency. It is not possible to combine `fact` or `rand` with `orth` or `eff` in the same script, since these design strategies are in conflict with each other.

3.5 Generating full factorial designs

The property `fact` can be used to generate a full factorial design. This property has no values and can be added directly as an instruction; see syntax below and also line 3 in the Script 3.3.

```
;fact
```

In Script 3.3, alternative `car` is characterised by three attributes, namely, engine type, colour and car type, while `no_car` is an opt-out alternative without any attributes. When using `fact` the property `rows` can be omitted since it automatically defaults to `rows = all`, but the latter could also be explicitly added to the script for clarity reasons. The `model` property, and how to define utility functions, will be explained in Section 3.7.

Running Script 3.3 generates a full factorial design with $4 \times 4 \times 4 = 64$ rows. A depiction of such a full factorial design was shown in Figure 1.4(a) in Chapter 1. Note that Ngene produces a warning when running this script: "Defaulting to prior values of zero for the following priors: 'proc, stor, cost'". This warning means that no priors for these parameters were specified and hence were set to zero. This warning can be ignored unless the user wants to specify informative or noninformative priors to avoid possible strictly dominant alternatives (see Section 3.8).

Now consider again Script 3.1. If property `rand` would be replaced by property `fact` and `rows = 20` would be replaced by `rows = all` (or simply omitted), then it would generate a full factorial design with $64^2 = 4,096$ choice tasks, based on $4^3 = 64$ unique profiles in each alternative, if no further constraints were imposed. However, since generic alternatives are defined in the script on line 2, Ngene automatically removes choice tasks where profiles of both laptops are identical, such that only 4,032 choice tasks remain in the full factorial design.

In most cases the size of a full factorial design is prohibitively large, and hence most choice experiments adopt a fractional factorial experimental design strategy (i.e., an efficient, orthogonal, or random design).

3.6 Generating random designs

A random fractional factorial design is a random selection from a full factorial design and is simply referred to as a random design. The property `rand` can be used to generate such a design by adding this property without any values to the script.

```
;rand
```

To generate a random design in the labelled car purchase example, we simply replace the property `fact` in Script 3.3 with the property `rand`.[1] Further, property `rows` is required to specify the desired number of design rows. For example, to generate a random design with 16 rows (randomly selected out of 64 rows in the full factorial), we specify `rows = 16` as shown in Script 3.4.

Figure 3.4 illustrates a possible result for a random design for the same labelled car purchase example, noting that Ngene will produce a different random design each time the script is run. In contrast to a full factorial design, a random design will only contain certain attribute level combinations and is generally not attribute level balanced.

Using a random design strategy is usually only a good idea if the expected sample size is large (i.e., thousands) and if the design size has many rows (i.e., hundreds) to benefit from the variety in the data that random designs offer.

3.7 Defining model utility functions

When designing a choice experiment, it is good practice to formulate preliminary utility functions for all alternatives in the choice model, as this assists in understanding how the data later will be used in model estimation. In Ngene, utility functions are specified through the mandatory property `model`. This is the most complex property that requires the most attention. All complex properties in Ngene, including `model` and properties for imposing attribute level constraints (`require`, `reject`, `cond`, see Chapter 6), are specified in an environment that starts with a colon (`:`) where each property

[1] In a previous version of Ngene, the property `fact` was used for both full factorial designs and random fractional factorial designs. While this functionality is still available, the `rand` property is preferred for random designs as we are expanding its algorithmic capabilities.

	Car			No car
Choice task	Engine	Colour	Type	
1	2	2	3	–
2	0	1	3	–
3	3	2	3	–
4	2	2	1	–
5	0	0	0	–
6	1	2	0	–
7	0	3	1	–
8	0	1	1	–
9	0	1	2	–
10	1	1	1	–
11	3	2	1	–
12	2	1	0	–
13	2	0	0	–
14	0	2	1	–
15	1	2	2	–
16	1	3	2	–

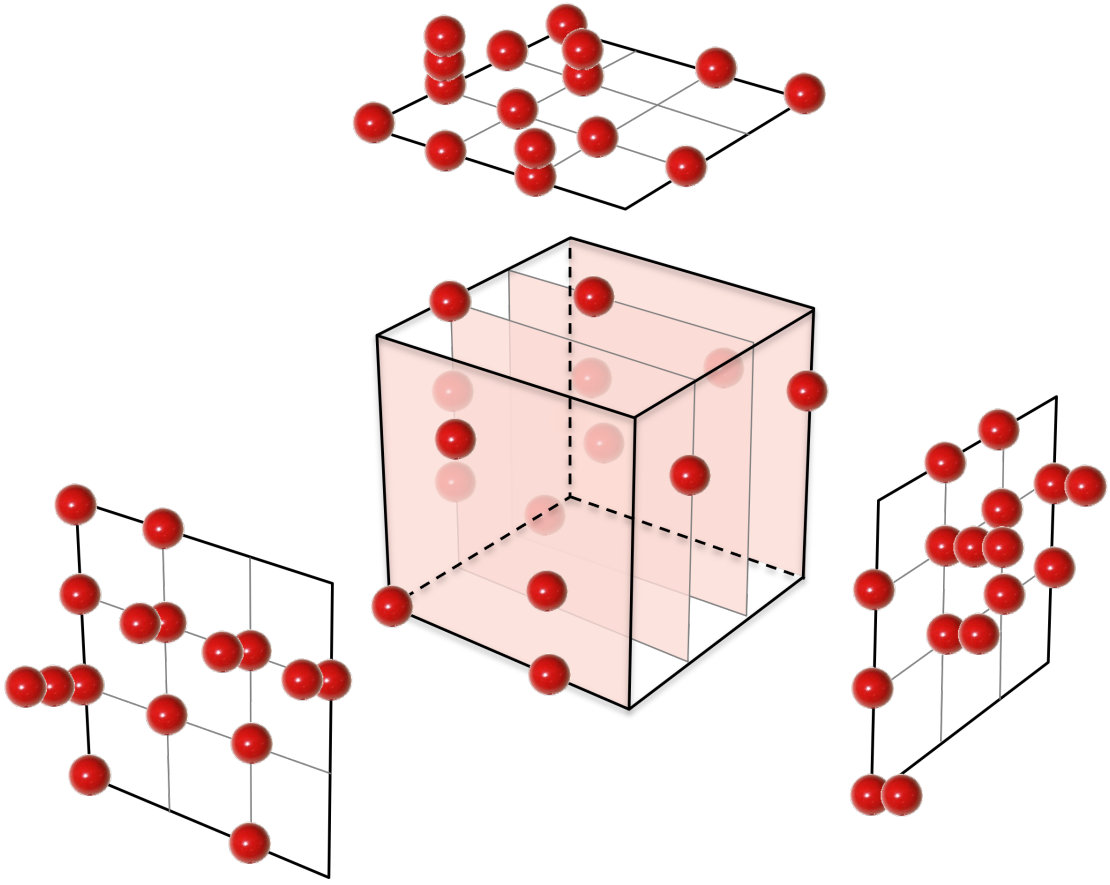


Figure 3.4: Random design and graphical depiction

```
design  ? labelled car purchase example
;alts = car, no_car
;rows = 16
;rand
;model:
U(car)    = con_c
          + eng.dummy  * ENGINE[0,1,2,3]
          + col.dummy  * COLOUR[0,1,2,3]
          + type.dummy * TYPE[0,1,2,3]
? ENGINE: 0(Diesel), 1(Petrol), 2(Hybrid),    3(Electric)
? COLOUR: 0(White),  1(Red),    2(Black),     3(Blue)
? TYPE:   0(Sedan),  1(Coupe),  2(Hatchback), 3(Station wagon)
$
```

Script 3.4: Random design

value is separated by a special symbol. In the case of the `model` environment, each property value is represented by a utility function for an alternative and is separated by a slash (`/`).

In laptop choice Script 3.1, two utility functions need to be defined, one for alternative `laptopA` and one for alternative `laptopB`. Each utility function in the `model` property starts with a `U` and the alternative label between parentheses, e.g., `(laptopA)`, followed by an equal sign (`=`). On the right-hand side of the equal sign, one specifies a function that is linearly additive in the parameters. These utility functions can contain main effects, interaction effects, and label-specific constants. An example for the laptop choice experiment is shown below. Note that there is no separator after the second utility function; including a `/` after the second utility function would result in an error when running the script.

```
;model:
U(laptopA)  = proc * PROCESSOR[0,1,2,3]
            + stor * STORAGE[0,1,2,3]
            + cost * PRICE[0,1,2,3]
            /
U(laptopB)  = proc * PROCESSOR
            + stor * STORAGE
            + cost * PRICE
? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7), 3(Core i9)
? STORAGE:    0(256 GB),  1(512 GB),  2(1 TB),    3(2TB)
? PRICE:      0($1200),   1($1500),   2($1800),   3($2100)
```

Only main effects are specified in the above utility functions, where `proc`, `stor`, and `cost` are parameters, and `PROCESSOR`, `STORAGE`, and `PRICE` are attributes. Note that we merely use lower and upper case in this manual to clearly indicate the difference between model parameters and attributes (but, as mentioned, the syntax is not case sensitive). Using the same string for a parameter across multiple alternatives means that it has the same value, i.e., it is a generic parameter. For example, the parameter `proc` in alternative `laptopA` is the same as the parameter `proc` in alternative `laptopB`.

Each attribute is followed by a set of allowable attribute levels indicated by a series of values between square brackets. These values need to be numerical in Ngene, it is not possible to define `PROCESSOR[Core_i3,Core_i5,Core_i7,Core_i9]` in the syntax. Instead, attribute levels need to be represented by numbers in the syntax using design coding, estimation coding, or another coding scheme that is preferred by the user; see Chapter 1.3. Design coding is traditionally used to generate orthogonal designs. When generating an efficient design, one should typically use estimation coding

to allow the correct computation of utilities, choice probabilities, and Fisher information. Using the same string for an attribute across multiple alternatives means that it represents the same generic attribute with the same attribute levels (but, of course, the levels shown to an agent in each choice task will generally be different across alternatives). Therefore, the levels of a generic attribute only need to be defined the first time this attribute appears in a utility function, and can be omitted in all subsequent alternatives. For example, attribute `PROCESSOR` in the alternative `laptopB` allows the same levels as `PROCESSOR` in `laptopA`, namely `[0,1,2,3]`.

If the utility functions of two or more alternatives are identical, e.g., in the case of unlabelled alternatives, then they can be specified simultaneously. For example, the utility functions of the alternatives `laptopA` and `laptopB` in the laptop choice experiment are the same; therefore, in Script 3.1 a convenient shortcut is used whereby both utility functions are specified simultaneously on lines 6–9 (repeated below). In this syntax, both alternatives are listed in the same utility function between parentheses, separated by a comma.

```
;model:
U(laptopA, laptopB)  = proc * PROCESSOR[0,1,2,3]
                     + stor * STORAGE[0,1,2,3]
                     + cost * PRICE[0,1,2,3]
```

In Script 3.1 we adopted *design coding* for the attribute levels, e.g., using levels 0, 1, 2, and 3. Note that these values are merely placeholders that need to be replaced with meaningful descriptions in the survey. Any other placeholder values could have been used, for example, we could have defined `PROCESSOR[3,5,7,9]` where the numbers are more representative for processor types i3, i5, i7, and i9 to improve the interpretation of attribute levels in the experimental design. To remind oneself of what each design coded level means, it is recommended to put comments in the script.

While design coding is fine when generating an orthogonal or random design, it is strongly recommended to always use estimation coding in Ngene to resemble the model specification that will be used in model estimation as this allows counting the number of parameters that need to be estimated (which influences the minimum required design size), and it also allows the computation of design efficiency measures. In the following syntax, we have converted the above utility functions to *estimation coding*. Estimation coding in Ngene requires dummy or effects coding for the levels of qualitative (categorical) attributes. With respect to quantitative (numerical) attributes, one typically uses the numerical values shown to agents in the survey as attribute levels in the utility function, where all levels of an attribute are expressed in the same unit. In the syntax below, processor type is dummy coded, hard-disk storage is a quantitative variable expressed in GB, and price is expressed in dollars.

```
;model:     ? using estimation coding
U(laptopA, laptopB)  = proc.dummy * PROCESSOR[0,1,2,3]
                     + stor       * STORAGE[256,512,1024,2048]
                     + cost       * PRICE[1200,1500,1800,2100]
? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7),  3(Core i9)
? STORAGE:    256 GB,     512 GB,     1024 GB,     2048 GB
? PRICE:      $1200,      $1500,      $1800,       $2100
```

A qualitative attribute can be dummy coded in Ngene by adding `.dummy` to the associated parameter. Dummy coding replaces the attribute variable with $L-1$ dummy variables, where L is the number of levels of the attribute, and where each of the $L-1$ dummy variables has its own parameter. In the above syntax, `.dummy` is added to `proc` so that the attribute `PROCESSOR` is converted into

three dummy coded variables. Since `proc` is a generic parameter in both alternatives, the attribute `PROCESSOR` in laptop B will automatically be dummy coded as well, even when `.dummy` is omitted after `proc` in the utility function of laptop B. Ngene considers the *last level* by default as the base level. This means that level 3 (Core i9 processor) has base dummy coding $(0, 0, 0)$, level 0 is dummy coded as $(1, 0, 0)$, level 1 is coded as $(0, 1, 0)$, and level 2 is coded as $(0, 0, 1)$. Parameter `proc` is now split into three separate parameters, where the first parameter expresses the utility of level 0 relative to base level 3, the second parameter expresses the utility of level 1 (relative to the base), and the third parameter expresses the utility of level 2 (relative to the base). If one prefers Core i3 to be the base level, then one can rearrange the levels in the utility function so that 0 becomes the last level, i.e., `PROCESSOR[1,2,3,0]` or `PROCESSOR[3,2,1,0]`.

Similarly, if effects coding is preferred over dummy coding, then one adds `.effects` to the relevant parameter name, where the last level is again by default considered the base level with associated effects coding $(-1, -1, -1)$. The coding of the other levels is the same as with dummy coding, namely, level 0 is effects coded as $(1, 0, 0)$. With effects coding, the first parameter expresses the utility of level 0 relative to the average (instead of the base level in dummy coding), the second parameter expresses the utility of level 1 relative to the average, etc.

It is possible to also apply dummy or effects coding to quantitative attributes hard-disk storage and price, but this removes the advantages that a quantitative variable has over a qualitative variable, namely the computation and interpretation of willingness-to-pay becomes less straightforward and one loses the ability to interpolate and extrapolate utilities (e.g., predict utilities for a hard-disk of size 768 GB or 3 TB).

If desired, a continuous non-linear transformation can be applied to the quantitative attribute levels to describe a specific type of behaviour. In the syntax above, `STORAGE` has exponentially increasing attribute levels and one may assume that the additional utility decreases when hard-disk storage increases. This could be achieved by applying, for example, a logarithmic transformation such that we replace the levels for hard-disk storage, `[256,512,1024,2048]`, with transformed levels, `[5.545,6.238,6.931,7.624]`, where $\ln(256) = 5.545$.

If attributes have different levels, then they need to be named with different strings in the syntax. In the utility functions below, we assume that there are four price levels in attribute `PRICEA` for Laptop A, while only two price levels exist for Laptop B in attribute `PRICEB`. Since the utility functions are no longer the same, they need to be specified separately.

```
;model:
U(laptopA)  = proc.dummy * PROCESSOR[0,1,2,3]
            + stor       * STORAGE[256,512,1024,2048]
            + cost       * PRICEA[1200,1500,1800,2100]
            /
U(laptopB)  = proc       * PROCESSOR
            + stor       * STORAGE
            + cost       * PRICEB[1800,2100]
? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7), 3(Core i9)
? STORAGE:    0(256 GB),  1(512 GB),  2(1 TB),    3(2TB)
? PRICEA/B:   $1200),     $1500,      $1800,      $2100
```

The levels of a qualitative attribute would preferably be assigned in a specific way, e.g., from small to large or in increasing or decreasing order of preference for an ordinal variable to facilitate checking for strictly dominant alternatives (see Section 3.8). For example, in Script 3.1 we assign design codes to processor types in increasing order of speed. The order in which attribute levels are shown in a

```
design  ? Treatment choice example
;alts = radiotherapy, surgery, neither
;rows = 48
;block = 4
;rand
;model:
U(radiotherapy) = con_r
                + side_r.effects * SIDE_EFFECTS[0,1,2]
                + cure_r         * PROB_CURE[0.3,0.5,0.7,0.9]
                /
U(surgery)      = con_s
                + side_s.effects * SIDE_EFFECTS
                + cure_s         * PROB_CURE
? SIDE_EFFECTS: 0(low risk), 1(medium risk), 2(high risk)
? PROB_CURE:    30%,         50%,            70%,          90%
$
```

Script 3.5: Design with opt-out alternative

utility function is not important. In other words, one could have used `PRICE[1800,1500,2100,1200]` or `PROCESSOR[2,0,3,1]`, where 0 and 3 still represent the slowest and fastest processors, respectively.

Our laptop choice example considered unlabelled alternatives where all utility functions have the same attributes and the same generic parameters. Let us now consider a labelled experiment related to the treatment choice task shown in Figure 3.3, which has labelled alternatives `radiotherapy`, `surgery`, and opt-out alternative `neither`. In Script 3.5, each treatment alternative has two attributes, namely the risk of side effects (with three levels) and the probability of curing the patient (with four levels). In a labelled experiment, the parameters can be alternative-specific, hence we use a different parameter name `side_r` and `side_s` for the side effects of radiotherapy and surgery, respectively. Similarly, two different parameters, `cure_r` and `cure_s`, are used for the attribute that describes the probability of curing the patient. In addition, label-specific coefficients `con_r` and `con_s` are added to the utility functions of radiotherapy and surgery, where the opt-out is the reference alternative and does not have a constant (that is, it is normalised to zero). Although adding label-specific constants to utility functions is important when generating efficient designs, they have no impact on the generation of full factorial, orthogonal, and random designs.

It is important to realise that dummy or effects coding increases the number of parameters if the attribute has more than two levels. The model specified in Script 3.5 has 8 parameters in total, namely 2 constants, 4 coefficients for the qualitative attribute, and 2 coefficients for the quantitative attribute. Experiments where quantitative attributes have many levels will result in a model with many parameters and therefore will require larger design sizes (i.e., more rows).

If a certain alternative is defined in `alts` but no utility function for this alternative is specified in `model`, then Ngene fixes its utility to zero. This is useful for an opt-out alternative, which has no attributes, and its utility is typically set to zero. To illustrate, in Script 3.5 we specified `neither` in the `alts` property but omitted its utility function in the `model` property such that its utility is set to zero. Instead of normalising the constant for `neither`, one could also normalise one of the other constants. For example, in the syntax below the constant for `radiotherapy` is normalised to zero while a constant for the opt-out alternative is estimated. This results in a model that describes identical behaviour with the same number of parameters.

```
design  ? Mode choice example
;alts = car, train, bike
;rows = 60
;block = 10
;rand
;model:
U(car)   = con_c
         + tt_car   * TRAVELTIME_CAR[5,10,15]   ? car travel time in minutes
         + fuel     * FUELCOST[1,2,3]           ? fuel cost in dollars
         + park     * PARKING[1,2]              ? parking cost in dollars
         /
U(train) = con_t
         + access   * ACCESSTIME[1,5,10]        ? access time in minutes
         + tt_train * INVEHICLETIME[10,15,20]   ? train time in minutes
         + wait     * WAITINGTIME[5,10]         ? waiting time in minutes
         + trans    * TRANSFERS[0,1]            ? number of transfers
         + seating  * SEAT[1,0]                 ? seating availability
         + fare     * TICKETPRICE[2,3,4]        ? train fare in dollars
         /
U(bike)  = tt_bike  * TRAVELTIME_BIKE[30,40,50] ? bike travel time in minutes
$
```

Script 3.6: Design for labelled experiment

```
;model:
U(radiotherapy) = side_r.effects * SIDE_EFFECTS[0,1,2]
                + cure_r         * PROB_CURE[0.3,0.5,0.7,0.9]
                /
U(surgery)      = con_s
                + side_s.effects * SIDE_EFFECTS
                + cure_s         * PROB_CURE
                /
U(neither)      = con_n
```

In labelled experiments, it is also possible to have entirely different attributes across alternatives. An example is shown in Script 3.6 where mode choice alternatives `car`, `train`, and `bike` have different utility functions with different attributes.

Finally, if an attribute has many equally spaced levels, then Ngene provides a shortcut to define levels. For example, `[6:20:2]` is automatically replaced in Ngene with levels `[6,8,10,12,14,16,18,20]`. The format of this shortcut is as follows: [*low* : *high* : *step*], where *low* is the lowest level, *high* is the highest level, and *step* is the difference between consecutive levels.

3.8 Defining attribute level preference order

As mentioned in Section 3.2, generating experimental designs for choice experiments with unlabelled alternatives may require additional checks to avoid undesirable choice tasks. Choice tasks with a strictly dominant alternative are among the most common undesirable choice tasks. To detect and avoid such choice tasks, it is necessary to specify the preference order of the levels of each attribute in the model specification in the Ngene script. The definition of a strictly dominant alternative is given in Section 1.6. An example is shown in Figure 3.5, where Laptop A is a strictly dominant alternative, since this laptop has a faster processor and more hard-disk storage at a lower price, and hence all agents are expected to choose Laptop A and make no trade-offs on any of the attributes.

You are looking to buy a new laptop *for at home*. Which of the following laptops would you prefer?	
Laptop A	Laptop B
Intel Core i7 processor 1 TB hard-disk drive $1500	Intel Core i3 processor 256 GB hard-disk drive $2100
◉	○

Figure 3.5: Laptop choice task with strictly dominant alternative

Note that strictly dominant alternatives are not a major concern in labelled experiments and, by definition, cannot occur when attributes are different across alternatives such as in Script 3.6. Therefore, defining the preference order of attribute levels is typically only needed in unlabelled experiments. Further, we remind the user that undesirable choice tasks cannot be avoided in orthogonal designs, and hence defining attribute level preference order is only advised when generating full factorial, random, or efficient designs.

The preference order of the attribute levels can be defined in Ngene by setting priors for each attribute in the utility functions in the `model` property. If no prior information is specified, then Ngene defaults to zero priors, meaning that no information is available, not even the sign of the parameter or the preference order of the attribute levels. We refer to Section 1.3 for more information about different types of priors. In this section, we consider local noninformative priors where only the preference order of attribute levels is known. Other types of prior are discussed when generating efficient designs in Chapter 5.

Priors in Ngene are indicated by a value in square brackets directly after the parameter string in the utility functions of the `model` property. To indicate the preference order of the attribute levels, this value consists of a plus (`+`) to indicate a positive relationship between the attribute levels and utility, or a minus (`-`) to indicate a negative relationship. If no prior is defined, as in all previous scripts in this chapter, then no a priori preference order for attribute levels is assumed.

The utility functions in Script 3.7 illustrate how to define priors to indicate preference orders of the attributes. Since utility is expected to increase with increasing hard disk storage, a positive relationship is indicated in the prior for parameter `stor`. To indicate that higher price levels result in lower utility, a negative relationship is indicated in the prior for parameter `cost`. Furthermore, to indicate that a Core i5 processor (level 1) is preferred over a Core i3 processor (level 0), and a Core i7 processor (level 2) is preferred over a Core i5 processor (level 1), and a Core i9 processor (level 3) is preferred over a Core i7 processor (level 2), a positive relationship between utility and processor level is indicated for dummy parameters `proc`. The preference order is based on the number used for each level, so `proc.dummy[+] * PROCESSOR[2,3,0,1]` would indicate the same preference order of the processor levels, where larger numbers indicate more utility.

Note that prior information does not need to be repeated for `laptopB` since the parameter names are generic across the two alternatives, and hence the priors for parameters in `laptopB` automatically take on the same value.

Script 3.7 generates a full factorial design that includes only 2,160 choice tasks, much less compared to the 4,032 choice tasks generated when no prior information was available. In other words, many undesirable choice tasks with strictly dominant alternatives have now automatically been removed from the design.

```
design  ? Laptop choice example
;alts = (laptopA, laptopB)
;fact   ? implicitly assumes that rows = all
;model:
U(laptopA, laptopB)  = proc.dummy[+] * PROCESSOR[0,1,2,3]
                     + stor[+]       * STORAGE[256,512,1024,2048]
                     + cost[-]       * PRICE[1200,1500,1800,2100]
? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7),  3(Core i9)
? STORAGE:    256 GB,     512 GB,     1024 GB,     2048 GB
? PRICE:      $1200,      $1500,      $1800,       $2100
$
```

Script 3.7: Indicating preference order to exclude strictly dominant alternatives

Sometimes not all attributes exhibit an obvious preference order for their levels. An example is shown in the syntax below, where the screen size is added to the utility function as a dummy-coded attribute with levels 13″, 15″, and 17″. Some agents may prefer a small screen size for portability reasons, while other agents may prefer a large screen size for productivity reasons. In this case, we do not indicate a preference order for the levels of attribute `SCREENSIZE` and hence Ngene ignores this attribute when checking for strictly dominant alternatives.

```
;model:
U(laptopA, laptopB)  = proc.dummy[+] * PROCESSOR[0,1,2,3]
                     + stor[+]       * STORAGE[256,512,1024,2048]
                     + cost[-]       * PRICE[1200,1500,1800,2100]
                     + size.dummy    * SCREENSIZE[13,15,17]  ? screen size in inches
```

Not all unlabelled experiments suffer from strictly dominant alternatives. In Script 3.8 we consider the choice to buy a new car based on the type of engine (diesel, petrol, LPG, or electric), the colour of the car (white, red, black, or blue), the type of transmission (manual or automatic) and the price ($20,000, $25,000 or $30,000). Only price has a obvious preference order, whereas preferences towards the other attributes will vary from person to person. Strictly dominant alternatives only occur when the levels of all (or most) attributes have a clear preference order, hence in this case there is no need to set priors to avoid dominant alternatives (although setting informative priors will assist in generating more efficient designs, see Chapter 5).

3.9 Interaction effects

In addition to main effects, utility functions can also contain interaction effects that describe the (additional) impact on utility of specific combinations attribute levels. Although the specification of interaction effects has no impact on the generation of a full factorial or random design, it does impact the generation of orthogonal designs (where Ngene will aim to minimise correlations) and efficient designs (where Ngene will optimise the design for the estimation of the parameters of the interaction effects as well). Not considering interaction effects during the design phase does not mean that these effects cannot be estimated later on; in most cases, a sufficiently large design size will allow the estimation of many (but perhaps not all) interaction effects. However, to guarantee that a certain interaction effect can be estimated after data collection, it may be wise to include the interaction effect in the utility function during the design phase.

To illustrate, the utility functions in the syntax below include a multiplication of the quantitative variable `STORAGE` with the quantitative variable `PRICE`. If a laptop in a certain choice task has a hard-disk with 256 GB and a price of $1500, then the interaction term becomes $256 \times 1500 = 384,000$,

```
design  ? Car choice example
;alts = (car1, car2)
;rows = 60
;block = 5
;rand
;model:    ? using estimation coding
U(car1)    = eng.dummy   * ENGINE[0,1,2,3]
           + col.dummy   * COLOUR[0,1,2,3]
           + trans.dummy * TRANSMISSION[0,1]
           + cost        * PRICE[20000,25000,30000]
           /
U(car2)    = eng         * ENGINE
           + col         * COLOUR
           + trans       * TRANSMISSION
           + cost        * PRICE
? ENGINE:       0(Diesel),  1(Petrol),  2(LPG),    3(Electric)
? COLOUR:       0(White),   1(Red),     2(Black),  3(Blue)
? TRANSMISSION: 0(Manual),  1(Automatic)
? PRICE:        $20,000,    $25,000,    $30,000
$
```

Script 3.8: No clear preference order for most attributes

which is multiplied with the parameter `stor_x_cost` to compute the utility of this interaction effect. A positive parameter associated with this interaction effect can be interpreted as agents becoming less price sensitive when hard-disk storage increases, or can be interpreted as agents attaching more utility to hard-disk storage at higher price levels.

```
;model:
U(laptopA, laptopB)  = proc.dummy[+] * PROCESSOR[0,1,2,3]
                     + stor[+]       * STORAGE[256,512,1024,2048]
                     + cost[-]       * PRICE[1200,1500,1800,2100]
                     + stor_x_cost   * STORAGE * PRICE
                     + proc0_x_cost  * PROCESSOR.level[0] * PRICE
                     + proc1_x_cost  * PROCESSOR.level[1] * PRICE
                     + proc2_x_cost  * PROCESSOR.level[2] * PRICE
? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7),  3(Core i9)
? STORAGE:    256 GB,     512 GB,     1024 GB,     2048 GB
? PRICE:      $1200,      $1500,      $1800,       $2100
```

In the syntax above, the qualitative processor attribute has also interacted with the quantitative price attribute. Since the processor type is dummy coded, separate multiplications need to be made. If an attribute has L levels, then only $L-1$ levels can appear in an interaction. In this case, we have interacted levels 0, 1, and 2 with price, but we could also have interacted levels 1, 2, and 3 with price, or levels 0, 2 and 3 with price. The suffix `.level[x]` after an attribute returns a value of one if this attribute has level `x`, and zero otherwise. For example, if a laptop in a certain choice task has a Core i7 processor and a price of $1200, then the interaction term `PROCESSOR.level[2] * PRICE` is equal to 1×1200 (while the other interaction effects between processor and price are equal to zero), which is multiplied with the parameter `proc2_x_cost` to obtain the utility of this interaction effect.

Ngene can check for strictly dominant alternatives based on main effects alone; therefore, it is not necessary to indicate a preference order of interaction effects (`+` or `-`). For example, no prior was specified for `stor_x_cost`.

One should be aware that adding interactions between two dummy or effects-coded variables may require a large number of interaction terms and parameters. For example, in the syntax below we now also dummy-coded the storage attribute and added nine interaction terms between levels of dummy coded attributes `PROCESSOR` and `STORAGE`.

```
;model:
U(laptopA, laptopB)
   = proc.dummy[+]    * PROCESSOR[0,1,2,3]
   + stor.dummy[+]    * STORAGE[256,512,1024,2048]
   + cost[-]          * PRICE[1200,1500,1800,2100]
   + proc0_x_stor256  * PROCESSOR.level[0] * STORAGE.level[256]
   + proc0_x_stor512  * PROCESSOR.level[0] * STORAGE.level[512]
   + proc0_x_stor1024 * PROCESSOR.level[0] * STORAGE.level[1024]
   + proc1_x_stor256  * PROCESSOR.level[1] * STORAGE.level[256]
   + proc1_x_stor512  * PROCESSOR.level[1] * STORAGE.level[512]
   + proc1_x_stor1024 * PROCESSOR.level[1] * STORAGE.level[1024]
   + proc2_x_stor256  * PROCESSOR.level[2] * STORAGE.level[256]
   + proc2_x_stor512  * PROCESSOR.level[2] * STORAGE.level[512]
   + proc2_x_stor1024 * PROCESSOR.level[2] * STORAGE.level[1024]
```

The syntax below also includes an interaction term between `PROCESSOR` and `STORAGE`, but instead of considering dummy variables, the interaction term treats both attributes as quantitative, e.g., it computes 3×2024 if both attributes have the largest level. In this case, only a single parameter, `proc_x_stor`, is estimated for the interaction. While this may be appropriate in some cases, in most cases with qualitative variables it is required to interaction individual attribute levels as in the syntax above.

```
;model:
U(laptopA, laptopB)  = proc.dummy[+]    * PROCESSOR[0,1,2,3]
                     + stor.dummy[+]    * STORAGE[256,512,1024,2048]
                     + cost[-]          * PRICE[1200,1500,1800,2100]
                     + proc_x_stor      * PROCESSOR * STORAGE
```

In some cases, an attribute may be added only as an interaction effect in the utility function and not as a main effect. This can be easily done for quantitative attributes, as shown in the syntax below where `STORAGE` appears only in an interaction with `PRICE`. The parameter `cost_x_stor` expresses to what extent the price sensitivity depends on storage capacity. However, adding a qualitative attribute such as `PROCESSOR` only as an interaction effect is more complex as it first needs to be defined as a dummy or effect-coded variable before it can be used in an interaction. This process is explained in Section 7.2.

```
;model:
U(laptopA, laptopB)  = proc.dummy[+]    * PROCESSOR[0,1,2,3]
                     + cost[-]          * PRICE[1200,1500,1800,2100]
                     + cost_x_storage   * PRICE * STORAGE[256,512,1024,2048]
```

Finally, the syntax below shows an example of utility functions in a labelled experiment where an interaction effect is only added for the train alternative. This interaction term is added to capture the effect that agents may become more sensitive to travel time inside the train if no seat is available, i.e., the value of the parameter `tt_x_noseat` is expected to be negative. Recall that strictly dominant alternatives by definition do not occur in labelled experiments, and hence it is not necessary to indicate the preference order of attribute levels.

```
;model:
U(car)   = con_c
         + tt_car       * TRAVELTIME_CAR[5,10,15]
         + fuel         * FUELCOST[1,2,3]
         + park         * PARKING[1,2]
         /
U(train) = con_t
         + access       * ACCESSTIME[1,5,10]
         + tt_train     * INVEHICLETIME[10,15,20]  ? in-vehicle travel time in minutes
         + wait         * WAITINGTIME[5,10]
         + trans        * TRANSFERS[0,1]
         + seat.dummy   * SEAT[1,0]                ? 1: seat available, 0: not available
         + fare         * TICKETPRICE[2,3,4]
         + tt_x_noseat  * INVEHICLETIME * SEAT.level[0]
         /
U(bike)  = tt_bike      * TRAVELTIME_BIKE[30,40,50]
```

4

Orthogonal designs

This chapter describes how to generate orthogonal designs in Ngene syntax for both labelled and unlabelled experiments. Orthogonal designs are fractional factorial experimental designs that have a specific structure of attribute level combinations that covers the space spanned by the attribute levels equally in all dimensions. Orthogonal designs are particularly useful when no prior information is available about the parameters, when no other constraints on combinations of attribute levels within a choice task are imposed, and when strictly dominant alternatives are not a concern (e.g., a labelled experiment or an unlabelled experiment where most attribute levels do not have an obvious preference order). In all other cases, an efficient design (see Chapter 5) would be preferred to allow more flexibility in terms of considering attribute level constraints and prior information.

4.1 Orthogonal design generation

In Ngene, you can create orthogonal designs by setting the property `orth`, which takes the place of the property `fact` or `rand` used to generate full factorial designs or random fractional factorial designs, as discussed in the previous chapter.

```
;orth
```

Property `orth` can have an additional value to indicate how the orthogonal design needs to be generated, namely,

- `orth = sim` – generates a simultaneous orthogonal design (default)
- `orth = ood` – generates a sequential optimal orthogonal design
- `orth = seq` or `seq2` – generates a sequential randomised orthogonal design

If no value is specified, then the default is `orth = sim` and Ngene applies a *simultaneous* orthogonal design procedure that locates a design that is orthogonal for all attributes and alternatives simultaneously. A simultaneous orthogonal design is typically used for labelled experiments, but it can also be applied to unlabelled experiments if desired.

For unlabelled experiments with generic alternatives one can specify `orth = ood` to use the method of Street et al. (2005) described in Section 1.5.2 to generate a *sequential* orthogonal design with

```
design  ? labelled car purchase example
;alts = car, no_car
;rows = 16
;orth
;model:
U(car)    = con_c
          + eng.dummy  * ENGINE[0,1,2,3]
          + col.dummy  * COLOUR[0,1,2,3]
          + type.dummy * TYPE[0,1,2,3]
? ENGINE: 0(Diesel), 1(Petrol), 2(Hybrid),    3(Electric)
? COLOUR: 0(White),  1(Red),    2(Black),     3(Blue)
? TYPE:   0(Sedan),  1(Coupe),  2(Hatchback), 3(Station wagon)
$
```

Script 4.1: Orthogonal design

minimum overlap. This procedure ensures minimum attribute level overlap across alternatives by sequentially applying generators to create attribute levels for each consecutive alternative based on the attribute levels of the first alternative. Although previous versions of Ngene reported D-efficiency percentages for optimal orthogonal designs, these are no longer reported since these percentages can be misleading as assumptions underlying these computations are generally not met in practice (e.g., these percentages are invalid when using dummy or effects coding, see Section 1.5.2) and these percentages can only be computed for some orthogonal designs.

An alternative method for generating sequential orthogonal designs is using `orth = seq` or `orth = seq2`, where the former is used in unlabelled experiments and the latter can be used in labelled experiments where only some attributes are the same across alternatives. When using `seq`, the attribute levels of each consecutive alternative are generated by randomising the order of the profiles in the first alternative across rows. In most cases, one would prefer to use `ood` over `seq`, but `seq` may be desirable if one prefers some attribute level overlap or if one prefers not to impose strong correlations between alternatives by applying generators. Using `orth = seq2` generates only the levels of *generic* attributes sequentially across the alternatives by randomising the levels, while the levels of other attributes are generated to be simultaneously orthogonal.

An example is shown in Script 4.1 where alternative `car` is characterised by three attributes, namely, engine type, colour and car type, while `no_car` is an opt-out alternative without any attributes.

The resulting orthogonal design with 16 choice tasks is shown in Figure 4.1 and its first choice task with profile (0,3,3) is shown in Figure 4.2. Orthogonal designs are not unique; there may exist multiple orthogonal arrays for the same design dimensions. Further, one can create different choice tasks with the same orthogonal array by assigning design codes to different attribute levels (e.g., instead of using design code 0 for diesel engine, one could assign it to petrol engine). Ngene simply reports the first orthogonal design it finds and will not report all. If one is interested in the most efficient orthogonal design, then property `orth` can be used in conjunction with property `eff` as explained in Chapter 5.

In Chapter 3 a depiction of a random fractional factorial design was shown in Figure 3.4. As can be seen, the orthogonal design in Figure 4.1 has a very specific structure, namely, each combination of attribute levels appears the same number of times (in this case exactly once). Graphically, this means that looking at the cube from any side one would see exactly 16 orbs.

Script 4.2 generates a sequential optimal orthogonal design for an unlabelled car purchase choice experiment with two alternatives using `orth = ood`. The resulting experimental design is shown in Table 4.1, where the attribute levels for the first alternative, `car1`, are based on an orthogonal

	Car			No car
Choice task	Engine	Colour	Type	
1	0	3	3	–
2	1	3	2	–
3	2	3	1	–
4	3	3	0	–
5	0	2	2	–
6	1	2	3	–
7	2	2	0	–
8	3	2	1	–
9	0	1	1	–
10	1	1	0	–
11	2	1	3	–
12	3	1	2	–
13	0	0	0	–
14	1	0	1	–
15	2	0	2	–
16	3	0	3	–

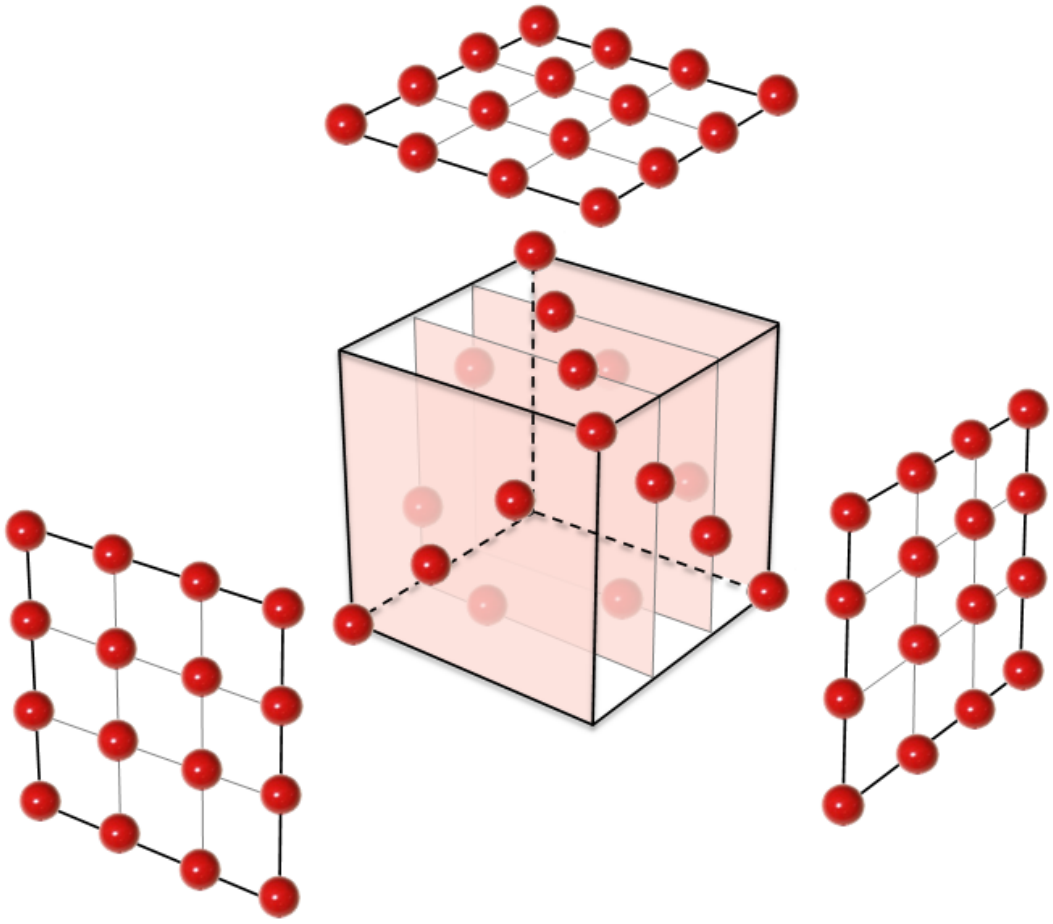


Figure 4.1: Orthogonal design and graphical depiction

Consider purchasing a new car of the same brand and price as your current car. Would you purchase this car with the following characteristics?	
Diesel engine Blue colour Station wagon type	
○ Yes, I would buy	◉ No thanks

Figure 4.2: Choice task for labelled car purchase example

```
design  ? unlabelled car purchase example
;alts = (car1, car2)
;rows = 16
;orth = ood
;model:
U(car1, car2)   = eng.dummy  * ENGINE[0,1,2,3]
                + col.dummy  * COLOUR[0,1,2,3]
                + type.dummy * TYPE[0,1,2,3]
? ENGINE: 0(Diesel), 1(Petrol), 2(Hybrid),    3(Electric)
? COLOUR: 0(White),  1(Red),    2(Black),     3(Blue)
? TYPE:   0(Sedan),  1(Coupe),  2(Hatchback), 3(Station wagon)
$
```

Script 4.2: Sequential orthogonal design

array, and the attribute levels for the second alternative, `car2`, are derived from the attribute levels of `car1` using generator 111, see also Section 1.5.2. The first choice task with profiles $(0, 0, 0)$ and $(1, 1, 1)$ is shown in Figure 4.3.

Replacement of `orth = ood` with `orth = seq` in Script 4.2 would generate a sequential randomised orthogonal design as shown in Table 4.2. The attribute levels for the first alternative, `car1`, are the same in this case as in Table 4.1. However, the profiles for the second alternative, `car2`, are derived from the profiles of `car1` by randomising their order. For example, the profile in row 1 for `car1`, $(0, 0, 0)$, appears as a profile in row 14 for `car2`, and the profile in row 2 for `car1` appears as a profile in row 15 for `car2`.

Consider purchasing a new car of the same brand and price as your current car. Which car would you prefer to purchase?	
Diesel engine White colour Sedan	Petrol engine Red colour Coupe
○ Car 1	◉ Car 2

Figure 4.3: Choice task for unlabelled car purchase example

	Car 1			Car 2		
Choice task	Engine	Colour	Type	Engine	Colour	Type
1	0	0	0	1	1	1
2	1	1	0	2	2	1
3	2	2	0	3	3	1
4	3	3	0	0	0	1
5	1	0	1	2	1	2
6	0	1	1	1	2	2
7	3	2	1	0	3	2
8	2	3	1	3	0	2
9	2	0	2	3	1	3
10	3	1	2	0	2	3
11	0	2	2	1	3	3
12	1	3	2	2	0	3
13	3	0	3	0	1	0
14	2	1	3	3	2	0
15	1	2	3	2	3	0
16	0	3	3	1	0	0

Table 4.1: Sequential optimal orthogonal design for unlabelled car purchase example

	Car 1			Car 2		
Choice task	Engine	Colour	Type	Engine	Colour	Type
1	0	0	0	2	0	2
2	1	1	0	3	0	3
3	2	2	0	1	3	2
4	3	3	0	0	2	2
5	1	0	1	2	2	0
6	0	1	1	3	1	2
7	3	2	1	1	2	3
8	2	3	1	0	1	1
9	2	0	2	3	3	0
10	3	1	2	2	3	1
11	0	2	2	1	0	1
12	1	3	2	3	2	1
13	3	0	3	0	3	3
14	2	1	3	0	0	0
15	1	2	3	1	1	0
16	0	3	3	2	1	3

Table 4.2: Sequential randomised orthogonal design for unlabelled car purchase example

4.2 Existence of orthogonal designs

Orthogonal fractional factorial designs only exist for specific design sizes (i.e., numbers of rows) depending on the number of attributes and the number of levels of each attribute and may not exist at all for certain design dimensions. In Ngene, a user can simply specify the desired number of rows, and Ngene will try to locate an orthogonal design of that size. If it does not exist, Ngene automatically increases the number of rows and reports the first orthogonal design that it finds (if it exists). If an orthogonal design does not exist for any design size, Ngene will report "No design found" and the user will need to change the number of levels of some of the attributes. Alternatively, one can use a full factorial design (see Chapter 3), which is orthogonal by definition but typically has a very large design size, or switch to a more flexible design type such as an efficient design (see Chapter 5).

For example, if the user specified `rows = 14` in line 3 of Script 4.1 then Ngene reports the following message: "Could not locate design in 14 rows. Switching to design with 16 rows". If the attribute colour has seven levels, i.e., if the user specified `COLOUR[0,1,2,3,4,5,6]` in Script 4.1, then Ngene increases the design size to 28 rows. If in addition the number of car types is increased to seven via `TYPE[0,1,2,3,4,5,6]`, then an orthogonal fractional factorial design no longer exists. Generally, orthogonal designs tend to be more feasible for choice experiments where the majority of attributes have only a few levels, and the number of levels is predominantly the same across different attributes.

If a choice experiment contains generic attributes, then a design generated using `orth = seq` or `orth = ood` is more likely to exist and have a smaller number of rows than a design generated using `orth = sim` since it requires only orthogonality across attributes within each alternative and not necessarily across alternatives. For example, replacing `orth = ood` with `orth = sim` in Script 4.2 would generate a design of size 24 instead of 16 rows.

4.3 Blocking of orthogonal designs

Orthogonal designs can be large in terms of the number of rows, hence in most cases it is necessary to block the design. Adding `block = 2` to Script 4.1 results in the design shown in Table 4.3. For orthogonal designs, Ngene applies an *orthogonal blocking* strategy such that attribute level balance is achieved within each block. In Table 4.3 it can be observed that attribute level 0 appears exactly twice within each block, and the same holds for every other level.

Script 4.3 illustrates another example with two labelled alternatives, `car` and `bus`, each having different attributes. The smallest simultaneous orthogonal design that exists has 18 rows; see Table 4.4. In this case, we have blocked the design into 3 equal parts, where each block satisfies attribute level balance.

4.4 Interactions in orthogonal designs

Section 3.9 discussed how to include interactions effects in the model specification. Although the specification of interaction effects does not affect the generation of full factorial and random designs, it does influence the generation of orthogonal designs when using `orth = sim` or `orth = seq`. Note that interaction effects are not considered when generating sequential optimal orthogonal designs using `orth = ood` due to the way such designs are generated.

If the analyst knows in advance which interaction effects will likely be included in the final model, then Ngene can make sure that the interaction effects are not perfectly correlated with the main effects, such that their associate parameters can be estimated. Although ideally each interaction effect is orthogonal with all main effects and all other interaction effects, this is in most cases not possible.

		Car			No car
Choice task	Block	Engine	Colour	Type	
1	1	0	0	0	–
2	1	2	0	2	–
3	1	3	3	0	–
4	1	1	3	2	–
5	1	1	2	3	–
6	1	3	2	1	–
7	1	2	1	3	–
8	1	0	1	1	–
9	2	2	2	0	–
10	2	0	2	2	–
11	2	1	1	0	–
12	2	3	1	2	–
13	2	3	0	3	–
14	2	1	0	1	–
15	2	0	3	3	–
16	2	2	3	1	–

Table 4.3: Blocked orthogonal experimental design

```
design  ? mode choice example
;alts = car, bus
;rows = 18
;block = 3
;orth = sim
;model:
U(car) = con_c
       + tt_car * TIME[15,20,25]      ? travel time (minutes)
       + toll   * TOLL[2,3,4]         ? toll costs (dollars)
       /
U(bus) = travel * INVEHTIME[20,25,30] ? in-vehicle time (minutes)
       + wait   * WAITTIME[2,4,6]     ? waiting time (minutes)
       + trans  * TRANSFERS[0,1]      ? number of transfers
       + fare   * FARE[1,2,3]         ? bus fare (dollars)
$
```

Script 4.3: Simultaneous orthogonal design with three blocks

		Car		Bus			
Choice task	Block	Time	Toll	In-veh time	Wait time	Transfers	Fare
1	1	15	2	20	2	0	1
2	1	20	3	25	4	0	2
3	1	25	4	30	6	0	3
4	1	20	3	30	6	1	1
5	1	25	4	20	2	1	2
6	1	15	2	25	4	1	3
7	2	25	3	25	2	0	1
8	2	15	4	30	4	0	2
9	2	20	2	20	6	0	3
10	2	25	2	30	4	1	1
11	2	15	3	20	6	1	2
12	2	20	4	25	2	1	3
13	3	20	4	20	4	0	1
14	3	25	2	25	6	0	2
15	3	15	3	30	2	0	3
16	3	15	4	25	6	1	1
17	3	20	2	30	2	1	2
18	3	25	3	20	4	1	3

Table 4.4: Blocked simultaneous orthogonal design for mode choice example

Instead, Ngene considers all possible orthogonal designs and selects the design that minimises the correlations between the interaction effects and the main effects that are specified in the utility functions via the `model` property.

Often, only at the model estimation stage it is known which interaction effects are relevant and no interaction effects are specified in the utility functions at the experimental design stage. In most cases, one will still be able to estimate most, but perhaps not all, interaction effects. To increase the likelihood that interaction effects can be estimated without adding all interaction effects a priori to utility functions, one could make all interaction effects uncorrelated with all main effects[1] by adding the following property to the script:

```
;foldover
```

This property appends the design with a mirror-image foldover. To illustrate, consider again Script 4.1 and let us add the `foldover` property to the syntax. Instead of the original 16 rows, the design now consists of 32 rows where the first 16 rows are the same as in Figure 4.1 while the next 16 rows are a mirror image of the previous 16 rows in which the attribute levels have been re-labelled in reverse, namely $0 \rightarrow 3$, $1 \rightarrow 2$, $2 \rightarrow 1$, and $3 \rightarrow 0$, see Table 4.5.

Although a design with a mirror-image foldover has twice as many choice tasks, the resulting design is readily blocked into two blocks, as indicated in Table 4.5. Therefore, the foldover only increases the size of the experimental design and does not increase the number of choice tasks given to each agent.

[1] Note that this does not mean that each interaction effect is uncorrelated with all other interaction effects. Therefore, this still does not guarantee that one can estimate all parameters of all interaction effects.

		Car			No car
Choice task	Block	Engine	Colour	Type	
1	1	0	3	3	–
2	1	1	3	2	–
3	1	2	3	1	–
4	1	3	3	0	–
5	1	0	2	2	–
6	1	1	2	3	–
7	1	2	2	0	–
8	1	3	2	1	–
9	1	0	1	1	–
10	1	1	1	0	–
11	1	2	1	3	–
12	1	3	1	2	–
13	1	0	0	0	–
14	1	1	0	1	–
15	1	2	0	2	–
16	1	3	0	3	–
17	2	3	0	0	–
18	2	2	0	1	–
19	2	1	0	2	–
20	2	0	0	3	–
21	2	3	1	1	–
22	2	2	1	0	–
23	2	1	1	3	–
24	2	0	1	2	–
25	2	3	2	2	–
26	2	2	2	3	–
27	2	1	2	0	–
28	2	0	2	1	–
29	2	3	3	3	–
30	2	2	3	2	–
31	2	1	3	1	–
32	2	0	3	0	–

Table 4.5: Orthogonal design with mirror-image foldover

```
design  ? Laptop choice example
;alts = (laptopA, laptopB)
;rows = 9
;orth = ood
;model:
U(laptopA, laptopB) = proc.dummy[+] * PROCESSOR[0,1,2]
                    + stor.dummy[+] * STORAGE[0,1,2]
                    + cost[-]       * PRICE[1500,1800,2100]
? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7)
? STORAGE:    0(256 GB),  1(512 GB),  2(1 TB)
? PRICE:      $1500,      $1800,      $2100
$
```

Script 4.4: Orthogonal design cannot avoid strictly dominant alternatives

	Laptop A			Laptop B		
Choice task	Processor	Storage	Price	Processor	Storage	Price
1	0 (Core i3)	0 (256 GB)	0 ($1,500)	1 (Core i5)	1 (512 GB)	1 ($1,800)
2	1 (Core i5)	1 (512 GB)	0 ($1,500)	2 (Core i7)	2 (1 TB)	1 ($1,800)
3	2 (Core i7)	2 (1 TB)	0 ($1,500)	0 (Core i3)	0 (256 GB)	1 ($1,800)
4	1 (Core i5)	0 (256 GB)	1 ($1,800)	2 (Core i7)	1 (512 GB)	2 ($2,100)
5	2 (Core i7)	1 (512 GB)	1 ($1,800)	0 (Core i3)	2 (1 TB)	2 ($2,100)
6	0 (Core i3)	2 (1 TB)	1 ($1,800)	1 (Core i5)	0 (256 GB)	2 ($2,100)
7	2 (Core i7)	0 (256 GB)	2 ($2,100)	0 (Core i3)	1 (512 GB)	0 ($1,500)
8	0 (Core i3)	1 (512 GB)	2 ($2,100)	1 (Core i5)	2 (1 TB)	0 ($1,500)
9	1 (Core i5)	2 (1 TB)	2 ($2,100)	2 (Core i7)	0 (256 GB)	0 ($1,500)

Table 4.6: Optimal orthogonal design for laptop choice example

4.5 Limitations of orthogonal designs

As pointed out in Section 3.2, Ngene does not check for undesirable choice tasks when generating orthogonal designs, even if the user groups generic alternatives together in the script and even if a preference order is specified for attribute levels. Ngene can only avoid undesirable choice tasks when generating full factorial, random, and efficient designs, since these designs offer more flexibility. Undesirable choice tasks are not an issue in Script 4.2 since the attributes levels do not have an obvious preference order.

Consider the laptop choice example in Script 4.4 where the attribute levels do have a preference order where higher processor and storage levels and lower price levels are preferred. The resulting design is shown in Table 4.6, where strictly dominant alternatives are present in Choice tasks 3 and 8 in Table 4.6.

Another limitation is that – due to the strict nature of orthogonality – orthogonal designs do not allow the flexibility of imposing constraints (prohibitions) on combinations of attribute levels as discussed in Chapter 6. Therefore, orthogonal designs cannot rule out unrealistic combinations of attribute levels. While in practice this is often resolved by simply removing choice tasks from the design that have unrealistic profiles, the resulting design is no longer orthogonal.

5

Efficient designs

This chapter describes the generation of efficient designs. Efficient designs assume prior information about the model to generate experimental designs that capture more (Fisher) information, resulting in more reliable parameter estimates in model estimation (i.e., smaller standard errors). Compared to orthogonal designs, efficient designs have greater flexibility; they can avoid strictly dominant alternatives and allow imposing constraints (prohibitions) on attribute level combinations.

Efficient designs are optimised for a specific model as defined by the *model type*, *utility function specification* (including consideration of alternative-specific constants, interaction effects, nonlinear effects, dummy or effects coding, etc.), and *priors* (best guesses for the parameter values). The model specified during the design phase may differ from the final model that is estimated, but the closer the 'estimation' model resembles the 'design' model, the more efficiency is retained.

5.1 Efficient design generation

An efficient design can be generated in Ngene by specifying the property `eff`, replacing the properties `fact`, `rand`, and `orth` discussed in Chapters 3 and 4, respectively.

```
;eff
```

Property `eff` can have additional values to indicate for which *model type*, *efficiency criterion*, *efficiency statistic*, and *decision rule* the design is optimised. If no values are specified, then by default it assumes the following:

```
;eff = (mnl,d,fixed,rum)
```

where `mnl` refers to the multinomial logit model as the default model type, `d` refers to the D-error as the default efficiency criterion, `fixed` refers to the evaluation of the efficiency criterion at the single midpoint as the default statistic, and `rum` refers to the maximisation of random utility as the default decision rule.

5.1.1 Model type

Ngene can generate efficient designs for the following model types:

- `mnl` – multinomial logit (default)
- `rp` – random parameter logit
- `ec` – error component logit
- `rpec` – random parameter and error component logit
- `rppanel` – panel random parameter logit
- `ecpanel` – panel error component logit
- `rpecpanel` – panel random parameter and error component logit

The multinomial logit model is often considered the workhorse of discrete choice models due to its widespread application, and it is the default model considered in Ngene. The other model types listed above are variations of a *mixed logit* model. Data collected based on design that was optimised for the multinomial logit model can also be used for estimating mixed logit models (although there may be a slight loss of efficiency). As mentioned in Section 1.5, generating efficient designs specifically for mixed logit models is not only computationally very demanding, but is also practically challenging since obtaining reliable priors for random parameters and error components is often difficult. Therefore, in most cases, it is recommended to optimise for a multinomial logit model, even if one intends to estimate a mixed logit model during the estimation phase.

In this chapter, we mainly focus on optimising designs for a *multinomial logit* (`mnl`) model. The generation of designs for mixed logit models is discussed in Section 5.7.

5.1.2 Efficiency criteria

Ngene can optimise a design according to several efficiency criteria (see also Section 1.5.1):

- `d` – Minimise D-error (default)
- `a` – Minimise A-error
- `s` – Minimise S-error
- `wtp` - Minimise C-error regarding willingness-to-pay

The D-error is used in almost all studies reported in the literature and is in most cases the best choice. The D-error is the determinant of the covariance matrix, the A-error is the trace of the covariance matrix, and the S-error is the minimum sample size needed to obtain statistically significant parameter estimates.

In all cases, the lower the value of the efficiency criterion, the more efficient the design. Note that efficiency values are only comparable between designs generated under the same model assumptions (model type, utility function specifications, and priors) and are not comparable otherwise. If Ngene returns an *Undefined* value, then this means that the D-, A-, or S-error is *infinite* and the design should *not* be used. This may happen when there is an issue regarding the identifiability of the parameters in the specified utility functions or when constraints imposed on the design create multicollinearity in the data.

The `eff` property does not need to contain all four arguments; it uses the default values for the last arguments if they are omitted. For example, to only change the efficiency criterion while keeping all other default values, one could use the syntax below to generate an S-efficient design assuming `mnl` as model type, `fixed` as efficiency statistic, and `rum` as decision rule.

```
;eff = (mnl,s)
```

The sample size estimates are by default based on a two-sided significance level of 0.05, which corresponds to a *t*-ratio of 1.96. If a different *t*-ratio is desired for sample size calculations, for example, 2.58 for a significance level of 0.01, then the syntax can be adjusted to:

```
;eff = (mnl,s(2.58))
```

Although less common, Ngene can also minimise the C-error based on a willingness-to-pay (WTP) definition in conjunction with the property `wtp`. For example, in Script 5.2 we could use the following syntax to optimise the design specifically to estimate the willingness-to-pay measures, for example `stor/cost` and referred to as `mywtp` (or any other preferred name). It is also possible to optimise for multiple or all WTP values simultaneously using `wtp = mywtp(proc,stor/cost)` or `wtp = mywtp(*/cost)`, respectively. Please refer to the *Syntax Help* in the script editor (see Section 2.2).

```
;eff = (mnl,wtp(mywtp))
;wtp = mywtp(stor/cost)
```

The `eff` property essentially sets the objective function for a mathematical minimisation problem. Instead of optimising only a single efficiency criterion, it is also possible to optimise for two or more criteria simultaneously. For example, the syntax below minimises a weighted sum of the D-error and S-error, whereby the D-error value is multiplied by 50. Appropriate weights for each criterion depend on their relative size (e.g., D-errors are generally much smaller than S-errors and therefore this needs to be accounted for in the weights).

```
;eff = 50*(mnl,d) + (mnl,s)
```

Another option is to add attribute level imbalance as an efficiency criterion. Attribute level imbalance exists if some attribute levels appear less frequently than other levels across the choice tasks in the design. This is an issue that occurs mainly when using the modified Fedorov algorithm; see Section 5.4. Ngene adopts the attribute level imbalance measure proposed by Collins et al. (2014), which equals 0 if the design is perfectly attribute level balanced, and 1 if the design is completely unbalanced (i.e., if only a single level appears in the design for each attribute). The syntax example below simultaneously minimises the D-error and attribute level imbalance assuming a weight of 0.5 for the latter (which again needs to consider the relative size of each efficiency criterion).

```
;eff = (mnl,d) + 0.5*(imbalance)
```

5.1.3 Efficiency statistic

Ngene can optimise design efficiency according to the following statistics:

- `fixed` – Design efficiency based on local prior or mid-point from prior distributions (default)
- `mean` – Average design efficiency across draws from prior distributions
- `median` – Median design efficiency across draws from prior distributions

To understand these statistics, we need to discuss the priors further. Priors are best guesses of the values of the parameters in the choice model. Such priors are useful when generating efficient designs to create choice tasks that capture maximum information from agents. *Noninformative priors* can be used if minimal or no information is available about the parameter values, whereas *informative*

priors can be used when more information is available, see also Table 1.3. Prior information can come in the form of fixed values, referred to as *local priors*, or as probability distributions to indicate uncertainty about the true values of the parameters, referred to as *Bayesian priors*.

If Bayesian priors are specified, then each design can be evaluated over a large number of prior values drawn from the specified prior distributions, whereby each draw yields a specific D-error or other efficiency criterion. When specifying `mean` as the efficiency statistic, Ngene will optimise the design based on the average efficiency across all draws from the specified prior distributions. An alternative statistic is `median`, which also considers the efficiency of the design across all draws from the prior distributions but instead optimises for the median value. In most cases, `mean` is preferred over `median`, but `median` may be a better choice when some prior distributions are very wide[1] (i.e., a large standard deviation). When a prior distribution is wide, extremely large values could be drawn from the distribution, producing outliers when evaluating design efficiency. Such outliers negatively affect the calculation of the mean, but not the median.

If local priors are specified, then each design is evaluated only once under this fixed set of prior values, resulting in a single value for the D-error or other efficiency criterion. Therefore, if local priors are specified, only `fixed` can be used as efficiency statistic in the `eff` property. If Bayesian priors are specified in the script but one prefers to generate an efficient design based on local priors, then using `fixed` means that Ngene assumes local priors based on the midpoint of each Bayesian prior.

As discussed in Section 5.1.2, Ngene allows flexibility in specifying multiple criteria for generating efficient design using the `eff` property, for example, the syntax below optimises the design for a mix of Bayesian and local efficiency statistics.

```
;eff = 0.5*(mnl,d,mean) + 2*(mnl,d,fixed)
```

5.1.4 Decision rule

The following decision rules can be specified in Ngene (see also Section 1.1):

- `rum` – Random utility maximisation (default)
- `rrm` – Random regret minimisation

If no decision rule is specified, Ngene assumes that the efficient design is optimised under the assumption of random utility maximisation.

It is only possible to consider `rrm` as a decision rule in an unlabelled experiment where all utility functions are identical. Furthermore, since `rum` produces the same model as `rrm` if there are only two alternatives, using `rrm` is only useful when considering three or more alternatives.

If desired, designs can be optimised under both decision rules simultaneously by combining two criteria as shown in the following syntax (see also Section 5.1.2).

```
;eff = 2*(mnl,d,fixed,rum) + (mnl,d,fixed,rrm)
```

[1] For instance, this may happen when priors are taken from a pilot study whereby some parameter estimates are unreliable as indicated with large standard errors

```
design  ? labelled car purchase example
;alts = car, no_car
;rows = 16
;eff
;model:
U(car)    = con_c
          + eng.dummy  * ENGINE[0,1,2,3]
          + col.dummy  * COLOUR[0,1,2,3]
          + type.dummy * TYPE[0,1,2,3]
? ENGINE: 0(Diesel), 1(Petrol), 2(Hybrid),    3(Electric)
? COLOUR: 0(White),  1(Red),    2(Black),     3(Blue)
? TYPE:   0(Sedan),  1(Coupe),  2(Hatchback), 3(Station wagon)
$
```

Script 5.1: Efficient design

5.2 Specifying noninformative priors

If one is unsure what prior values to use, utilising zero or near-zero noninformative priors is always a safe strategy. Efficient designs are generally not orthogonal, but under certain conditions[2] efficient designs will be orthogonal or near-orthogonal when all attributes are dummy or effects coded and zero priors are used.

For example, consider again the labelled car purchase example in Script 5.1, which is the same as Script 4.1, where `orth` is replaced by `eff`. The resulting efficient design with 16 choice tasks is depicted in Figure 5.1. This design is in fact orthogonal since each pair of attribute levels appears exactly once, which means that one would see exactly 16 orbs when looking at the cube from any side. Although the design in Figure 4.1 is also orthogonal, the design in Figure 5.1 has a lower D-error (assuming a multinomial logit model) and is therefore more D-efficient. Therefore, if no prior information is available (e.g., in a pilot study), instead of generating an orthogonal design one could consider generating an efficient design assuming zero priors whereby all (qualitative and quantitative) attributes are dummy or effects coded. Such a design strategy has the benefit that it allows more flexibility in removing strictly dominant alternatives as well as imposing attribute level constraints.

In general, efficient designs will not be orthogonal, and this is fine because orthogonality has no particular benefit in model estimation. If for whatever reason one desires the efficient design to be orthogonal, both the `eff` and `orth` properties can be used in conjunction in the script to generate an efficient orthogonal design, see for example the syntax below. Note, however, that in this case all orthogonal design restrictions apply, namely strictly dominant alternatives cannot be avoided and attribute level constraints cannot be imposed.

```
;eff = (mnl,d)
;orth = seq
```

Suppose we would like to generate a D-efficient design in our laptop choice example, see Script 5.2. In this script, we used estimation coding, which is required for efficient designs, and zero priors are specified where only the preference order of the attribute levels is indicated. Table 5.1 shows the generated efficient design with a D-error of 0.002373. In contrast to an orthogonal design, this

[2]These conditions include, but are not limited to, the existence of an orthogonal array for the specified number of levels for each attribute as well as the specified number of design rows.

	Car			No car
Choice task	Engine	Colour	Type	
1	0	1	3	–
2	0	3	0	–
3	2	1	1	–
4	2	2	3	–
5	1	2	2	–
6	1	1	1	–
7	3	0	2	–
8	3	1	2	–
9	1	0	1	–
10	3	2	3	–
11	2	0	0	–
12	0	2	2	–
13	1	3	3	–
14	0	0	1	–
15	3	3	0	–
16	2	3	0	–

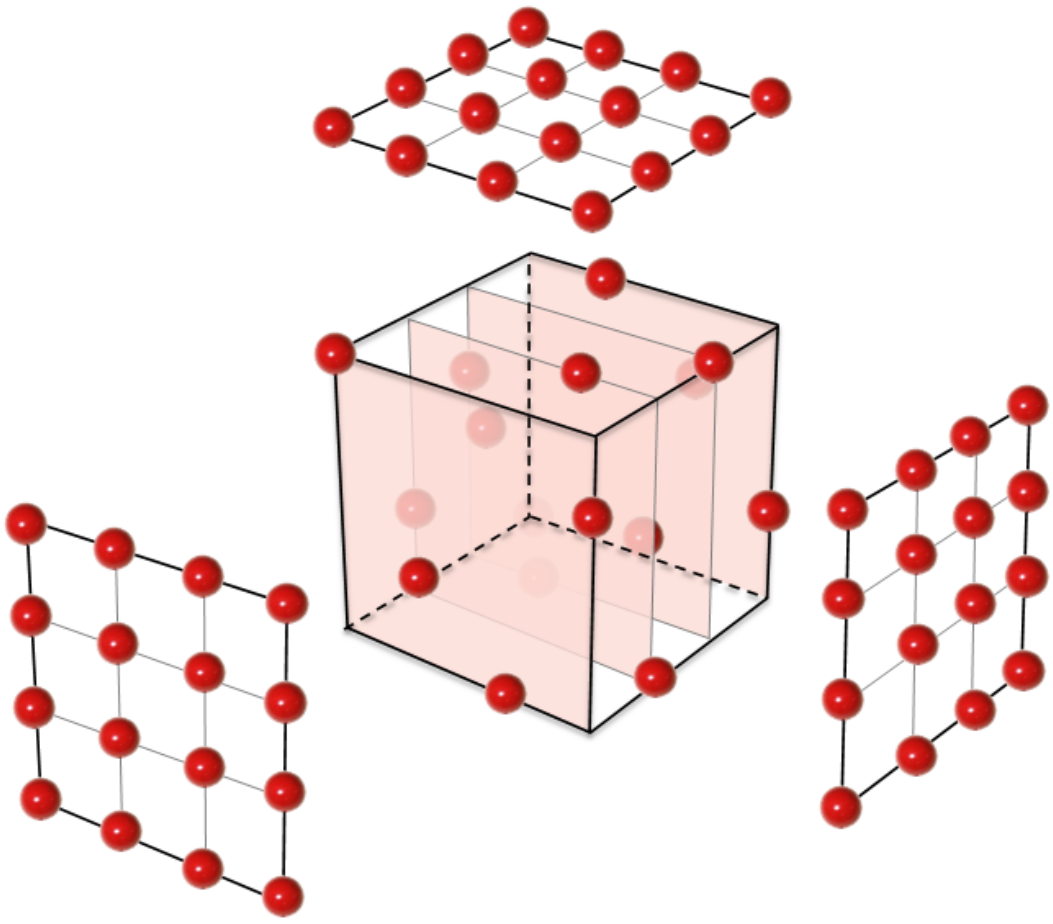


Figure 5.1: Efficient design and graphical depiction

```
design  ? Laptop choice example
;alts = (laptopA, laptopB)
;rows = 12
;block = 2
;eff = (mnl,d)
;model:     ? using estimation coding
U(laptopA, laptopB) = proc.dummy[+] * PROCESSOR[0,1,2,3]
                    + stor[+]       * STORAGE[256,512,1024,2048]
                    + cost[-]       * PRICE[1200,1500,1800,2100]
? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7),  3(Core i9)
? STORAGE:    256 GB,     512 GB,     1024 GB,     2048 GB
? PRICE:      $1200,      $1500,      $1800,       $2100
$
```

Script 5.2: Efficient design for unlabelled experiment

		Laptop A			Laptop B		
Choice task	Block	Processor	Storage	Price	Processor	Storage	Price
1	1	2	512	1200	3	1024	2100
2	1	0	2048	1800	3	256	1500
3	1	1	1024	2100	0	512	1200
4	1	2	256	1500	0	2048	1800
5	1	1	512	1200	2	1024	2100
6	1	3	1024	2100	1	512	1200
7	2	0	1024	1800	2	512	1500
8	2	3	256	1500	2	2048	1800
9	2	3	2048	1200	1	256	2100
10	2	0	256	2100	3	2048	1200
11	2	1	2048	1800	0	256	1500
12	2	2	512	1500	1	1024	1800

Table 5.1: D-efficient design based on noninformative local priors for laptop choice example

efficient design has taken the preference order of the attribute levels into account, such that there are no strictly dominant alternatives across any of the twelve choice tasks.

The design in Table 5.1 is attribute level balanced. In addition, each block also exhibits a high degree of attribute level balance, albeit not perfect because only orthogonal designs can achieve that. While each design is generated, Ngene runs a blocking algorithm to allocate choice tasks to blocks such that a reasonable degree of attribute level balance within each block is achieved. However, if a higher degree of attribute level balance within is block is desired, it is recommended to generate a (new) blocking *after* the design has been generated, as explained in Section 5.8.

Although the design in Table 5.1 is fine to use in a choice experiment, it is important to be aware that efficient designs for unlabelled experiments often contain specific (optimal) comparisons of levels for quantitative attributes in each choice task instead of a broader range of comparisons seen in orthogonal designs. This can also be observed for quantitative attributes in a labelled experiment that have a generic coefficient across multiple alternatives.

For example, each choice task in Table 5.1 makes a comparison between Laptop A and B, whereby the two outer levels are considered for `STORAGE` (256 GB versus 2048 GB) and `PRICE` ($1200 versus

		Laptop A			Laptop B		
Choice task	Block	Processor	Storage	Price	Processor	Storage	Price
1	1	2	2048	2100	0	1024	1800
2	1	3	512	1500	0	2048	2100
3	1	1	512	1500	2	1024	1200
4	1	2	2048	1800	3	256	1200
5	1	1	512	1200	2	256	1500
6	1	0	256	1800	1	1024	1500
7	2	2	256	1200	3	2048	2100
8	2	1	1024	1800	0	512	1500
9	2	0	1024	1200	1	256	1800
10	2	3	1024	2100	1	2048	1200
11	2	3	256	2100	2	512	1800
12	2	0	2048	1500	3	512	2100

Table 5.2: D-efficient design for laptop choice example when all attributes are dummy coded

$2100), or the two inner levels (512 GB versus 1024 GB, $1500 versus $1800). This happens especially with noninformative priors because larger trade-offs (using comparisons of extreme levels) capture much more (Fisher) information – and hence result in smaller standard errors during model estimation – than small trade-offs when estimating a single coefficient (representing a linear effect) for a quantitative attribute. For qualitative attributes that are dummy or effects coded, such as `PROCESSOR`, such a pattern is generally not observed. Therefore, if the limited set of attribute level comparisons in Table 5.1 is deemed undesirable, one can simply apply dummy or effects coding to the relevant quantitative attributes, see, for example, the syntax below. The resulting design is shown in Table 5.2 and has a larger variety of attribute level comparisons for storage and price attributes. Of course, this design still allows estimating a single parameter to represent a linear effect; it does not need to be dummy coded during the model estimation phase.

```
U(laptopA, laptopB) = proc.dummy[+] * PROCESSOR[0,1,2,3]
                    + stor.dummy[+] * STORAGE[256,512,1024,2048]
                    + cost.dummy[-] * PRICE[1200,1500,1800,2100]
```

5.3 Specifying informative local priors

While specifying noninformative (zero) priors is a safe strategy, more information can be captured in the data collection when informative priors are specified. However, informative priors must be carefully chosen since poorly chosen priors can lead to an *in*efficient experimental design. For example, if the true value of a parameter is $\beta_k = 0.3$ but a prior of 5 was used when generating the efficient design, then this will have a detrimental effect on the resulting efficiency in the data. Informative priors should preferably be chosen based on parameter estimates obtained in a pilot study, while experienced users may be able to adapt priors from the literature or use expert judgement. The closer the priors are to the true parameter values, the more efficient the design will be. However, since priors chosen during the design phase are only best guesses, and the parameter estimates after data collection will inescapably deviate somewhat from the assumed priors.

Suppose that we would like to generate a D-efficient design with informative priors for our laptop choice experiment. Then we need to modify the specification of the utility function in Script 5.2.

		Laptop A			Laptop B		
Choice task	Block	Processor	Storage	Price	Processor	Storage	Price
1	1	2	1024	1800	1	512	1200
2	1	2	512	1500	0	2048	1800
3	1	3	512	1500	2	1024	2100
4	1	0	2048	2100	3	256	120
5	1	2	1024	1800	0	256	1500
6	1	0	256	1200	03	1024	2100
7	2	3	2048	2100	1	512	1500
8	2	3	512	1800	2	1024	1500
9	2	1	256	1200	0	2048	1800
10	2	0	256	1200	1	2048	2100
11	2	1	1024	1500	3	512	1800
12	2	1	2048	2100	2	256	1200

Table 5.3: D-efficient design based on informative local priors for laptop choice example

Informative priors can be specified by adding values between square brackets to each parameter as shown in the example syntax below. Since the parameters in alternative `laptopB` are the same, the prior values do not have to be repeated there.

```
U(laptopA, laptopB) = proc.dummy[-0.7|-0.5|-0.1] * PROCESSOR[0,1,2,3]
                    + stor[0.0015]               * STORAGE[256,512,1024,2048]
                    + cost[-0.003]               * PRICE[1200,1500,1800,2100]
```

In this syntax, the cost coefficient `cost` has a prior value of -0.003, so that the expected utility contribution of a price of \$2100 is $-0.003 \times 2100 = -6.3$, while compared to \$2100, a price level of \$1800 yields $-0.003 \times (1800 - 2100) = 0.9$ more utility. Similarly, a storage capacity of 2048 GB yields $0.0015 \times (2048 - 256) = 2.688$ more utility than a storage capacity of 256 GB. Prior values for dummy or effects coded coefficients need to be separated using a pipe (`|`) symbol. Since the qualitative attribute `PROCESSOR` has four levels, whereby the last level (`3`) is the base level, it is necessary to specify three priors for the three parameters associated with `proc`. The first reflects the prior associated with level 0 (Core i3), the second reflects the prior associated with level 1 (Core i5), and so on, all relative to base level 3 (Core i9). Processor level 0 (Core i3) yields 0.7 less utility than processor level 3 (Core i9), while level 1 (Core i5) yields $-0.5 - (-0.7) = 0.2$ more utility than a level 0 (Core i3) processor, etc.

Table 5.3 shows the generated design with a D-error of 0.004027, and Table 5.4 provides the corresponding choice probabilities for each choice task based on the provided informative local priors. Note that comparing this D-error to the D-error of 0.002373 for the design in Table 5.1 is meaningless because the assumed priors are different.

Changing the order of the attribute levels does not change the behavioural model, but for dummy or effects-coded attributes, it is important that the priors are listed in an order that is consistent with the order in which the attribute levels appear. For example, if for attribute `PROCESSOR` we would like the base level to be 0 (Core i3) instead of 3 (Core i9), then level 0 should be listed as the *last* level for `PROCESSOR` (see Section 3.7). This is shown in the syntax below, which also requires changing the order of the priors, namely processor level 1 (Core i5) has again 0.2 more utility than base level 0 (Core i3) and level 3 has again 0.7 more utility than the base level (which has zero utility when

Choice task	Block	Laptop A	Laptop B
1	1	0.347057	0.652943
2	1	0.309171	0.690829
3	1	0.756208	0.243792
4	1	0.329157	0.670843
5	1	0.700987	0.299013
6	1	0.700147	0.299853
7	2	0.731844	0.268156
8	2	0.172502	0.827498
9	2	0.334478	0.665522
10	2	0.453138	0.546862
11	2	0.762783	0.237217
12	2	0.398433	0.601567

Table 5.4: Choice probabilities in D-efficient design for laptop choice example

dummy coding is assumed). Note that the order in which the levels of a quantitative attribute without dummy or effects coding appear is irrelevant; as shown below, changing the level order for `STORAGE` and `PRICE` does not impact the prior specification.

```
U(laptopA, laptopB) = proc.dummy[0.2|0.6|0.7] * PROCESSOR[1,2,3,0]
                    + stor[0.0015]            * STORAGE[512,256,2048,1024]
                    + cost[-0.003]            * PRICE[2100,1800,1500,1200]
```

It is very important to use appropriate prior values, preferably from a pilot study. A common mistake is that a prior is manually set to a value that is relatively too large. For example, a prior value of -0.1 for `cost` would be too large since this parameter is multiplied by large values for price. This would yield unrealistically large utility differences between price levels such that the price attribute would dominate choices in the model, resulting in choice probabilities near 0 and 1 and a relatively high D-error (larger than 1) that indicates that the resulting design is very *in*efficient.

Script 5.3 generates a D-efficient design for a labelled mode choice experiment. In the same way, parameter priors are used. In this case, the label-specific constants `con_car` and `con_bus` also require prior values (which are typically obtained from a pilot study). The resulting D-efficient design is shown in Table 5.5, and the associated choice probabilities are presented in Table 5.6.

Script 5.3 also introduces a new property on line 6, namely:

```
;con
```

When `con` is added as a property to the script, Ngene will also optimise the design for estimating all constants. Without this property, the rows and columns representing the constants are omitted from the covariance matrix before computing the D-error[3], in which case the design generation process does not aim to reduce the standard errors of the constants. Adding `con` is a good idea when the objective of the study is to forecast demand or market shares, but is not needed when the main interest is in determining marginal rates of substitution such as willingness-to-pay.

[3]But prior values of the constants are always considered when calculating choice probabilities, Fisher information, and the covariance matrix.

```
design ? mode choice example
;alts = car, bus, train
;rows = 18
;block = 3
;eff = (mnl,d)
;con
;model:
U(car)   = con_car[0.3]                          ? Constant for car
         + ctime[-0.05]      * CTIME[10,15,20,25] ? car driving time (min)
         + fuel[-0.5]        * FUEL[1,2]          ? fuel cost ($)
         + toll[-0.6]        * TOLL[1,2,3]        ? toll cost ($)
         /
U(bus)   = con_bus[-0.2]                         ? Constant for bus
         + btime[-0.07]      * BTIME[30,35,40,45] ? bus in-vehicle time (min)
         + trans.dummy[-0.4] * TRANSFER[1,0]      ? transfer: 0 = no (base), 1 = yes
         + wait[-0.12]       * WAIT[1,5,10]       ? waiting time (min)
         + bseat.dummy[0.3]  * SEATING[1,0]       ? seat available: 0 = no (base), 1 = yes
         + cost[-0.5]        * BFARE[1,2,3]       ? bus fare ($)
         /
U(train) = ttime[-0.06]      * TTIME[5,10,15,20]  ? train in-vehicle time (min)
         + trans             * TRANSFER
         + wait              * WAIT
         + tseat.dummy[0.2]  * SEATING
         + cost              * TFARE[2,3,4]       ? train fare ($)
$
```

Script 5.3: Efficient design for labelled experiment

		Car			Bus					Train				
Task	Block	time	fuel	toll	time	tran	wait	seat	fare	time	tran	wait	seat	fare
1	1	10	2	3	30	0	1	1	3	20	1	5	1	2
2	1	20	2	3	45	0	1	0	1	15	0	10	1	4
3	1	15	1	1	40	1	10	0	3	15	1	1	1	2
4	1	25	2	2	30	1	1	1	1	10	0	10	1	4
5	1	10	1	1	40	1	10	0	3	5	0	5	0	2
6	1	25	2	2	30	0	10	1	1	5	1	1	0	4
7	2	20	1	3	45	0	5	1	1	15	1	1	0	4
8	2	25	1	3	35	1	5	0	2	20	0	10	0	2
9	2	10	2	2	40	1	5	1	3	5	0	10	0	2
10	2	20	1	1	35	1	5	0	2	20	0	1	0	3
11	2	15	1	2	45	0	5	1	2	15	1	5	0	3
12	2	25	1	1	40	1	10	0	2	5	0	5	1	3
13	3	15	2	2	30	0	1	0	1	10	1	5	0	4
14	3	20	2	1	35	1	10	0	2	20	0	1	1	3
15	3	25	2	2	45	0	1	1	1	15	1	10	0	3
16	3	10	1	3	35	1	10	1	2	10	0	1	1	4
17	3	20	1	3	35	0	1	0	3	5	1	10	1	2
18	3	15	2	1	40	0	5	1	3	10	1	5	1	3

Table 5.5: D-efficient design based on informative local priors for mode choice example

Choice task	Block	Car	Bus	Train
1	1	0.394020	0.211961	0.394020
2	1	0.435669	0.272294	0.292037
3	1	0.656911	0.006942	0.336147
4	1	0.360171	0.410173	0.229655
5	1	0.642244	0.005286	0.352471
6	1	0.336937	0.194395	0.468668
7	2	0.506667	0.160429	0.332904
8	2	0.474540	0.117020	0.408440
9	2	0.508738	0.030936	0.460325
10	2	0.705000	0.040780	0.254220
11	2	0.730679	0.059978	0.209343
12	2	0.529259	0.015203	0.455538
13	3	0.465090	0.355040	0.179869
14	3	0.562275	0.029429	0.408296
15	3	0.494576	0.294035	0.211389
16	3	0.483912	0.041758	0.474330
17	3	0.380128	0.106752	0.513120
18	3	0.670537	0.042866	0.286597

Table 5.6: Choice probabilities in D-efficient design for mode choice example

5.4 Algorithms to generate efficient designs

Several algorithms are available in Ngene to determine an efficient design. The preferred algorithm can be specified using the `alg` property:

- `alg = swap` – column-based swapping algorithm (default)
- `alg = mfedorov` – row-based modified Federov algorithm
- `alg = all` – evaluates all possible designs

More information on column- and row-based algorithms can be found in Section 1.5.1. Each algorithm has several default settings that could be changed in the script, but it is generally not necessary to deviate from the default settings. For more information on algorithm settings, please refer to the *Syntax help* in the script editor.

For choice experiments where the number of possible experimental designs is small enough, one could evaluate all possible designs by specifying `alg = all`. For example, if the full factorial has only 16 rows (e.g., two alternatives, each with two attributes that have two levels) and one would like to select the most efficient design with 4 rows, then there exist $16 \cdot 15 \cdot 14 \cdot 13 = 65,536$ possible designs. However, it is clear that in essentially all practical applications, the total number of possible designs is far too large to make this algorithm a viable choice.

The other two algorithms, `swap` and `mfedorov`, solve a complex optimisation problem with the criterion specified in the `eff` property serving as the objective function, which is computationally intensive. If no algorithm is specified in the script to generate an efficient design, then by default `alg = swap` is assumed, which randomises attribute levels in each column and randomly swaps levels within each column. The main benefit of the swapping algorithm is that it maintains attribute level balance when changing levels within an attribute column (see Section 1.5 and Figure 1.5(a)). For this reason, this is often the best algorithm for generating an efficient design.

Sometimes, the default swapping algorithm will struggle to find a design. This may happen when there is a high occurrence of strictly dominant alternatives (often when only a small number of attributes and attribute levels are specified), or when a large number of constraints are imposed (see Chapter 6. For example, consider Script 5.2 and suppose that the number of rows is doubled from 12 to 24. Running this script will not produce a design because the default swapping algorithm will not be able to generate 24 choice tasks without any strictly dominant alternatives. A message similar to the one below would be shown in the log screen:

> "*A valid initial random design could not be generated after approximately 10 seconds. In this time, of the 543999 attempts made, there were 0 row repetitions, 17900 alternative repetitions, and 526099 cases of dominance. There are a number of possible causes for this, including the specification of too many constraints, not having enough attributes or attribute levels for the number of rows required, and the use of too many scenario attributes. A design may yet be found, and the search will continue for 10 minutes. Alternatively, you can stop the run and alter the syntax.*"

If the default swapping algorithm cannot locate a design, then one can switch to the modified Fedorov algorithm, which can be invoked by adding the following syntax to the script:

```
;alg = mfedorov
```

The modified Federov algorithm is a row-based algorithm that first creates a candidate set and then composes a design by selecting the choice tasks from this candidate set. For each row in the design, the algorithm iteratively replaces the row with each choice task in the candidate set (except for those that are already in the design) and evaluates the design efficiency. The default candidate set consists of 2,000 randomly generated choice tasks, which is typically sufficient for most studies. A larger candidate set size will make the algorithm slower while often only marginally improving the efficiency of the generated design; therefore, a candidate set with more than 5,000 choice tasks is generally not recommended. If one wishes to change the candidate set size, for example, to 3,000 choice tasks, then the following syntax can be used. The maximum candidate set size in Ngene is 10,000 choice tasks. If Ngene is not able to generate a candidate set size of the desired size, it will (in most cases) automatically reduce the candidate set size and report this in the log screen.

```
;alg = mfedorov(candidates = 3000)
```

Script 5.4 generates a design for the laptop choice experiment using the modified Fedorov algorithm. Table 5.7 shows the resulting design after about 25,000 design evaluations. This design has a D-error of 0.003722, which is lower than the D-error of 0.004027 for the design in Table 5.3. However, this increase in efficiency comes at the cost of reduced attribute level balance.

Observe that for qualitative attribute `PROCESSOR` its four levels each appear six times across both alternatives, although there are some differences within each alternative (which is usually fine). Attributes that are dummy or effects coded generally show a high degree of attribute level balance because the design will not be efficient if one or more levels are rarely present because this would mean that the parameter associated with that level cannot be estimated reliably. However, for quantitative attributes `STORAGE` and `PRICE` mostly the two extreme levels appear (256 GB and 2048 GB, $1200 and $2100), while the inner levels are under-represented (512 GB and 1024 GB, $1500 and $1800). As also mentioned in Section 5.2, large trade-offs between extreme levels provide more (Fisher) information than small trade-offs with inner levels when estimating a linear effect. Hence,

```
design  ? Laptop choice example
;alts = (laptopA, laptopB)
;rows = 12
;block = 2
;eff = (mnl,d)
;alg = mfedorov
;model:
U(laptopA, laptopB) = proc.dummy[-0.7|-0.5|-0.1] * PROCESSOR[0,1,2,3]
                    + stor[0.0015]               * STORAGE[256,512,1024,2048]
                    + cost[-0.003]               * PRICE[1200,1500,1800,2100]
? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7),  3(Core i9)
? STORAGE:    256 GB,     512 GB,     1024 GB,     2048 GB
? PRICE:      $1200,      $1500,      $1800,       $2100
$
```

Script 5.4: Using modified Fedorov algorithm

		Laptop A			Laptop B		
Choice task	Block	Processor	Storage	Price	Processor	Storage	Price
1	1	0	256	1200	2	2048	2100
2	1	1	2048	2100	0	256	1200
3	1	1	2048	2100	3	512	1200
4	1	3	2048	2100	0	256	1200
5	1	3	256	1200	0	2048	2100
6	1	3	256	1200	2	2048	2100
7	2	2	2048	2100	1	256	1200
8	2	2	1024	1200	3	2048	2100
9	2	1	2048	2100	2	256	1800
10	2	1	256	1500	0	2048	1800
11	2	3	1024	2100	1	256	1200
12	2	2	256	1200	0	2048	1800

Table 5.7: D-efficient design using the modified Fedorov algorithm for laptop choice example

when using the modified Fedorov algorithm, the quantitative attributes will often exhibit a low degree of attribute level balance.

Three options exist to achieve a better degree of attribute level balance when using the modified Federov algorithm. First, one can use dummy or effects coding for quantitative attributes; see also Section 5.2. While this is often fine when using noninformative (zero priors), with informative priors one would preferably stay closer to the utility function specification that will be used in the model estimation phase. Secondly, one can add `(imbalance)` as an optimisation criterion in the `eff` property, as discussed in Section 5.1.2, although this will not guarantee that all attributes exhibit an acceptable degree of level balance.

A third option is to specify the frequency with which each attribute level must occur across all choice tasks by defining these frequencies in the utility functions in the `model` property. For example, in the utility function specification in Script 5.4 we can add attribute level frequency constraints for the `STORAGE` attribute. In the following syntax, we added `(3,3,3,3)` directly after the levels for this attribute, which indicates that each of the 4 levels should appear exactly 3 times in the 12 rows (to create perfect attribute level balance).

```
U(laptopA, laptopB)
   = proc.dummy[-0.7|-0.5|-0.1] * PROCESSOR[0,1,2,3]
   + stor[0.0015]               * STORAGE[256,512,1024,2048](3,3,3,3)
   + cost[-0.003]               * PRICE[1200,1500,1800,2100]
```

One could also add `(3,3,3,3)` directly after the `PRICE` levels in the utility function. A row-based algorithm, like the modified Fedorov algorithm, may not be able to find designs if too many strict attribute level frequency constraints are imposed. In addition, any imposed attribute level frequency constraints will make the generated design less efficient. Therefore, it is often best to somewhat relax these constraints by specifying lower and upper bounds on how often each attribute level should appear. For example, in the syntax below we added `(2-4,2-4,2-4,2-4)` to require that each storage capacity level appears no less than twice across the 12 rows and no more than 4 times. For the price attribute, similar attribute level frequency constraints are added, whereby in this example we require the outer levels to appear exactly twice and each middle level in needs to appear at least 3 times across the 12 rows. Table 5.8 presents an efficient design that satisfies these attribute level frequency constraints.

```
U(laptopA, laptopB)
   = proc.dummy[-0.7|-0.5|-0.1] * PROCESSOR[0,1,2,3]
   + stor[0.0015]               * STORAGE[256,512,1024,2048](2-4,2-4,2-4,2-4)
   + cost[-0.003]               * PRICE[1200,1500,1800,2100](2,3-12,3-12,2)
```

The swapping and modified Fedorov algorithms use a randomisation process; when one runs the same script again, a different design (with similar efficiency) will be found. These algorithms will automatically terminate when no better design has been found after 5,000[4] consecutive design evaluations, but will ultimately and time out after 10 hours. To avoid unnecessary long run times, it is advised to stop the design search manually if the search graph indicates that new designs only marginally improve efficiency; see also Section 2.3. These termination criteria can be modified within the script. In the first example syntax below, the swapping algorithm runs for a total of 50,000 iterations (design evaluations), while in the second example, the modified Fedorov algorithm terminates after 10 minutes, or after no better design was found during 10,000 consecutive iterations (design evaluations), whichever comes first.

[4]To avoid premature termination in the modified Fedorov algorithm, this termination criterion is adjusted to twice the candidate set if the candidate set size is larger than 2,500 rows.

		Laptop A			Laptop B		
Choice task	Block	Processor	Storage	Price	Processor	Storage	Price
1	1	1	512	1500	0	2048	1800
2	1	1	256	1500	2	1024	1800
3	1	2	256	1200	1	1024	1800
4	1	0	2048	1800	2	256	1500
5	1	3	2048	2100	2	512	1500
6	1	0	1024	1800	3	256	1200
7	2	3	256	1500	2	2048	2100
8	2	2	1024	2100	0	512	1500
9	2	1	2048	1800	3	256	1500
10	2	0	512	1800	3	256	1500
11	2	1	2048	1800	0	512	1200
12	2	1	1024	1200	3	2048	2100

Table 5.8: D-efficient design using modified Fedorov algorithm with balanced attributes

```
;alg = swap(stop=total(50000 iterations))
```

```
;alg = mfedorov(stop=total(10 mins), stop=noimprov(10000))
```

Both the swapping and the modified Fedorov algorithm are computationally intensive, and sometimes it may be useful to give it an initial design for a 'hot start' of the algorithm. For example, suppose that one previously generated a design in Ngene or imported a design (see Section 2.5) named 'My Design' in the same experiment folder. Then it is possible to use this design as a starting point in the algorithm by adding the syntax below to the script.

```
;start = My Design
```

5.5 Specifying Bayesian priors

A Bayesian prior accounts for the fact that there is uncertainty about the true parameter value. Therefore, a Bayesian prior is defined by a random distribution; see Section 1.5.1. To explain how to use Bayesian priors in Ngene, consider Script 5.5. This script generates a Bayesian efficient design for the laptop choice experiment. In this script, we consider the D-error as efficiency criterion and the multinomial logit as model type of interest, but different efficiency criteria or model types could of course be specified.

In lines 8–10 of Script 5.5 Bayesian priors have been specified for the parameters. Ngene accepts two types of Bayesian priors, specified in the following format (with parentheses):

- `(n,estim,se)` – Normally distributed prior based on estimate `estim` and standard error `se`
- `(u,lower,upper)` – Uniformly distributed prior based on estimated range [`lower`,`upper`]

A normal distribution is often useful when parameter estimates from a pilot study are available. Even if parameter estimates are not statistically significant, they often still provide the best guess available, and standard errors can be used to indicate the level of accuracy (precision) of these estimates. For parameter `stor` in Script 5.5 a normally distributed Bayesian prior based on a parameter estimate of 0.0015 and a standard error of 0.0005 is assumed in line 9, while in line 10 a normal distribution with

```
1  design  ? Laptop choice example
2  ;alts = (laptopA, laptopB)
3  ;rows = 12
4  ;block = 2
5  ;eff = (mnl,d,mean)
6  ;bdraws = sobol(300)
7  ;model:
8  U(laptopA, laptopB)
9     = proc.dummy[(u,-1,-0.6)|(u,-0.6,-0.4)|(u,-0.4,0)] * PROCESSOR[0,1,2,3]
10    + stor[(n,0.0015,0.0005)]                          * STORAGE[256,512,1024,2048]
11    + cost[(n,-0.003,0.001)]                           * PRICE[1200,1500,1800,2100]
12 ? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7),  3(Core i9)
13 ? STORAGE:    256 GB,     512 GB,     1024 GB,     2048 GB
14 ? PRICE:      $1200,      $1500,      $1800,       $2100
15 $
```

Script 5.5: Assuming Bayesian priors

estimate −0.003 and standard error 0.001 is assumed for the prior of the `cost` parameter, indicating that the most likely prior values are 0.0015 and −0.003, but there is uncertainty about these values.

A uniform distribution is useful when one would like to specify a range of prior values. By using a lower (upper) bound of 0 one can indicate that the parameter is expected to have a positive (negative) value. In Script 5.5 uniformly distributed Bayesian priors are specified for the `proc` parameters, e.g., the prior for the dummy parameter associated with level 0 (Core i3) is believed to have a value between −1 and −0.6, while the prior for level 2 (Core i7) is assumed to be between −0.4 and 0. The range of the uniform distribution indicates the level of uncertainty about the prior value.

In line 5 of Script 5.5 we indicated in the `eff` property that we would like to generate a Bayesian efficient design by minimising the average (`mean`) D-error. It is important to specify `mean` or `median` as the third argument in the `eff` property if one wants to generate a Bayesian efficient design. If only `eff` or `eff = (mnl,d)` is specified, then this would default to `eff = (mnl,d,fixed)`, which would generate a locally efficient design, despite the fact that Bayesian priors are specified in the script.

In line 6 we added the property `bdraws` to specify which type of draw and how many draws we would like to take from each of these distributions for the Bayesian priors. In the script, we chose 300 draws using Sobol sequences. The following types of draws are available in Ngene for Bayesian priors, showing default values between parentheses:

- `bdraws = random(200)` – 200 pseudo-random draws
- `bdraws = halton(200)` – 200 quasi-random draws using Halton sequences (default)
- `bdraws = sobol(200)` – 200 quasi-random draws using Sobol sequences
- `bdraws = mlhs(200)` – 200 quasi-random draws using modified latin hybercube sampling
- `bdraws = gauss(3)` – Gaussian quadrature using 3 abscissas for each Bayesian prior

If `bdraws` is not specified in the script, `bdraws = halton(200)` is assumed by default. Quasi-random draws (`halton`, `sobol`, and `mlhs`) are 'smarter' draws than pseudo-random draws (`random`) and result in the same accurate calculations of Bayesian efficiency with a smaller number of draws. For this reason, quasi-random draws are generally preferred over pseudo-random draws. Although Halton draws are typically good when no more than five Bayesian priors are specified, its accuracy deteriorates for larger numbers of Bayesian priors, and either Sobol sequences or MLHS would be recommended in such cases. Another type of draw with superior accuracy is Gaussian quadrature (`gauss`). Gaussian quadrature differs from pseudo/quasi-random draws, namely it first determines

abscissas for each prior distribution, which are specific points on the distribution, and assigns them a weight. Then it creates draws by creating all possible combinations (i.e., the full factorial) of abscissas across all prior distributions. The average Bayesian efficiency is then calculated as a weighted average across all draws. Note that `median` cannot be used in combination with Gaussian quadrature due to the nature of this type of draw.

If the number of draws is not specified, Ngene defaults to 200 pseudo- or quasi-random draws or 3 abscissas for Gaussian quadrature. The required number of draws to compute Bayesian efficiency with sufficient accuracy increases with the number of Bayesian priors. More draws means more accurate Bayesian efficiency calculations but increased computation time.[5] While a few hundred draws may suffice with five Bayesian priors, thousands of draws may be required with ten or more Bayesian priors. The maximum number of draws that can be specified is 10,000 draws. To avoid long run times or inaccurate Bayesian efficiency calculations (which may generate designs that are actually not that efficient), it is usually recommended to use no more than ten Bayesian priors and use local priors for parameters attached to attributes of lesser relative importance (as measured by their contribution to utility).

One should be careful when using Gaussian quadrature since the total number of draws is only indirectly specified via the number of abscissas. If we used `bdraws = gauss(3)` in Script 5.5, which has five Bayesian priors, then the total number of draws would be $3 \cdot 3 \cdot 3 \cdot 3 \cdot 3 = 3^5 = 243$ draws. The number of draws exponentially increases with the number of abscissas, so, for example, using `bdraws = gauss(5)` in Script 5.5 would increase the number of draws to $5^5 = 3,125$. If one specifies Bayesian priors for each of the 12 parameters (including constants) in Script 5.3, then `bdraws = gauss(3)` would already amount to a total of $3^{12} = 531,441$ draws, which would far exceed the maximum of 10,000 draws. It is possible to specify a different number of abscissas for each Bayesian prior.[6] For example, `bdraws = gauss(2,2,2,4,4)` in Script 5.5 would result in $2 \cdot 2 \cdot 2 \cdot 4 \cdot 4 = 2^3 4^2 = 128$ draws, whereby two abscissas are used for the first three Bayesian priors listed in the script, namely those associated with parameter `proc`, and four abscissas are used for the prior distributions of `stor` and `cost`.

In the Result tab of the project screen one can view various design efficiency measures, such as average D-error across the draws, minimum and maximum D-error across the draws, and even the D-error for each of the draws. It also reports the fixed D-error by assuming the midpoint of each prior distribution as a local prior (e.g., in Script 5.5 the mid-point values are `proc.dummy[-0.8|-0.5|-0.2]`, `stor[0.0015]` and `cost[-0.003]`).

Note that the draw types `random` and `mlhs` rely on randomisation and will each time produce different draws. To fix the draws, one can specify a random seed for Bayesian draws using the property `bseed`. For example, `bseed = 12345` uses the number 12345 as a seed to generate random numbers. Adding this property to the script uses the same Bayesian draws each time the script is run, which is especially useful to obtain the same result in design evaluation; see Section 5.8.

5.6 Specifying functions of attributes

In some cases, one may want to use functions of one or more attributes in the utility function specifications. Ngene supports such functions by specifying the levels of an attribute using `fcn(...)`, but only in combination with the default swapping algorithm. Currently only simple linear functions

[5] For instance, calculating design efficiency assuming Bayesian priors with 1,000 draws takes 1,000 times longer than calculating design efficiency assuming local priors.

[6] For instance, using more abscissas for wide prior distributions or ones associated with more important attributes and less abscissas for other prior distributions.

```
design  ? Route choice example
;alts = (optA, optB)
;rows = 12
;eff = (mnl,d)
;model:
U(optA, optB)
   = b1[0.5]  * prEarly[0.2,0.4]                              * Early[10,12,14]
   + b2[0.2]  * prOntime[0.5,0.3]                             * Ontime[20,22,24]
   + b3[-0.4] * prLate[fcn(1 - optA.prEarly - optA.prOntime)] * Late[25,27,29]
? prEarly:  0.2(20%),        0.4(40%)
? prOntime: 0.5(50%),        0.3(30%)
? prLate:   1-0.4-0.5(10%),  1-0.2-0.5(30%),  1-0.4-0.3(30%),  1-0.2-0.3(50%)
? Early:    10 min,     12 min,    14 min
? Ontime:   20 min,     22 min,    24 min
? Late:     25 min,     27 min,    29 min
$
```

Script 5.6: Design with functions of attributes

are supported in Ngene, therefore only plus (`+`), minus (`-`), attribute names and constants are allowed.

Script 5.6 shows an example in which the level of the attribute `prLate` is a function of the levels of attributes `prEarly` and `prOntime`. Note that attributes need to be referenced by their combined alternative and attribute name, e.g. `optA.prEarly`. This function ensures that the sum of attributes `prEarly`, `prOntime` and `prLate`, which represent probabilities, is always equal to 1. Because `prLate` is a linear combination of `prEarly` on `prOntime`, it can only appear as an interaction effect as otherwise there would exist multicollinearity.

5.7 Designs for mixed logit models

Ngene can generate efficient designs for the model types mentioned in Section 5.1.1. Except for the multinomial logit (`mnl`) model, all other model types are variants of the mixed logit model whereby random parameters and/or random error components can be considered.

Ngene can optimise for mixed logit models where distributions of random parameters and/or error components are specified as follows (without parentheses):

- `n,mu,sigma` – Normally distributed parameter with mean `mu` and standard deviation `sigma`
- `u,min,max` – Uniformly distributed parameter with minimum `min` and maximum `max`

Similarly to `bdraws` defined in Section 5.5, `rdraws` is used to specify the type and number of draws for random parameters or error components, and has the following options (default values shown between parentheses):

- `rdraws = random(200)` – 200 pseudo-random draws
- `rdraws = halton(200)` – 200 quasi-random draws using Halton sequences (default)
- `rdraws = sobol(200)` – 200 quasi-random draws using Sobol sequences
- `rdraws = mlhs(200)` – 200 quasi-random draws using modified latin hybercube sampling
- `rdraws = gauss(3)` – Gaussian quadrature using 3 abscissas for each Bayesian prior

Model types `rppanel`, `ecpanel`, `rpecpanel` account for the panel nature of the data when each agent receives multiple choice tasks, while model types `rp`, `ec` and `rpec` are only appropriate when each agent is given only a single choice task from the experimental design. For panel mixed logit models –

```
1  design  ? Laptop choice example
2  ;alts = (laptopA, laptopB)
3  ;rows = 12
4  ;block = 2
5  ;eff = (rppanel,d)
6  ;rdraws = gauss(3)
7  ;rep = 500
8  ;model:
9  U(laptopA, laptopB)
10    = proc.dummy[n,-0.7,0.4|n,-0.5,0.3|n,-0.1,0.2] * PROCESSOR[0,1,2,3]
11    + stor[n,0.0015,0.0007]                        * STORAGE[256,512,1024,2048]
12    + cost[n,-0.003,0.0012]                        * PRICE[1200,1500,1800,2100]
13 ? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7),  3(Core i9)
14 ? STORAGE:    256 GB,     512 GB,     1024 GB,     2048 GB
15 ? PRICE:      $1200,      $1500,      $1800,       $2100
16 $
```

Script 5.7: Design for panel random parameter logit model

`rppanel`, `ecpanel`, and `rpecpanel` – the covariance matrix can only be approximated via a simulated sample of agents. Such a sample is simulated in Ngene by repeating the entire design with different draws from the specified random parameter distributions.[7] The default size of the simulated sample (i.e., the number of design repetitions) is 200, but this can be specified via the `rep` property. The larger this simulated sample, the more accurate the covariance matrix and efficiency calculation will be, but the more computation time is required.

```
;rep = 500
```

The seed for draw types `random` and `mlhs` can be fixed by using property `rseed` in the same way as `bseed`, see Section 5.5.

5.7.1 Random parameter models

Script 5.7 generates a D-efficient design for the laptop choice experiment that is optimised to estimate a panel mixed logit model with random parameters, as indicated by `eff = (rppanel,d)` on line 5. The utility function specification on lines 9–11 has been adapted to indicate that all parameters are now randomly distributed. In addition, properties `rdraws` and `rep` were specified in lines 6 and 7, respectively.

Random parameters expresses preference heterogeneity of agents, which should not be confused with Bayesian priors that express uncertainty of the analyst about the true parameter values. In a mixed logit model, distributional parameters `mu`, `sigma`, `min` and `max` are estimated. In Script 5.7, there are five random parameters, and for each parameter a mean and a standard deviation must be estimated. Therefore, the covariance matrix associated with this model has ten rows and columns representing these distributional parameters, whereby the standard deviation parameters on the Results screen in Ngene are indicated with the extension 'std dev.'.

Each of the distributional parameters can have a local or Bayesian prior when generating an efficient design. For example, the parameter associated with processor level 0 (Core i3) follows a normal distribution described by `n,-0.7,0.4`. This means that the local prior for the mean is −0.7

[7] In Ngene currently any blocking is ignored, i.e., it is implicitly assumed that each simulated agent is given all choice tasks.

and the local prior for the standard deviation is 0.4. Similarly, `stor[n,0.0015,0.0007]` as specified in the utility function indicates a normally distributed random parameter with local priors. It is possible to specify Bayesian priors for each distributional parameter, for example, one could specify `stor[n,(n,0.0015,0.0005),(u,0,0.001)]` whereby the mean has a normally distributed Bayesian prior with an estimate of 0.0015 and a standard error of 0.0005, and the standard deviation has a uniformly distributed Bayesian prior with a lower bound of 0 (since a standard deviation cannot be negative) and an upper bound of 0.001. However, combining random parameters with Bayesian priors is computationally feasible only for models with a very small number of random parameters and Bayesian priors.

In line 6 of Script 5.7 we chose Gaussian quadrature with 3 abscissas, and since there are five random parameters, this means a total of $3^5 = 243$ draws. Together with `rep = 500`, this means that this script will run $243 \times 500 = 121,500$ times slower than a script that generates an efficient design for a multinomial logit model. Given this very large increase in computational complexity, one is often better off optimising for a multinomial logit model rather than for a mixed logit model.

5.7.2 Error component models

Error components are additional error terms in a utility function that expresses differences in error variance between labelled alternatives. Error components can also be nested to indicate similarities in error variance between alternatives. An error component is essentially a normally distributed constant with zero mean, whereby only the standard deviation is estimated, and can be defined in Ngene as follows (without parentheses):

- `ec,sigma` – Error component with standard deviation `sigma`

Script 5.8 generates a D-efficient design for the mode choice experiment to estimate a panel mixed logit model with error components, as indicated by `eff = (ecpanel,d)` on line 5. Two error components are added, namely `ec_road` on lines 14 and 22 of the utility functions of the road-based alternatives (car and bus), and `ec_pt` in both public transport modes (bus and train) on lines 23 and 30. Their standard deviations have local priors of 0.5 and 0.3, respectively.

To take draws from the two error components, Script 5.8 uses Gaussian quadrature with 5 abscissas, see line 6, which means case $5 \times 5 = 25$ draws in total. In conjunction with a simulated sample of 1,000 agents (see `rep = 1000` on line 7), this script will run $25 \times 1000 = 25,000$ times slower than when generating a design for a multinomial logit model.

Random parameters can be combined with error components in the utility function specifications. When doing so, one should specify either `rpecpanel` or `rpec` as model type.

5.8 Evaluating or blocking existing designs

If one is interested in evaluating the efficiency of an existing experimental design rather than generating a new design, the `eval` property can be used. Consider a design named 'My Design' that was imported (see Section 2.5) or that was previously generated under different assumptions of the utility functions, model type, priors, or efficiency criterion.

To evaluate the design, create a new design in the same experiment folder and write a script with the desired utility functions, model type, priors,and efficiency criterion, and add the syntax below to the script. Note that the `alg` property should be removed if it is specified in the script.

```
;eval = My Design
```

```
design ? mode choice example
;alts = car, bus, train
;rows = 18
;block = 3
;eff = (ecpanel,d)
;rdraws = gauss(5)
;rep = 1000
;con
;model:
U(car)   = con_car[0.3]                          ? Constant for car
         + ctime[-0.05]      * CTIME[10,15,20,25] ? car driving time (min)
         + fuel[-0.5]        * FUEL[1,2]          ? fuel cost ($)
         + toll[-0.6]        * TOLL[1,2,3]        ? toll cost ($)
         + ec_road[ec,0.5]
         /
U(bus)   = con_bus[-0.2]                         ? Constant for bus
         + btime[-0.07]      * BTIME[30,35,40,45] ? bus in-vehicle time (min)
         + trans.dummy[-0.4] * TRANSFER[1,0]      ? transfer: 0 = no (base), 1 = yes
         + wait[-0.12]       * WAIT[1,5,10]       ? waiting time (min)
         + bseat.dummy[0.3]  * SEATING[1,0]       ? seat available: 0 = no (base), 1 = yes
         + cost[-0.5]        * BFARE[1,2,3]       ? bus fare ($)
         + ec_road
         + ec_pt[ec,0.3]
         /
U(train) = ttime[-0.06]      * TTIME[5,10,15,20]  ? train in-vehicle time (min)
         + trans             * TRANSFER
         + wait              * WAIT
         + tseat.dummy[0.2]  * SEATING
         + cost              * TFARE[2,3,4]       ? train fare ($)
         + ec_pt
$
```

Script 5.8: Design for panel error component logit model

Evaluation of a design is also useful in the case that one first generates a design under the assumption of a multinomial logit model and then would like to evaluate the efficiency of this design for estimating a mixed logit model (see Section 5.7).

When evaluating Bayesian efficiency of a design, or design efficiency for estimating a mixed logit model, you will need to make sure that you specify Bayesian priors and/or random parameters in the utility functions in the `model` property, together with the number of Bayesian draws, set via `bdraws`, and/or the number of random draws, set via `rdraws`, and the number of design repetitions for the simulated sample, set via `rep`.

Another useful application of the `eval` property is when one would like to generate a (new) blocking column for an existing design. When the `block` property is added to the script to generate a design, the blocking algorithm only runs for a limited number of iterations to allocate choice tasks to blocks such that each block exhibits a reasonable degree of attribute level balance. However, if a higher degree of balance is required within each block, it is recommended to run the blocking algorithm in Ngene for a longer period of time *after* generating the design.

For example, suppose that we would like to block the design named 'My Design' into three blocks. We can then modify the `block` property to something like the following:

```
;eval = My Design
;block = 3, total(1 mins), newblocking
```

In this example, the algorithm will run for one minute to find a (new) blocking column that optimises attribute level balance within each block of choice tasks.

6

Constrained choice tasks

This chapter describes how to impose constraints on certain attribute level combinations for each choice task in an experimental design. Such constraints are also referred to as prohibitions. They are often needed to make choice tasks, or profiles within a choice task, more realistic. Constraints can also be used to create scenario variables (which have the same value across all alternatives), to fix attribute levels in status quo alternatives, to create attribute level overlap, or for other purposes.

The type of constraints that can be imposed depends on the design type and the algorithm used to generate the design. It should be noted that constraints cannot be considered for orthogonal designs due to the strict nature of orthogonality.

6.1 Strategies for applying constraints

Constraints should only be applied if they are needed (e.g., for realism purposes), since imposing any constraints will make the design less efficient and more difficult to generate. Ngene supports two types of constraints, namely *conditional constraints* and *check constraints*. Conditional constraints can only be used when generating an efficient design using the default swapping algorithm, while check constraints can only be used when generating an efficient design using the modified Fedorov algorithm. Check constraints can also be applied to generate a constrained full factorial design using the `fact` property or a constrained random fractional factorial design using the `rand` property.

In most cases, the default swapping algorithm is preferred to generate an efficient design because it aims to satisfy attribute level balance, in contrast to the modified Fedorov algorithm. However, finding designs that satisfy all conditional constraints in a row-based algorithm (such as the swapping algorithm) is much harder than finding designs that satisfy all check constraints in a row-based algorithm (like the modified Fedorov algorithm). Therefore, while using the swapping algorithm in combination with conditional constraints is usually a good starting point, one may need to switch to the modified Fedorov algorithm in combination with check constraints in case Ngene is not able to locate a feasible design. In almost all cases, one can rewrite conditional constraints into check constraints (and vice versa) that apply the same prohibitions.

It is important to avoid constraints that introduce multicollinearity in the data, as this will result in an experimental design that cannot be used to estimate all parameters in the model. If this happens,

this is easily detected because the resulting design will have very poor efficiency, e.g., a very large or undefined (infinite) D-error. It is recommended to first generate a design without any constraints and then gradually add constraints, as it is easier to detect issues caused by constraints when specified one at the time.

6.2 Logical expressions

Constraints can be formulated in Ngene using *logical expressions*. Each logical expression can be *True* or *False*. In a logical expression, one can refer to a specific attribute of an alternative by first stating the alternative name, followed by a dot (`.`) and then the attribute name. For example, `laptopA.PRICE` refers to the `PRICE` attribute of the alternative `laptopA`.

Logical expressions can use the following mathematical symbols: `=` (equal to), `>` (greater than), `<` (smaller than), `>=` (greater than or equal to), `<=` (smaller than or equal to), `<>` (not equal to). In addition, the following arithmetic operators are supported: `+` (addition), `-` (subtraction).

Some examples of logical expressions:

- `laptopA.PRICE <> 1200`
- `laptopA.PRICE > laptopB.PRICE`
- `laptopA.PRICE + laptopB.PRICE <= 3000`

Although `*` (multiplication) and `/` (division) are currently not supported, it is often possible to rewrite the logical expression in terms of additions and/or subtractions. For example, `1.5 * laptop.PRICE > 1500` can simply be rewritten as `laptop.PRICE + laptop.PRICE + laptop.PRICE > 3000`.

Compounded logical expressions can be made using Boolean logical operators: `and`, `or`. For example:

- `laptopA.PRICE > 1200 and laptopA.PROCESSOR <= laptopB.PROCESSOR`
- `laptopA.PRICE > 1500 and laptopB.PRICE = 1800 and laptopB.STORAGE = 1`
- `laptopA.PRICE = 1200 or laptopA.PRICE = 1500`

6.3 Conditional constraints

Conditional constraints are if-then rules using two logical expressions, namely one logical expression for the 'if' part of the constraint and another logical expression for the 'then' part of the constraint. These constraints can be used when generating an efficient design, but *only in combination with the default swapping algorithm*. When conditional constraints are specified, Ngene will try to maintain attribute level balance, but this may not always be possible. Due to the nature of the column-based swapping algorithm, Ngene may not always be able to find a design that satisfies the conditional constraints, especially if many such constraints are specified. If one needs to impose many constraints, a better option may be to use check constraints in combination with the modified Fedorov algorithm; see Section 6.4.

In Ngene, conditional constraints are specified by the optional property `cond`. All constraints are specified in an environment that starts with a colon (`:`), similar to the `model` environment. In the `cond` environment, each property value is a conditional constraint represented by an if-then statement, separated by a comma (`,`). No commas should appear after the last if-then statement (Ngene generates an error otherwise). Each conditional constraint starts with `if` and is followed by two logical expressions between parentheses, as shown in the format below, where *logical expression 1* relates to the 'if' part and *logical expression 2* relates to the 'then' part.

```
1  design  ? Laptop choice example
2  ;alts = (laptopA, laptopB)
3  ;rows = 12
4  ;block = 2
5  ;eff = (mnl,d)
6  ;cond:
7  ? Laptops with Core i7/i9 processors cannot have a low price
8  if(laptopA.PROCESSOR >= 2, laptopA.PRICE >= 1500),
9  if(laptopB.PROCESSOR >= 2, laptopB.PRICE >= 1500),
10 ? Laptops with Core i3/i5 processors cannot have a high price
11 if(laptopA.PROCESSOR <= 1, laptopA.PRICE <= 1800),
12 if(laptopB.PROCESSOR <= 1, laptopB.PRICE <= 1800)
13 ;model:
14 U(laptopA, laptopB) = proc.dummy[-0.7|-0.5|-0.1] * PROCESSOR[0,1,2,3]
15                     + stor[0.0015]               * STORAGE[256,512,1024,2048]
16                     + cost[-0.003]               * PRICE[1200,1500,1800,2100]
17 ? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7),  3(Core i9)
18 ? STORAGE:    256 GB,     512 GB,     1024 GB,     2048 GB
19 ? PRICE:      $1200,      $1500,      $1800,       $2100
20 $
```

Script 6.1: Conditional constraints

```
;cond:
if( logical expression 1 , logical expression 2 ), ...
```

Ngene evaluates each choice task against all conditional constraints. A choice task passes a conditional constraint if both the *logical expression 1* and the *logical expression 2* are true, or if the *logical expression 1* is False (i.e., it is irrelevant). If a choice task passes all conditional constraints, it is accepted.

Consider again the laptop choice experiment, see Script 6.1. It may be unrealistic that a laptop with a fast processor is cheap and that a laptop with a slow processor is expensive. Therefore, to make laptop profiles more realistic, one could impose the following two constraints for a laptop:

1. If `PROCESSOR` ≥ 2, then `PRICE` ≥ 1500
2. If `PROCESSOR` ≤ 1, then `PRICE` ≤ 1800

Lines 8–9 in this script define Constraint 1 and lines 11–12 define Constraint 2. Note that Constraints 1 and 2 need to be applied to both alternatives, therefore, in total four conditional constraints need to be specified in the script.

There are often many ways to specify the same constraint. For example, we could replace the conditional constraint on line 8 in Script 6.1 with any of the following to yield the same result:

```
if(laptopA.PROCESSOR > 1, laptopA.PRICE <> 1200)

if(laptopA.PROCESSOR = 2 or laptopA.PROCESSOR = 3, laptopA.PRICE > 1200)

if(laptopA.PROCESSOR = [2,3], laptopA.PRICE = [1500,1800,2100])
```

The last constraint uses a shortcut that only works with conditional constraints: `X = [1,2,...]` is equivalent to `X = 1 or X = 2 or ...`, where `X` is an attribute of an alternative.

		Laptop A			Laptop B		
Choice task	Block	Processor	Storage	Price	Processor	Storage	Price
1	1	0	2048	1800	2	512	1500
2	1	1	512	1500	2	2048	1800
3	1	3	256	1500	2	2048	2100
4	1	2	512	1800	1	1024	1500
5	1	3	2048	2100	1	1024	1200
6	1	0	256	1200	1	512	1800
7	2	1	2048	1800	0	512	1200
8	2	0	1024	1500	3	256	1800
9	2	2	1024	2100	3	256	1500
10	2	1	256	1200	3	2048	2100
11	2	3	1024	1800	0	256	1500
12	2	2	512	1500	0	1024	1800

Table 6.1: D-efficient design based with conditional constraints for laptop choice example

The efficient design resulting is shown in Table 6.1, whereby laptops with processor levels 0 and 1 never have a price of $2100, and laptops with processor levels 2 or 3 never have a price of $1200.

6.4 Check constraints

Check constraints are a set of rules that use logical expressions. These constraints can be used when generating an efficient design, but *only in combination with the modified Fedorov algorithm*. Check constraints can also be applied when generating a full factorial design or random fractional factorial design.

In Ngene, check constraints are specified through the optional properties `reject` and/or `require`. All constraints are specified in an environment that starts with a colon (`:`), similar to the `cond` environment. In the `reject` and `require` environments, each property value is a check constraint represented by a logical expression, separated by a comma (`,`). No comma should appear after the last logical expression. Check constraints are specified in the following format.

```
;reject:
logical expression , ...

;require:
logical expression , ...
```

Ngene evaluates each choice task against all check constraints. When using `reject`, a choice task passes a check constraint when the *logical expression* is False. Similarly, when using `require`, a choice task passes a check constraint when the *logical expression* is True. If the choice task passes all check constraints, it is accepted.

Whether to use `reject` or `require` depends on the situation. To rule out undesirable choice tasks, which is the most common use of check constraints, one should use `reject`. One would use `require` to force certain attributes to have certain values, for example, to create a status quo alternative (see Section 6.5) or scenario variables (see Section 6.6). Both `reject` and `require` can appear in the same script.

```
1  design  ? Laptop choice example
2  ;alts = (laptopA, laptopB)
3  ;rows = 12
4  ;block = 2
5  ;eff = (mnl,d)
6  ;alg = mfedorov
7  ;reject:
8  ? Laptops with Core i7/i9 processors cannot have a low price
9  laptopA.PROCESSOR >= 2 and laptopA.PRICE = 1200,
10 laptopB.PROCESSOR >= 2 and laptopB.PRICE = 1200,
11 ? Laptops with Core i3/i5 processors cannot have a high price
12 laptopA.PROCESSOR <= 1 and laptopA.PRICE = 2100,
13 laptopB.PROCESSOR <= 1 and laptopB.PRICE = 2100
14 ;model:     ? using estimation coding
15 U(laptopA, laptopB) = proc.dummy[-0.7|-0.5|-0.1] * PROCESSOR[0,1,2,3]
16                     + stor[0.0015]               * STORAGE[256,512,1024,2048]
17                     + cost[-0.003]               * PRICE[1200,1500,1800,2100]
18 ? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7),  3(Core i9)
19 ? STORAGE:    256 GB,     512 GB,     1024 GB,     2048 GB
20 ? PRICE:      $1200,      $1500,      $1800,       $2100
21 $
```

Script 6.2: Check constraints

Let us consider again the two laptop constraints that were formulated as conditional constraints in Section 6.3. These conditional constraints can be reformulated as the following check constraints:

1. Reject: `PROCESSOR` ≥ 2 and `PRICE` $= 1200$
2. Reject: `PROCESSOR` ≤ 1 and `PRICE` $= 2100$

In other words, if a choice task contains a profile of a laptop with a fast processor together and a price of $1200, or a profile with a slow processor and a price of $2100, then Ngene rejects this choice task.

In Script 6.2 we now use the modified Fedorov (as indicated by `alg = mfedorov` in line 6). Lines 9–10 in this script define Constraint 1 and lines 12–13 define Constraint 2. As before, these constraints need to be applied to both alternatives, so in total four check constraints are needed.

There are again various ways of specifying the same constraint. For example, the following reject constraints for line 9 in Script 6.1 would yield the same result:

```
laptopA.PROCESSOR >= 2 and laptopA.PRICE = 1200

laptopA.PROCESSOR > 1 and laptopA.PRICE < 1500

laptopA.PRICE = 1200 and laptopA.PROCESSOR = 2 or laptopA.PROCESSOR = 3
```

Table 6.2 shows the resulting D-efficient design with a D-error of 0.004114. Similarly to Table 6.1, laptops with processor levels 0 and 1 never have a price of $2100, and laptops with processor levels 2 or 3 never have a price of $1200. Since the modified Fedorov algorithm will usually result in designs with low attribute level balance for quantitative attributes, one may want to impose attribute level frequency constraints as discussed in Section 5.4.

		Laptop A			Laptop B		
Choice task	Block	Processor	Storage	Price	Processor	Storage	Price
1	1	3	256	1500	0	512	1800
2	1	1	2048	1800	3	512	1500
3	1	1	2048	1800	0	512	1200
4	1	3	2048	2100	1	256	1200
5	1	3	512	1500	0	2048	1800
6	1	0	512	1200	2	1024	2100
7	2	3	2048	2100	2	256	1500
8	2	2	2048	2100	0	256	1200
9	2	3	512	2100	2	256	1500
10	2	1	512	1800	2	256	1500
11	2	1	256	1200	0	2048	1800
12	2	1	1024	1200	2	2048	2100

Table 6.2: D-efficient design based with check constraints for laptop choice example

6.5 Status quo alternatives

As defined in Section 1.1, a status quo alternative is an existing alternative described by a fixed profile. An example of a status quo alternative is shown in Figure 1.2, where 'Active surveillance' indicates the current situation of monitoring without treating.

Although adding a status quo alternative to the choice model does not influence the generation of other design types, it does influence the generation of an efficient design because it affects the choice probabilities. The way a status quo alternative is specified in Ngene depends on the algorithm used to generate the efficient design, as well as on the type of attribute (quantitative or qualitative).

Script 6.3 generates an efficient design assuming informative local priors for a treatment choice experiment with two treatment options, namely 'radiotherapy' (`radio`) and 'surgery' (`surgery`), and 'active surveillance' (`active`) as a non-treatment status quo alternative. There are only two attributes, namely side effects of treatment (`SIDEEFFECTS`) and probability of curing the patient (`PROBCURE`). The modified Fedorov algorithm (using a candidate set size of 256[1]) was chosen for design generation. To ensure reasonable attribute level balance, attribute level frequency constraints were imposed in the utility function on line 11 (see Section 5.4).

The status quo alternative has fixed attribute levels, namely, no side effects (`SIDEEFFECTS = 0`, which is the base level) and zero probability of curing the patient (`PROBCURE = 0`). For quantitative variables such as `PROBCURE`, it is easy to fix to a specific level by specifying a new attribute with one level only. In line 19 of Script 6.3, `PROBCURE_SQ` was defined with a single level (`0`), multiplied by the same parameter `cure` as `PROBCURE` in the utility functions of the other alternatives. For the qualitative attribute `SIDEEFFECTS`, we specified the following check constraint on lines 7–8 of the script to fix its level to 0 for the status quo alternative:

```
;require:
active.SIDEEFFECTS = 0
```

[1]Note that the full factorial contains $4^4 = 256$ choice tasks, therefore the default candidate set size of 2000 could not be achieved and was therefore decreased to 256.

```
1  design  ? treatment choice example
2  ;alts = radio, surgery, active  ? radiotherapy, surgery, active surveillance
3  ;rows = 16
4  ;block = 2
5  ;eff = (mnl,d)
6  ;alg = mfedorov(candidates = 256)
7  ;require:
8  active.SIDEEFFECTS = 0
9  ;model:
10 U(radio)   = side.dummy[-0.2|-0.4|-0.9] * SIDEEFFECTS[1,2,3,0]
11            + cure[0.02]                 * PROBCURE[30,50,70,90](3-5,3-5,3-5,3-5)
12            /
13 U(surgery) = cons[-0.6]
14            + side                       * SIDEEFFECTS
15            + cure                       * PROBCURE
16            /
17 U(active)  = cona[-0.5]
18            + side                       * SIDEEFFECTS
19            + cure                       * PROBCURE_SQ[0]
20 ? SIDEEFFECTS: 0(None),    1(Mild),    2(Moderate), 3(Severe)
21 ? PROBCURE:    0% (SQ),    30%,        50%,         70%,         90%
22 $
```

Script 6.3: Status quo alternative

The resulting efficient design is shown in Table 6.3. As expected, the last two columns show fixed attribute levels for status quo alternative 'active surveillance'.

When using the default swapping algorithm, check constraints cannot be imposed to fix status quo levels of qualitative attributes. Instead, for each dummy coded qualitative attribute one needs to ensure that the status quo level is the base level. In this case, the attribute can simply be omitted from the utility function of the status quo alternative as it defaults to zero.

In Script 6.3 we assumed that 'radiotherapy' and 'surgery' could have no side effects. Now suppose that these treatment options are expected to always have some side effects. A common mistake is that one naively expands the check constraints on lines 7–8 in Script 6.3 to:

```
;require:
radio.SIDEEFFECTS > 0,
surgery.SIDEEFECTS > 0,
active.SIDEEFFECTS = 0
```

Running the updated descript would produce a design with an Undefined (infinite) D-error. The reason is that base level 0 (no side effects) only appears in 'active surveillance', and since this attribute was dummy coded it now represents an additional constant in the utility function. As a result, the model becomes overspecified, and the parameters are not identifiable in model estimation. Changing to effects coding or using a different normalisation of the constants will not resolve the issue. The solution is to absorb level 0 (no side effects) in the constant (`cona`) of 'active surveillance' and remove this level from attribute `SIDEEFFECTS` in the utility functions of 'radiotherapy' and 'surgery', see Script 6.4. Since the base level of the dummy coded attribute `SIDEEFFECTS` has now changed, several priors also require consistent updating.

		Radiotherapy		Surgery		Active surveillance	
Choice task	Block	Side eff	Cure prob	Side eff	Cure prob	Side eff	Cure prob
1	1	1	30	3	90	0	0
2	1	2	30	1	90	0	0
3	1	0	90	1	30	0	0
4	1	3	30	2	90	0	0
5	1	2	50	3	70	0	0
6	1	2	90	0	30	0	0
7	1	2	90	0	50	0	0
8	1	0	70	2	30	0	0
9	2	1	50	0	90	0	0
10	2	3	90	2	30	0	0
11	2	0	50	1	70	0	0
12	2	3	90	0	50	0	0
13	2	0	30	2	70	0	0
14	2	3	30	1	50	0	0
15	2	1	70	2	30	0	0
16	2	1	70	3	90	0	0

Table 6.3: D-efficient design with status quo alternative for treatment choice example

```
design  ? treatment choice example
;alts = radio, surgery, active  ? radiotherapy, surgery, active surveillance
;rows = 16
;block = 2
;eff = (mnl,d)
;alg = mfedorov(candidates = 144)
;model:
U(radio)   = side.dummy[-0.2|-0.7] * SIDEEFFECTS[2,3,1]
           + cure[0.02]            * PROBCURE[30,50,70,90](3-5,3-5,3-5,3-5)
           /
U(surgery) = cons[-0.6]
           + side                  * SIDEEFFECTS
           + cure                  * PROBCURE
           /
U(active)  = cona[-0.3]
           + cure                  * PROBCURE_SQ[0]
? SIDEEFFECTS: 0(None),    1(Mild),    2(Moderate), 3(Severe)
? PROBCURE:    0% (SQ),    30%,        50%,         70%,         90%
$
```

Script 6.4: Specifying an identifiable model

```
1  design  ? mode choice example
2  ;alts = walk, bike, car
3  ;rows = 12
4  ;block = 2
5  ;eff = (mnl,d)
6  ;con
7  ;model:
8  U(walk) = conw[0.3]                              ? Constant for walk
9          + wtime[-0.03]        * WTIME[20,30,40]  ? Walking time (min)
10         + ww.dummy[-0.1|-0.4] * WEATHER[1,2,0]   ? Weather: 0 = Sun (base), 1 = Wind, 2 = Rain
11         /
12 U(bike) = conb[0.2]                              ? Constant for bus
13         + btime[-0.02]        * BTIME[10,15,20]  ? Riding time (min)
14         + wb.dummy[-0.4|-0.6] * WEATHER[WEATHER]
15         /
16 U(car)  = ctime[-0.015]       * TTIME[6,8,10]    ? Driving time (min)
17         + cost[-0.2]          * FUEL[1,2,3]      ? Fuel cost ($)
18 $
```

Script 6.5: Scenario variable with default swapping algorithm

6.6 Scenario variables

As stated in Chapter 1, a scenario describes the context in which an agent makes a decision. A scenario variable differs from an attribute in that it does not relate to a specific alternative, but rather the choice context is constant across all alternatives.

To create a scenario variable in Ngene, it needs to be added to one or more utility functions in the `model` property, and then its value must be restricted to be the same across all alternatives.

Script 6.5 generates an efficient design for a mode choice experiment whereby scenario variable `WEATHER` describes the weather conditions (sun, wind, rain). The weather condition is not an attribute of any of the alternatives (`walk`, `bike`, or `car`), but rather a context that is the same across all the alternatives. In this script, `WEATHER` is added as a variable in the utility functions of both `walk` and `bike`, but is left out as a main effect in alternative `car` for model identifiability reasons. The priors of the alternative-specific weather coefficients `ww` and `wb` indicate that during windy and rainy weather, walking and cycling become less attractive *compared to the car*.

In Script 6.5 the default swapping algorithm is used. To create a scenario variable, attribute levels are linked on line 14 using the syntax: `WEATHER[WEATHER]`. This means that the attribute `WEATHER` for alternative `bike` has the same level as the attribute `WEATHER` for alternative `walk`. Alternatively, one could specify the following conditional constraints in the syntax to define the scenario variable:

```
;cond:
if(walk.WEATHER = 0, bike.WEATHER = 0),
if(walk.WEATHER = 1, bike.WEATHER = 1),
if(walk.WEATHER = 2, bike.WEATHER = 2)
```

Table 6.4 shows the resulting efficient design. It is clear that the Weather columns for 'walk' and 'bike' have the same level. To illustrate what this would look like in a choice experiment, the first two choice tasks from Table 6.4 are shown in Figure 6.1. In the first choice task, the Weather level is 0 (Sun), while in the second choice task the Weather level is 2 (Rain).

Script 6.6 generates an efficient design for the same example of mode choice, but now using the modified Fedorov algorithm (with attribute level frequency constraints `(4,4,4)`). In this case, the

		Walk		Bike		Car	
Choice task	Block	Time	Weather	Time	Weather	Time	Fuel cost
1	1	20	0	15	0	6	1
2	1	40	2	20	2	10	3
3	1	20	1	10	1	8	2
4	1	30	1	20	1	10	1
5	1	40	0	10	0	8	3
6	1	30	2	15	2	6	2
7	2	40	0	15	0	10	1
8	2	40	1	20	1	6	3
9	2	20	0	20	0	8	3
10	2	30	2	15	2	6	1
11	2	30	1	10	1	8	2
12	2	20	2	10	2	10	2

Table 6.4: D-efficient design with scenario variable for mode choice example

Choice task 1. Suppose that is *sunny weather*. How would you like to travel to work?

Walk	Bicycle	Car
20 minutes walking	15 minutes riding	6 minutes driving $1 fuel cost
◉	○	○

Choice task 2. Suppose that is *rainy weather*. How would you like to travel to work?

Walk	Bicycle	Car
40 minutes walking	20 minutes riding	10 minutes driving $3 fuel cost
○	○	◉

Figure 6.1: Mode choice tasks with varying scenarios

```
design  ? mode choice example
;alts = walk, bike, car
;rows = 12
;block = 2
;eff = (mnl,d)
;alg = mfedorov
;require:
walk.WEATHER = bike.WEATHER
;con
;model:
U(walk) = conw[0.3]                                    ? Constant for walk
        + wtime[-0.03]        * WTIME[20,30,40](4,4,4)  ? Walking time (min)
        + ww.dummy[-0.1|-0.4] * WEATHER[1,2,0]   ? Weather: 0 = Sun (base), 1 = Wind, 2 = Rain
        /
U(bike) = conb[0.2]                                    ? Constant for bus
        + btime[-0.02]        * BTIME[10,15,20](4,4,4)  ? Riding time (min)
        + wb.dummy[-0.4|-0.6] * WEATHER
        /
U(car)  = ctime[-0.015]       * TTIME[6,8,10](4,4,4)    ? Driving time (min)
        + cost[-0.2]          * FUEL[1,2,3](4,4,4)      ? Fuel cost ($)
$
```

Script 6.6: Scenario variable with modified Fedorov algorithm

A patient is *50 years old* and has a *pre-existing heart disease*. As his doctor, which treatment option for prostate cancer do you think is best for this patient?		
Radiotherapy	Surgery	Active surveillance
Mild side effects	Severe side effects	No side effects
70% chance of curing patient	90% chance of curing patient	0% chance of curing patient
◉	○	○

Figure 6.2: Treatment choice task with multiple scenario variables

variable `WEATHER` has been defined as a regular variable in the utility functions of `walk` and `bike` , but the following check constraint is specified on lines 7–8 to ensure that `WEATHER` has the same level across both alternatives in each choice task:

```
;require:
walk.WEATHER = bike.WEATHER
```

Figure 6.2 shows an example choice task in the treatment choice experiment where we added two scenario variables that vary from choice task to choice task. In this example, the choice for 'radiotherapy', 'surgery', or 'active surveillance' may depend on the age of the patient and whether the patient has pre-existing heart disease or not. A design for this choice experiment can be generated with Script 6.7. Scenario variables `AGE` and `HEARTDISEASE` have been added to the utility functions of `radio` and `surgery`, whereby the priors of their parameters `age_r`, `age_s`, `heart_r`, and `heart_s` indicate the beliefs that 'active surveillance' is more preferably for older patients, 'surgery' is less (and 'radiotherapy is more) preferred for patients with a heart disease. The attribute levels of the scenario variables are linked together in lines 9—-10 of the script.

```
design  ? treatment choice example
;alts = radio, surgery, active  ? radiotherapy, surgery, active surveillance
;rows = 16
;block = 2
;eff = (mnl,d)
;alg = mfedorov
;require:
active.SIDEEFFECTS = 0,
radio.AGE = surgery.AGE,
radio.HEARTDISEASE = surgery.HEARTDISEASE
;model:
U(radio)   = conr[0.5]
           + side.dummy[-0.2|-0.4|-0.9] * SIDEEFFECTS[1,2,3,0]
           + cure[0.02]                 * PROBCURE[30,50,70,90](3-5,3-5,3-5,3-5)
           + age_r[-0.01]               * AGE[40,50,60,70](4,4,4,4)
           + heart_r.dummy[0.1]         * HEARTDISEASE[1,0]
           /
U(surgery) = cons[-0.1]
           + side                       * SIDEEFFECTS
           + cure                       * PROBCURE
           + age_s[-0.02]               * AGE[40,50,60,70](4,4,4,4)
           + heart_s.dummy[-0.3]        * HEARTDISEASE[1,0]
           /
U(active)  = side                       * SIDEEFFECTS
           + cure                       * PROBCURE_SQ[0]
? SIDEEFFECTS:  0(None),  1(Mild),  2(Moderate), 3(Severe)
? PROBCURE:     0%,       30%,      50%,         70%,      90%
? AGE:          30 years, 40 years, 50 years,    60 years
? HEARTDISEASE: 0(No),    1(Yes)
$
```

Script 6.7: Status quo alternative and multiple scenario variables

While scenario variables can be added as main effects in $J-1$ alternatives in a labelled choice experiment, whereby J is the number of alternatives, scenario variables can only be added as interaction effects in an unlabelled choice experiment. An example is shown in Script 6.8, in which we added a qualitative scenario variable called `PURPOSE` that indicates the context of purchase of a laptop, namely home use (level 0) or office use (level 1). This scenario variable is interacted with the `PRICE`, whereby the prior of the interaction parameter `cost_x_purpose` indicates the belief that consumers are less price sensitive when the laptop is purchased for office use.

Observe that `PURPOSE` only appears in an interaction effect in Script 6.8; adding it as a main effect in both alternatives would make the model unidentified, while putting it as a main effect in only one unlabelled alternative does not have a meaningful interpretation. Instead of interacting `PURPOSE` only with `PRICE`, it could also interact with the other attributes as shown in the syntax below (see also Section 3.9). This would increase the number of parameters, and therefore one would want to increase the design size.

```
design  ? laptop choice example
;alts = (laptopA, laptopB)
;rows = 12
;block = 2
;eff = (mnl,d)
;model:
U(laptopA) = proc.dummy[-0.7|-0.5|-0.1] * PROCESSOR[0,1,2,3]
           + stor[0.0015]               * STORAGE[256,512,1024,2048]
           + cost[-0.003]               * PRICE[1200,1500,1800,2100]
           + cost_x_purpose[0.001]      * PRICE * PURPOSE[0,1]
           /
U(laptopB) = proc                       * PROCESSOR
           + stor                       * STORAGE
           + cost                       * PRICE
           + cost_x_purpose             * PRICE * PURPOSE[PURPOSE]
? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7),  3(Core i9)
? STORAGE:    256 GB,     512 GB,     1024 GB,     2048 GB
? PRICE:      $1200,      $1500,      $1800,       $2100
? PURPOSE:    0(Home),    1(Office)
$
```

Script 6.8: Scenario variable in unlabelled experiment

```
U(laptopA)  = proc.dummy[-0.7|-0.5|-0.1] * PROCESSOR[0,1,2,3]
            + stor[0.0015]               * STORAGE[256,512,1024,2048]
            + cost[-0.003]               * PRICE[1200,1500,1800,2100]
            + cost_x_purpose[0.001]      * PRICE              * PURPOSE[0,1]
            + proc0_x_purpose            * PROCESSOR.level[0] * PURPOSE
            + proc1_x_purpose            * PROCESSOR.level[1] * PURPOSE
            + proc2_x_purpose            * PROCESSOR.level[2] * PURPOSE
            + stor_x_purpose             * STORAGE            * PURPOSE
```

If PURPOSE had more than two levels, then even more interaction effects would be required. In that case, in the script one would first need to define PURPOSE as a dummy or effects-coded attribute before it can be used in an interaction. This is explained in more detail in Section 7.2.

6.7 Attribute level overlap

When a choice experiment has more attributes, the complexity of each choice task increases, increasing the cognitive burden on agents. A strategy to simplify choice tasks is to create *attribute level overlap* across unlabelled alternatives, see also Sections 1.2 and 1.5.

Attribute level overlap can be imposed via check constraints. Consider the laptop choice experiment and assume that each unlabelled alternative (A, B) has five attributes, namely processor (PROC), storage capacity (STOR), screen size (SCRN), internal memory (MEM), and price (PRICE). Script 6.9 generates an efficient design in which each choice task has exactly two overlapping attributes. This is achieved with a large number of reject constraints as shown on lines 7–34 of this script.

Table 6.5 shows the resulting design, in which the overlapping attributes are coloured light blue. In each choice task, an agent makes a decision by trading off on three different attributes, while the other attributes have the same level. Attributes that overlap vary between choice tasks. It can be seen that the processor attribute is often not overlapping, while storage capacity and price often have overlapping attribute levels. This is a result of the efficient design generation process and depends on the priors specified for each attribute, as well as the coding of each attribute. In this example, we

		Laptop A					Laptop B				
Task	Block	Proc	Stor	Screen	Memory	Price	Proc	Stor	Screen	Memory	Price
1	1	3	2048	13	16	2100	0	512	13	16	1200
2	1	3	512	15	16	1200	3	2048	13	16	2100
3	1	0	1024	15	32	2100	2	1024	17	8	2100
4	1	0	2048	17	32	1500	1	2048	15	8	1500
5	1	2	256	13	16	1200	0	2048	13	16	2100
6	1	1	1024	13	32	2100	2	1024	15	8	2100
7	1	3	1024	13	8	1500	2	1024	17	32	1500
8	1	1	512	17	8	2100	3	512	13	32	2100
9	2	2	256	13	8	1800	1	256	15	32	1800
10	2	0	256	13	16	1500	1	2048	13	16	2100
11	2	0	2048	17	8	2100	3	256	17	32	2100
12	2	3	2048	17	8	1800	1	2048	13	32	1800
13	2	2	512	13	32	1800	3	512	17	8	1800
14	2	1	512	13	32	1200	0	2048	13	8	1200
15	2	3	2048	15	32	2100	1	2048	17	32	1500
16	2	3	1024	13	8	1500	2	1024	15	8	2100

Table 6.5: Explicit partial profile design with 2 overlapping attributes in each choice task

You are looking to buy a new laptop. Which of the following laptops would you prefer?	
Laptop A	Laptop B
Intel Core i7 processor	Intel Core i3 processor
2048 GB hard-disk drive	512 GB hard-disk drive
13" screen size	13" screen size
16 GB RAM	16 GB RAM
$2100	$1200
○	◉

Figure 6.3: Laptop choice task with 2 overlapping attributes

chose a relatively small design size of 16, but with overlapping attributes a larger number of rows would be preferred as each choice task now captures less information.

The design in Table 6.5 is also referred to as an *explicit partial profile design* whereby overlapping attributes are explicitly shown in each choice task. Figure 6.3 illustrates what the first choice task in Table 6.5 could look like in a survey.

If one would omit overlapping attributes, one would obtain what is called an *implicit partial profile* design in Table 6.6. Although a choice task with implicit partial profiles is simpler (in terms of cognitive burden on an agent) than a choice task with explicit partial profiles, implicit partial profiles should not be used in the presence of one or more labelled alternatives (such as an opt-out alternative) or in the presence of potential interaction effects.

Script 6.9 contains the property `alg = mfedorov` and by default creates a candidate set of 2000 choice tasks (see Section 5.4) that satisfy all specified check constraints in this script. To be able

```
design  ? Laptop choice example
;alts = (A, B)
;rows = 16
;block = 2
;eff = (mnl,d)
;alg = mfedorov
;reject:
? Cannot have 5 overlapping attributes
A.PROC= B.PROC and A.STOR= B.STOR and A.SCRN= B.SCRN and A.MEM= B.MEM and A.PRICE= B.PRICE,
? Cannot have 4 overlapping attributes
A.PROC<>B.PROC and A.STOR= B.STOR and A.SCRN= B.SCRN and A.MEM= B.MEM and A.PRICE= B.PRICE,
A.PROC= B.PROC and A.STOR<>B.STOR and A.SCRN= B.SCRN and A.MEM= B.MEM and A.PRICE= B.PRICE,
A.PROC= B.PROC and A.STOR= B.STOR and A.SCRN<>B.SCRN and A.MEM= B.MEM and A.PRICE= B.PRICE,
A.PROC= B.PROC and A.STOR= B.STOR and A.SCRN= B.SCRN and A.MEM<>B.MEM and A.PRICE= B.PRICE,
A.PROC= B.PROC and A.STOR= B.STOR and A.SCRN= B.SCRN and A.MEM= B.MEM and A.PRICE<>B.PRICE,
? Cannot have 3 overlapping attributes
A.PROC<>B.PROC and A.STOR<>B.STOR and A.SCRN= B.SCRN and A.MEM= B.MEM and A.PRICE= B.PRICE,
A.PROC<>B.PROC and A.STOR= B.STOR and A.SCRN<>B.SCRN and A.MEM= B.MEM and A.PRICE= B.PRICE,
A.PROC<>B.PROC and A.STOR= B.STOR and A.SCRN= B.SCRN and A.MEM<>B.MEM and A.PRICE= B.PRICE,
A.PROC<>B.PROC and A.STOR= B.STOR and A.SCRN= B.SCRN and A.MEM= B.MEM and A.PRICE<>B.PRICE,
A.PROC= B.PROC and A.STOR<>B.STOR and A.SCRN<>B.SCRN and A.MEM= B.MEM and A.PRICE= B.PRICE,
A.PROC= B.PROC and A.STOR<>B.STOR and A.SCRN= B.SCRN and A.MEM<>B.MEM and A.PRICE= B.PRICE,
A.PROC= B.PROC and A.STOR<>B.STOR and A.SCRN= B.SCRN and A.MEM= B.MEM and A.PRICE<>B.PRICE,
A.PROC= B.PROC and A.STOR= B.STOR and A.SCRN<>B.SCRN and A.MEM<>B.MEM and A.PRICE= B.PRICE,
A.PROC= B.PROC and A.STOR= B.STOR and A.SCRN<>B.SCRN and A.MEM= B.MEM and A.PRICE<>B.PRICE,
A.PROC= B.PROC and A.STOR= B.STOR and A.SCRN= B.SCRN and A.MEM<>B.MEM and A.PRICE<>B.PRICE,
? Cannot have 1 overlapping attribute
A.PROC<>B.PROC and A.STOR<>B.STOR and A.SCRN<>B.SCRN and A.MEM<>B.MEM and A.PRICE= B.PRICE,
A.PROC<>B.PROC and A.STOR<>B.STOR and A.SCRN<>B.SCRN and A.MEM= B.MEM and A.PRICE<>B.PRICE,
A.PROC<>B.PROC and A.STOR<>B.STOR and A.SCRN= B.SCRN and A.MEM<>B.MEM and A.PRICE<>B.PRICE,
A.PROC<>B.PROC and A.STOR= B.STOR and A.SCRN<>B.SCRN and A.MEM<>B.MEM and A.PRICE<>B.PRICE,
A.PROC= B.PROC and A.STOR<>B.STOR and A.SCRN<>B.SCRN and A.MEM<>B.MEM and A.PRICE<>B.PRICE,
? Cannot have 0 overlapping attributes
A.PROC<>B.PROC and A.STOR<>B.STOR and A.SCRN<>B.SCRN and A.MEM<>B.MEM and A.PRICE<>B.PRICE
;model:
U(A,B) = proc.dummy[-0.7|-0.5|-0.1] * PROC[0,1,2,3]
       + stor[0.0015]               * STOR[256,512,1024,2048]
       + scrn.dummy[-0.2|0.3]       * SCRN[13,15,17]
       + mem[0.03]                  * MEM[8,16,32]
       + cost[-0.003]               * PRICE[1200,1500,1800,2100]
? PROC:   0(Core i3), 1(Core i5), 2(Core i7),  3(Core i9)
? STOR:   256 GB,     512 GB,     1024 GB,     2048 GB
? SCRN:   13",        15",        17"
? MEM:    8 GB,       16 GB,      32 GB
? PRICE:  $1200,      $1500,      $1800,       $2100
$
```

Script 6.9: Design with two overlapping attributes

		Laptop A					Laptop B				
Task	Block	Proc	Stor	Screen	Memory	Price	Proc	Stor	Screen	Memory	Price
1	1	3	2048			2100	0	512			1200
2	1		512	15		1200		2048	13		2100
3	1	0		15	32		2		17	8	
4	1	0		17	32		1		15	8	
5	1	2	256			1200	0	2048			
6	1	1		13	32		2		15	8	
7	1	3		13	8		2		17	32	
8	1	1		17	8		3		13	32	
9	2	2		13	8		1		15	32	
10	2	0	256			1500	1	2048			2100
11	2	0	2048		8		3	256		32	
12	2	3		17	8		1		13	32	
13	2	2		13	32		3		17	8	
14	2	1	512		32		0	2048		8	
15	2	3		15		2100	1		17	1500	
16	2	3		13		1500	2		15		2100

Table 6.6: Implicit partial profile design with 2 overlapping attributes in each choice task

to inspect this candidate set before generating an efficient design, it is often useful to separate candidate set generation from efficient design generation. This can be achieved by creating two scripts as illustrated below. Script 1 first generates a random candidate set in a design named, say, 'My Candidate Set', taking into account any specified check constraints. Then script 2 reads that candidate set and generates an efficient design using the modified Fedorov algorithm.

```
design  ? Script 1: Generates candidate set
;rows = 2000
;rand
;reject:
...
```

```
design  ? Script 2: Generates efficient design
;rows = 16
;alg = mfedorov(candidates = My Candidate Set)
...
```

Writing check constraints to create attribute level overlap, such as on lines 7–34 in Script 6.9, may be quite tedious, especially if the number of attributes is large. Instead of manually typing these constraints, it may be more practical to generate the desired candidate set outside Ngene, see Section 6.9.

6.8 Availability of alternatives

When a choice experiment has many alternatives, choice task complexity is high and agents may start adopting simplifying decision rules, which is undesirable. A strategy to simplify choice tasks is to make only a subset of alternatives *available* in a choice task, which results in a *partial choice set design*, see also Section 1.2.

Which of the following available transport modes would you prefer to travel to work?				
Car	Train	Bus	Tram	Bike
15 min $3		25 min $1		40 min $0
○		○		◉

Figure 6.4: Mode choice task with 3 available alternatives

The availability of alternatives in a choice task can be imposed via check constraints. It involves a trick that ensures that some alternatives are assigned a zero choice probability in the model such that they mathematically drop out of the choice set in the efficient design generation process. Consider a mode choice experiment with five alternatives: `car`, `train`, `bus`, `tram`, and `bike`. Script 6.10 generates an efficient design in which exactly three alternatives are available in each choice task. This is achieved with a large number of `reject` constraints as shown on lines 8–35 of this script.

To assign a zero choice probability to a mode, we need to have the option to make an alternative very unattractive. This is achieved by allowing for a very large positive or negative level to be assigned to one of the attributes. For this trick to work, this attribute needs to be quantitative (not dummy or effects coded) and have a non-zero prior such that the utility can become very negative and hence would receive a zero choice probability. In Script 6.10 we chose to add level `999` to the cost attribute in each alternative.[2]

Table 6.7 shows the resulting design, where the unavailable alternatives are shown in light blue.[3] In each choice task, an agent chooses between three available alternatives, while the other alternatives are not available. The available choice options now vary across choice tasks. Since the modified Fedorov algorithm does not ensure attribute level balance, one may want to impose additional attribute level frequency constraints, see Section 5.4, with which one can also control how often each alternative must be available across all choice tasks. Figure 6.4 illustrates what the first choice task in Table 6.7 could look like in a survey.

Writing check constraints to define availability of alternatives, such as on lines 8–35 in Script 6.10, can be tedious if the number of alternatives is large. Instead of manually typing these constraints, it may be more practical to generate the desired candidate set outside Ngene; see Section 6.9.

6.9 External candidate sets

When constraints are complex, it may be easier to generate a candidate set outside of Ngene. For example, imposing overlap across many attributes (see Section 6.7), or defining the availability of many alternatives (see Section 6.8), or other types of constraints that cannot easily be formulated via conditional constraints or check constraints in Ngene.

A candidate set is a (typically large) design that contains allowable choice tasks. Suppose that one has created a candidate set with file name 'My Candidate Set' in Ngene design matrix format, which can be seen in Figure 2.13(a) and requires that the string 'Choice situation' appears in the

[2]For instance, if `cst1` equals `999` so this attracts a disutility of $-0.3 \times 999 \approx -300$, which yields a choice probability of zero for the car. If the prior value was smaller, say -0.0001, then a higher level such as 999999 should be chosen to allow a large negative utility.

[3]Whenever an alternative has an attribute level of 999, then it receives a zero choice probability and hence it is assumed to be unavailable.

```
design  ? Mode choice example
;alts = car, train, bus, tram, bike
;rows = 20
;block = 2
;eff = (mnl,d)
;alg = mfedorov
;con
;reject:
? Cannot have zero available alternatives
car.CST1= 999 and train.CST2= 999 and bus.CST3= 999 and tram.CST4= 999 and bike.CST5= 999,
? Cannot have one available alternative
car.CST1<>999 and train.CST2= 999 and bus.CST3= 999 and tram.CST4= 999 and bike.CST5= 999,
car.CST1= 999 and train.CST2<>999 and bus.CST3= 999 and tram.CST4= 999 and bike.CST5= 999,
car.CST1= 999 and train.CST2= 999 and bus.CST3<>999 and tram.CST4= 999 and bike.CST5= 999,
car.CST1= 999 and train.CST2= 999 and bus.CST3= 999 and tram.CST4<>999 and bike.CST5= 999,
car.CST1= 999 and train.CST2= 999 and bus.CST3= 999 and tram.CST4= 999 and bike.CST5<>999,
? Cannot have two available alternatives
car.CST1<>999 and train.CST2<>999 and bus.CST3= 999 and tram.CST4= 999 and bike.CST5= 999,
car.CST1<>999 and train.CST2= 999 and bus.CST3<>999 and tram.CST4= 999 and bike.CST5= 999,
car.CST1<>999 and train.CST2= 999 and bus.CST3= 999 and tram.CST4<>999 and bike.CST5= 999,
car.CST1<>999 and train.CST2= 999 and bus.CST3= 999 and tram.CST4= 999 and bike.CST5<>999,
car.CST1= 999 and train.CST2<>999 and bus.CST3<>999 and tram.CST4= 999 and bike.CST5= 999,
car.CST1= 999 and train.CST2<>999 and bus.CST3= 999 and tram.CST4<>999 and bike.CST5= 999,
car.CST1= 999 and train.CST2<>999 and bus.CST3= 999 and tram.CST4= 999 and bike.CST5<>999,
car.CST1= 999 and train.CST2= 999 and bus.CST3<>999 and tram.CST4<>999 and bike.CST5= 999,
car.CST1= 999 and train.CST2= 999 and bus.CST3<>999 and tram.CST4= 999 and bike.CST5<>999,
car.CST1= 999 and train.CST2= 999 and bus.CST3= 999 and tram.CST4<>999 and bike.CST5<>999,
? Cannot have four available alternatives
car.CST1<>999 and train.CST2<>999 and bus.CST3<>999 and tram.CST4<>999 and bike.CST5= 999,
car.CST1<>999 and train.CST2<>999 and bus.CST3<>999 and tram.CST4= 999 and bike.CST5<>999,
car.CST1<>999 and train.CST2<>999 and bus.CST3= 999 and tram.CST4<>999 and bike.CST5<>999,
car.CST1<>999 and train.CST2= 999 and bus.CST3<>999 and tram.CST4<>999 and bike.CST5<>999,
car.CST1= 999 and train.CST2<>999 and bus.CST3<>999 and tram.CST4<>999 and bike.CST5<>999,
? Cannot have five available alternatives
car.CST1<>999 and train.CST2<>999 and bus.CST3<>999 and tram.CST4<>999 and bike.CST5<>999
;model:
U(car)   = b1[0.3]                              ? car constant
         + b2[-0.05]  * TIME1[15,20,25]         ? car travel time
         + b3[-0.3]   * CST1[1,2,3,     999]    ? car travel cost
         /
U(train) = b4[0.2]                              ? train constant
         + b5[-0.04]  * TIME2[10,15,20]         ? train travel time
         + b3         * CST2[2,3,4,     999]    ? train travel cost
         /
U(bus)   = b6[-0.2]                             ? bus constant
         + b7[-0.06]  * TIME3[15,20,25]         ? bus travel time
         + b3         * CST3[1,2,3,     999]    ? bus travel cost
         /
U(tram)  = b8[0.1]                              ? tram constant
         + b9         * TIME4[10,15,20]         ? tram travel time
         + b3         * CST4[1,2,3,     999]    ? tram travel cost
         /
U(bike)  = b10[-0.08] * TIME5[20,30,40]         ? bike travel time
         + b3         * CST5[0,         999]    ? bike travel cost
$
```

Script 6.10: Partial choice set design with three available alternatives

		Car		Train		Bus		Tram		Bike	
Task	Block	Time	Cost	Time	Cost	Time	Cost	Time	Cost	Time	Cost
1	1	15	3	20	999	25	1	10	999	40	0
2	1	20	999	15	999	15	1	20	3	20	0
3	1	20	999	20	3	15	3	15	999	40	0
4	1	20	999	10	4	25	1	15	999	20	0
5	1	25	1	10	2	20	999	20	3	30	999
6	1	25	2	20	2	25	999	10	3	40	999
7	1	25	3	20	999	25	1	15	999	40	0
8	1	15	1	10	4	15	3	20	999	30	999
9	1	15	3	20	999	15	999	10	1	20	0
10	1	25	3	10	2	25	3	10	999	40	999
11	2	25	3	10	999	15	3	20	999	40	0
12	2	15	1	20	999	15	1	10	3	40	999
13	2	25	1	20	4	25	999	15	999	20	0
14	2	20	999	20	2	25	1	10	3	20	999
15	2	15	3	10	2	25	999	10	1	40	999
16	2	25	999	20	4	25	1	10	999	40	0
17	2	15	1	10	999	25	999	20	3	20	0
18	2	25	999	10	4	15	1	10	999	40	0
19	2	25	999	20	4	15	1	15	999	20	0
20	2	15	3	20	2	20	999	20	1	40	999

Table 6.7: Partial choice set design with 3 available alternatives in each choice task

first row/column.[4] This can be an Excel spreadsheet (.XLSX or .XLS) or CSV file generated using Python, R, or Matlab for example. Then it can be imported into Ngene (see Section 2.5) and used in conjunction with the modified Fedorov algorithm to generate an efficient design via the syntax below.

```
;alg = mfedorov(candidates = My Candidate Set)
```

To be able to use an external candidate set, it is important that the alternatives and attributes in the candidate set appear exactly the same order as those defined in the script to generate an efficient design. In addition, all attribute levels in the external candidate set need to appear as an allowable attribute level in the script. If the generated design has an Undefined efficiency, then this is often a sign that too strict constraints have been imposed on the candidate set, resulting in multi-collinearity or identifiability issues.

[4]This format can be viewed when exporting a design to Excel or when downloading an example file from the Import Design screen (see Section 2.5).

7

Multiple model specifications

This chapter describes how to generate an efficient design that is optimised for multiple model specifications simultaneously. At the design generation phase, a choice model is characterised by a model type, utility functions, and parameter priors. Since one may be uncertain which model specification best reflects the model that one will ultimately estimate, it may be useful to optimise the design over a range of model specifications.

7.1 Different model specifications

Multiple specifications of utility functions and/or priors can be achieved by specifying multiple `model` properties in the script, whereby each model is named. For example, suppose that we would like to specify two different models named `m1` and `m2` (or any other names defined by the analyst). Then in the script, we would add the `model` property twice with the name of each model in parentheses as shown in the syntax below.

```
;model(m1):
...
;model(m2):
...
```

When specifying the `model` property multiple times, it is also necessary to specify the `alts` property multiple times, as shown below. This is necessary because the alternatives could be different in the model specifications.

```
;alts(m1) = ...
;alts(m2) = ...
```

Finally, the property `eff` needs to be adapted to indicate which (combination of) model(s) one would like to optimise. For example, in the following syntax we aim to optimise the design for both models `m1` and `m2` simultaneously, whereby the D-error of model `m1` is multiplied by 2. Ngene produces a single experimental design in which efficiency results are presented for each model separately in the Results screen.

```
;eff = 2*m1(mnl,d) + m2(mnl,d)
```

In addition to different utility specifications, models could also have different model types (see Section 5.1) and prior types (see Sections 5.3 and 5.5). In the syntax below `m1` is specified as a multinomial logit model with Bayesian priors, and model `m2` is a panel random parameter logit model with local priors.

```
;eff = m1(mnl,d,mean) + 0.5*m2(rppanel,d)
```

All other design properties are applied *across all model specifications*. This includes conditional constraints through the property `cond`, check constraints specified through the properties `reject` or `require`, Bayesian and random draws specified through properties `bdraws` and `rdraws`, etc.

7.1.1 Different utility specifications

Consider Script 7.1, which generates an efficient design for two models simultaneously, namely model `simple` as specified on lines 7–14 for which five parameters are estimated and model `elaborate` as specified on lines 15–28 for which 10 parameters are estimated. The model `elaborate` uses dummy coding for the attribute `STORAGE` and also considers interaction effects between the attributes `PROCESSOR` and `PRICE`. The levels of the attributes in model `elaborate` must be identical to the attribute levels specified in model `simple` or can simply be omitted. Parameter priors need to be specified in both models and will typically be different for different model specifications. If no priors are specified, they default to zero. Informative priors for each model specification can be obtained by estimating both models using data from a pilot study. In lines 2—-3 it is specified that both models have the same alternatives and in line 6 it is specified that the D-error of model `simple` carries a weight of 3. Note that this does not necessarily mean that the model `simple` is three times as important because efficiency is not comparable between different models and may have entirely different magnitudes.

Table 7.1 presents the generated efficient design. The D-error of this design for model `simple` is 0.003608, and has a weighted value of $3 \times 0.003608 = 0.010824$. The D-error for model `elaborate` is 0.008342 and therefore carries only slightly less weight in the optimisation process than model `simple`. The efficiency criterion for which the design was optimised is the weighted D-error with value $3 \times 0.003608 + 0.008342 = 0.019166$.

Another example is shown in Script 7.2, whereby model `allattributes` contains all attributes and model `nostorage` excludes the `STORAGE` attribute. Such a script may be of interest if one includes a certain attribute but is not sure if it will be included in the final model or not (as it may be unclear whether it is a relevant attribute or not). Ngene will now check both models for strictly dominant alternatives (see also Section 3.8), which are more likely to occur in model `nostorage` because it only contains two attributes. This is reflected in the smaller candidate set size in the modified Fedorov algorithm.

7.1.2 Different model types

Script 7.3 illustrates the generation of an efficient design for two model specifications with a different model type, namely a model named `mnl`, which is a multinomial logit model with Bayesian priors, and a model named `mxl`, which is a mixed logit model with random parameters and local priors. This script optimises the design for estimating both the `mnl` model and the `mxl` model. As discussed in Section 5.7, optimisation for mixed logit models is very computationally intensive. This script also shows that constraints can be imposed as usual; see Chapter 6.

```
1  design  ? Laptop choice example
2  ;alts(simple)    = (laptopA, laptopB)
3  ;alts(elaborate) = (laptopA, laptopB)
4  ;rows = 16
5  ;block = 2
6  ;eff = 3*simple(mnl,d) + elaborate(mnl,d)
7  ;model(simple):     ? simple model with 5 parameters
8  U(laptopA, laptopB) = proc.dummy[-0.7|-0.5|-0.1] * PROCESSOR[0,1,2,3]
9                      + stor[0.0015]               * STORAGE[256,512,1024,2048]
10                     + cost[-0.003]               * PRICE[1200,1500,1800,2100]
11 ;model(elaborate):  ? more elaborate model with 10 parameters
12 U(laptopA, laptopB) = proc.dummy[-0.6|-0.4|-0.2] * PROCESSOR
13                     + stor.dummy[-2.6|-1.8|-0.9] * STORAGE
14                     + cost[-0.002]               * PRICE
15                     + proc0_x_price[-0.002]      * PROCESSOR.level[0] * PRICE
16                     + proc1_x_price[-0.0015]     * PROCESSOR.level[1] * PRICE
17                     + proc2_x_price[-0.001]      * PROCESSOR.level[2] * PRICE
18 ? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7),  3(Core i9)
19 ? STORAGE:    256 GB,     512 GB,     1024 GB,     2048 GB
20 ? PRICE:      $1200,      $1500,      $1800,       $2100
21 $
```

Script 7.1: Different utility function specifications

Instead of optimising for both models, one could also optimise only for the `mnl` model by replacing the syntax on line 6 with the efficiency criterion below. This allows the script to run much faster, while in the Results screen one will still be able to inspect the design efficiency results for the `mxl` model.

```
;eff = mnl(mnl,d,mean)
```

7.1.3 Different alternatives

An example of a script whereby each model has a different set of alternatives is shown in Script 7.4. In the considered mode choice experiment, agents are asked to choose between a set of transport modes, namely `walk`, `bike`, and `car`. The model called `allmodes` assumes that all three modes are available to the agent. However, if the agent indicated earlier in the survey that they do not have a car or bicycle available, then that alternative is not shown. Model `nocar` represents the model whereby an agent does not have a car available, whereas model `nobike` is a model where the bike is omitted as an alternative. This script minimises the average D-error across the three specified models. Weights could be applied in the efficiency criterion on line 7 if one believes that certain mode choice sets are more likely than others.

Special attention should be paid to the specifications of the utility functions. In particular, the label-specific constants and the scenario variable `WEATHER` need to be added to the utility functions in such a way that the parameters in all three models are identifiable. If we add label-specific constants for alternatives `walk` and `bike` then the model `nocar` would have constants in both utility functions, resulting in a model that cannot be estimated. Similarly, if scenario variable `WEATHER` would be added to alternatives `walk` and `bike` then the parameters in model `nocar` would again not be identified. Since `walk` is the only alternative available in all three model specifications, we normalise the constant to zero for this alternative and also omit the scenario variable in this alternative.

		Laptop A			Laptop B		
Choice task	Block	Processor	Storage	Price	Processor	Storage	Price
1	1	0	2048	1800	1	1024	1500
2	1	2	256	1500	1	2048	2100
3	1	2	512	1800	3	256	1800
4	1	0	512	1200	1	2048	2100
5	1	3	1024	2100	3	512	1200
6	1	2	1024	2100	0	256	1200
7	1	1	2048	1200	3	1024	1500
8	1	2	256	1500	0	512	1800
9	2	3	256	1200	2	2048	1500
10	2	0	2048	2100	1	512	1800
11	2	3	512	1500	0	1024	1800
12	2	1	2048	1500	2	512	1200
13	2	1	1024	1800	3	256	1500
14	2	3	512	2100	0	1024	1200
15	2	1	256	1200	2	2048	2100
16	2	0	1024	1800	2	256	2100

Table 7.1: Efficient design for multiple model specifications

```
design  ? Laptop choice example
;alts(allattributes) = (laptopA, laptopB)
;alts(nostorage)     = (laptopA, laptopB)
;rows = 12
;block = 2
;eff = 2*allattributes(mnl,d) + nostorage(mnl,d)
;alg = mfederov(candidates = 1000)
;model(allattributes):  ? model containing all attributes
U(laptopA, laptopB) = proc.dummy[-0.7|-0.5|-0.1] * PROCESSOR[0,1,2,3]
                    + stor[0.0015]               * STORAGE[256,512,1024,2048]
                    + cost[-0.003]               * PRICE[1200,1500,1800,2100]
;model(nostorage):  ? model that excludes storage attribute
U(laptopA, laptopB) = proc.dummy[-0.9|-0.6|-0.2] * PROCESSOR[0,1,2,3]
                    + cost[-0.0035]              * PRICE[1200,1500,1800,2100]
? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7),  3(Core i9)
? STORAGE:    256 GB,     512 GB,     1024 GB,     2048 GB
? PRICE:      $1200,      $1500,      $1800,       $2100
$
```

Script 7.2: Omitted attribute

```
1  design  ? Laptop choice example
2  ;alts(mnl) = (laptopA, laptopB)
3  ;alts(mxl) = (laptopA, laptopB)
4  ;rows = 16
5  ;block = 2
6  ;eff = mnl(mnl,d,mean) + 0.3*mxl(rppanel,d)
7  ;bdraws = sobol(300)
8  ;rdraws = gauss(3)
9  ;rep = 500
10 ;alg = mfederov(candidates = 1000)
11 ;reject:
12 laptopA.PROCESSOR >= 2 and laptopA.PRICE = 1200,
13 laptopB.PROCESSOR >= 2 and laptopB.PRICE = 1200,
14 laptopA.PROCESSOR <= 1 and laptopA.PRICE = 2100,
15 laptopB.PROCESSOR <= 1 and laptopB.PRICE = 2100
16 ;model(mnl):  ? Multinomial logit model with Bayesian priors
17 U(laptopA, laptopB)
18    = proc.dummy[(u,-1,-0.6)|(u,-0.6,-0.4)|(u,-0.4,0)] * PROCESSOR[0,1,2,3]
19    + stor[(n,0.0015,0.0005)]                          * STORAGE[256,512,1024,2048]
20    + cost[(n,-0.003,0.001)]                           * PRICE[1200,1500,1800,2100]
21 ;model(mxl):  ? Mixed logit model with random parameters and local priors
22 U(laptopA, laptopB)
23    = proc.dummy[n,-0.7,0.4|n,-0.5,0.3|n,-0.1,0.2] * PROCESSOR
24    + stor[n,0.0015,0.0007]                        * STORAGE
25    + cost[n,-0.003,0.0012]                        * PRICE
26 ? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7),  3(Core i9)
27 ? STORAGE:    256 GB,     512 GB,     1024 GB,     2048 GB
28 ? PRICE:      $1200,      $1500,      $1800,       $2100
29 $
```

Script 7.3: Different model types and prior types

Another example in which models have different alternatives is shown in Script 7.5, which reflects the choice task shown in Figure 7.1 containing a forced and unforced choice. This script generates an efficient design for estimating models with and without an opt-out alternative. The opt-out alternative is named `neither` in the script, and its utility function contains only a constant (`none`). Model `unforced` is the model with the opt-out alternative and reflects the unforced choice set, while model `forced` is the model without the opt-out alternative and reflects the forced choice set. The efficiency criterion on line 6 in the script gives more weight to the unforced choice because the forced choice only becomes relevant when an agent first selects the opt-out alternative in the unforced choice set.

7.2 Auxiliary model specification

The ability to specify different utility functions offers additional flexibility beyond optimisation of a design for multiple models. Sometimes it is useful to specify an auxiliary model to first define all the variables in the model, while specifying another model to optimise the design for.

As mentioned in Sections 3.9 and 6.6, it is not possible within the current syntax capabilities in Ngene to define a dummy or effects-coded attribute (or scenario variable) when it only appears in an interaction effect. However, this limitation can be overcome by first specifying an auxiliary model that defines all variables in the model and then specifying the model of interest. To illustrate, in Script 7.6 we have defined an auxiliary model named `aux` and the model of interest is called `main`. On line 6 we indicate that the design should only be optimised for model `main`. The script includes the

```
design  ? mode choice example
;alts(allmodes) = walk, bike, car
;alts(nocar) = walk, bike
;alts(nobike) = walk, car
;rows = 12
;block = 2
;eff = allmodes(mnl,d) + nocar(mnl,d) + nobike(mnl,d)
;con
;model(allmodes): ? walk, bike and car available
U(walk) = wtime[-0.03]        * WTIME[20,30,40]  ? Walking time (min)
        /
U(bike) = conb[-0.1]                             ? Constant for bus
        + btime[-0.02]        * BTIME[10,15,20]  ? Riding time (min)
        + wb.dummy[-0.1|-0.4] * WEATHER[1,2,0]   ? Weather: 0 = Sun (base), 1 = Wind, 2 = Rain
        /
U(car)  = conc[-0.3]                             ? Constant for car
        + ctime[-0.015]       * TTIME[6,8,10]    ? Driving time (min)
        + cost[-0.2]          * FUEL[1,2,3]      ? Fuel cost ($)
        + wc.dummy[0.1|0.3]   * WEATHER[WEATHER]
;model(nocar): ? only walk and bike available
U(walk) = wtime[-0.04]        * WTIME
        /
U(bike) = conb[-0.15]
        + btime[-0.025]       * BTIME
        + wb.dummy[-0.1|-0.5] * WEATHER
;model(nobike): ? only walk and car available
U(walk) = wtime[-0.05]        * WTIME
        /
U(car)  = conc[-0.35]
        + ctime[-0.02]        * TTIME
        + cost[-0.25]         * FUEL
        + wc.dummy[0.15|0.4]  * WEATHER
$
```

Script 7.4: Different alternatives

		Walk	Bike		Car		
Choice task	Block	Time	Time	Weather	Time	Fuel	Weather
1	1	30	20	0	6	1	0
2	1	20	20	2	10	3	2
3	1	40	10	2	6	2	2
4	1	30	15	1	10	1	1
5	1	30	10	0	8	3	0
6	1	40	15	1	8	2	1
7	2	20	15	2	6	1	2
8	2	40	20	0	8	3	0
9	2	20	10	0	10	1	0
10	2	30	20	1	8	2	1
11	2	40	15	2	10	2	2
12	2	20	10	1	6	3	1

Table 7.2: Efficient design for models with different alternatives

```
design  ? Laptop choice example
;alts(forced)   = (laptopA, laptopB)
;alts(unforced) = (laptopA, laptopB), neither
;rows = 12
;block = 2
;eff = 2*unforced(mnl,d) + forced(mnl,d)
;model(unforced):
U(laptopA, laptopB) = proc.dummy[-0.7|-0.5|-0.1] * PROCESSOR[0,1,2,3]
                    + stor[0.0015]               * STORAGE[256,512,1024,2048]
                    + cost[-0.003]               * PRICE[1200,1500,1800,2100]
                    /
U(neither)          = none[-4.6]
;model(forced):
U(laptopA, laptopB) = proc.dummy[-0.7|-0.5|-0.1] * PROCESSOR
                    + stor[0.0015]               * STORAGE
                    + cost[-0.003]               * PRICE
? PROCESSOR:  0(Core i3), 1(Core i5), 2(Core i7),  3(Core i9)
? STORAGE:    256 GB,     512 GB,     1024 GB,     2048 GB
? PRICE:      $1200,      $1500,      $1800,       $2100
$
```

Script 7.5: Unforced and forced choice

scenario variable PURPOSE, and the check constraint on line 8 guarantees that this variable maintains the same level for both laptop options.

In the auxiliary model (aux) we added scenario variable PURPOSE as a main effect in the utility functions, simply to define it as a dummy-coded attribute. Model aux is actually not identified, as scenario variables can only appear as interaction effects in an unlabelled experiment. As a result, the D-error for model aux will always be Undefined (infinite), but this is not important since we are not optimising the design for this model.

In the model of interest (main) we can now interact the scenario variable PURPOSE with attribute PROCESSOR to investigate whether preferences towards processor speed depend on whether the laptop will be used at home, at the office, or both. Note that including interactions between dummy or effects coded variables requires interacting individual levels in the choice model as shown on lines 23–28 and 33–38.

You are considering to purchase a new laptop. Which of the following would you prefer?		
Laptop A	Laptop B	Neither
Intel Core i3 processor 1 TB hard-disk drive $1500	Intel Core i7 processor 512 GB hard-disk drive $1800	
○	○	◉
◉	○	

Figure 7.1: Laptop choice task with unforced and forced choice

```
design  ? Laptop choice example
;alts(aux)  = (laptopA, laptopB)
;alts(main) = (laptopA, laptopB)
;rows = 24
;block = 3
;eff = main(mnl,d)
;alg = mfedorov
;require: laptopA.purpose = laptopB.purpose
;model(aux):  ? auxiliary model
U(laptopA, laptopB) = proc.dummy[-0.7|-0.5|-0.1] * PROCESSOR[0,1,2,3]
                    + stor[0.0015]               * STORAGE[256,512,1024,2048]
                    + cost[-0.003]               * PRICE[1200,1500,1800,2100]
                    + purp.dummy                 * PURPOSE[1,2,0]
;model(main): ? main model of interest
U(laptopA, laptopB)
   = proc.dummy[-0.7|-0.5|-0.1] * PROCESSOR
   + stor[0.0015]               * STORAGE
   + cost[-0.003]               * PRICE
   + proc0_x_purp1              * PROCESSOR.level[0] * PURPOSE.level[1]
   + proc1_x_purp1              * PROCESSOR.level[1] * PURPOSE.level[1]
   + proc2_x_purp1              * PROCESSOR.level[2] * PURPOSE.level[1]
   + proc0_x_purp2              * PROCESSOR.level[0] * PURPOSE.level[2]
   + proc1_x_purp2              * PROCESSOR.level[1] * PURPOSE.level[2]
   + proc2_x_purp2              * PROCESSOR.level[2] * PURPOSE.level[2]
? PROCESSOR:  0(Core i3),   1(Core i5),   2(Core i7),    3(Core i9)
? STORAGE:    256 GB,       512 GB,       1024 GB,       2048 GB
? PRICE:      $1200,        $1500,        $1800,         $2100
? PURPOSE:    0(Home),      1(Office),    2(Home+Office)
$
```

Script 7.6: Auxiliary model to define variables

8

Considering population segments

This chapter describes how to generate designs that are optimised for multiple population segments. In most cases, it is not necessary to account for sociodemographic variables (e.g., gender, age) and/or socioeconomic variables (e.g., income) during the experimental design phase. But if one believes that different population segments have substantially different preferences, then one could consider generating designs that are optimised for different segments in the population.

8.1 Defining population segments

As shown in Equation (1.6) in Section 1.5, the Fisher information matrix for a single model is calculated as a sum over agents. If agents belonging to different population segments have different preferences, one could calculate the Fisher information matrix as a weighted sum over population segments, whereby each weight indicates the proportion of a segment within the population. This is achieved using the `fisher` property in Ngene. This property specifies how Fisher information should be calculated based on data from multiple population segments to estimate a single model.

In the syntax below, Fisher information across all population segments is combined in a matrix called `MyFisher` (or any other name) and is passed on to the `eff` property to determine the design efficiency; in this example the D-error for estimating a multinomial logit model. Although there is only a single underlying model when using the `fisher` property, different population segments in Ngene need to be defined by specifying multiple `model` properties. In this example, three population segments named `pop1`, `pop2`, and `pop3` are considered. In order to be able to pool the data from multiple population segments, all model specifications need to be identical in terms of alternatives, attributes, attribute levels and parameters, and can only differ with respect to characteristics of the population segments.

```
;eff = MyFisher(mnl,d)
;fisher(MyFisher) = ...
;model(pop1): ...
;model(pop2): ...
;model(pop3): ...
```

The specification of the `fisher` property depends on whether one would like to generate a *homogeneous design* or a *heterogeneous design*. A homogeneous design is a single design that is simultaneously optimised across multiple population segments, whereas a heterogeneous design is a design in which each population segment is given different choice tasks. These two design types are discussed in the following subsections.

Note that the property `fisher` cannot be used in conjunction with conditional constraints (via the `cond` property) or check constraints (via the properties `reject` and `require`) and is compatible only with the swapping algorithm, not the modified Fedorov algorithm.

8.2 Homogeneous designs

Consider again three population segments, `pop1`, `pop2`, and `pop3`. Each population segment requires an associated weight between 0 and 1, whereby the weights need to sum to 1. Suppose that the weights are respectively 0.7, 0.1 and 0.3, indicating that the first population segment 1 has a larger presence in the sample population than the second segment. In the following syntax, we instruct Ngene to generate a homogeneous design called `MyDesign` (or any other name) that is efficient when data from all three segments are combined with their respective weights.

```
;fisher(MyFisher) = MyDesign(pop1[0.7], pop2[0.1], pop3[0.3])
```

Instead of exogenously specifying weights for each population segment based on its expected proportion in the sample of the population, it is also possible to let Ngene determine optimal weights. Such optimal weights could inform the sampling strategy whereby more information can be gained by over- or under-sampling certain population segments. In the syntax below, Ngene will determine the optimal weights for each population segment between the lower bound 0.1 and the upper bound 0.9.[1]

```
;fisher(MyFisher) = MyDesign(pop1[0.1:0.9], pop2[0.1:0.9], pop3[0.1:0.9])
```

Script 8.1 generates a homogeneous design for a mode choice experiment by which two population segments are considered, namely `male` and `female`, each with a weight of 0.5 in the population as specified on line 8. Lines 10–18 specify the model for segment `male` and lines 19–28 specify the model for segment `female`. Note that both model specifications are identical, except for lines 11 and 21 where a variable called `GENDER` has been added with the suffix `.covar`. This suffix indicates a fixed value of this attribute and is equal to 1 in the model for segment `male` and 0 in the model for segment `female`, thus creating a dummy variable that is added as a main effect to the model. Its parameter, `gender`, has a prior value of 0.5, indicating that males are more likely to choose the alternative `car` than females. As usual, when specifying multiple `model` properties, property `alts` must be specified for each model as indicated on lines 2–3.

The output is a single design named `des1`, shown in Table 8.1, omitting the values for `GENDER`. This design has a D-error of 0.102213 when the data is pooled for male and female, but has an undefined (infinite) D-error for each model for male and female separately, because the parameter `gender` is not identified if all agents are only men or only women.

[1] Note that it must be possible to let these weights sum to 1. Therefore, the sum of the upper bounds in the models must be at least 1.

```
design ? mode choice example
;alts(male)   = car, train
;alts(female) = car, train
;rows = 16
;block = 2
;eff = all(mnl,d)
;con
;fisher(all) = des1(male[0.5], female[0.5])
;model(male):
U(car)   = con_car[0.3]                          ? constant for car
         + gender[0.5]   * GENDER.covar[1]       ? gender: 1=male, 0=female
         + ctime[-0.05]  * CTIME[10,15,20,25]    ? car driving time (min)
         + cost[-0.25]   * FUELTOLL[2,3,4]       ? fuel and toll cost ($)
         /
U(train) = ttime[-0.06]  * TTIME[5,10,15,20]     ? train in-vehicle time (min)
         + wait[-0.04]   * WAIT[5,10,15]         ? waiting time (min)
         + trans[-0.5]   * TRANSFER[0,1]         ? number of transfers
         + cost          * FARE[1,2,3]           ? train fare ($)
;model(female):
U(car)   = con_car[0.3]
         + gender[0.5]   * GENDER.covar[0]
         + ctime[-0.05]  * CTIME[10,15,20,25]
         + cost[-0.25]   * FUELTOLL[2,3,4]
         /
U(train) = ttime[-0.06]  * TTIME[5,10,15,20]
         + wait[-0.04]   * WAIT[5,10,15]
         + trans[-0.5]   * TRANSFER[0,1]
         + cost          * FARE[1,2,3]
$
```

Script 8.1: Homogeneous design with multiple population segments

		Car		Train			
Choice task	Block	Time	Fuel & toll	Time	Wait	Transfers	Fare
1	1	10	4	5	5	0	1
2	1	25	4	20	15	0	1
3	1	10	3	10	10	1	2
4	1	15	2	15	10	1	3
5	1	25	4	5	15	0	1
6	1	15	3	15	15	1	2
7	1	25	2	10	5	0	3
8	1	15	3	20	10	1	2
9	2	25	4	20	5	1	1
10	2	20	3	5	5	1	2
11	2	20	2	15	10	0	3
12	2	15	2	10	15	0	3
13	2	10	4	20	5	0	1
14	2	10	3	15	15	0	2
15	2	20	2	10	10	1	3
16	2	20	4	5	15	1	1

Table 8.1: Homogeneous design optimised across multiple population segments

8.3 Heterogeneous designs

Alternatively, one could ask Ngene to generate a heterogeneous design as shown in the syntax below, which produces three separate experimental designs called `Design1`, `Design2`, and `Design3`. Each design belongs to a specific population segment; for example, the choice tasks in `Design1` are given to agents in the population segment `pop1`. The three designs are jointly optimised to estimate a single model based on pooled data from all population segments.

```
;fisher(MyFisher) = Design1(pop1[0.7])) + Design2(pop2[0.1]) + Design3(pop3[0.3])
```

A hybrid of a homogeneous and heterogeneous design is also possible, as shown in the syntax below, which produces two experimental designs. `Design1` is given to agents in population segment `pop1`, while `MyDesign2` is given to agents in population segments `pop2` and `pop3`.

```
;fisher(MyFisher) = Design1(pop1[0.7])) + Design2(pop2[0.1]), pop3[0.3])
```

A heterogeneous design can be obtained by replacing line 8 of Script 8.1 with the following syntax.

```
;fisher(all) = des1(male[0.5])) + des2(female[0.5])
```

This generates two designs, namely, one for `male` and one for `female`, as shown in Table 8.2. Each design contains different choice tasks but uses the same attribute levels. This heterogeneous design has a D-error of 0.09674, which, as expected, is somewhat more efficient than the homogeneous design in Table 8.1.

More than one sociodemographic/economic variable can be used to define segments. In Script 8.2 a socioeconomic variable called `INCOME` is added as an interaction effect with cost attributes `FUELTOLL` and `FARE`. It has two categories, namely low income defined with an average of $30,000, and high income defined on average as $90,000. Together with the `GENDER` variable now four segments can be distinguished, namely male with low income (`male_low`), female with low income (`female_low`), male with high income (`male_high`), and female with high income (`male_low`), each with specific weights defined in the `fisher` property on lines 6–7. This script generates a design `des1` for low-income agents and a design `des2` for high-income agents, each optimised for both genders.

Male		Car		Train			
Choice task	Block	Time	Fuel & toll	Time	Wait	Transfers	Fare
1	1	15	3	15	15	1	2
2	1	15	2	10	15	0	3
3	1	25	2	5	5	0	3
4	1	20	2	15	10	1	3
5	1	20	3	20	10	1	2
6	1	25	4	20	5	1	1
7	1	10	4	5	5	0	1
8	1	10	4	10	15	0	1
9	2	25	3	20	5	0	2
10	2	25	4	5	15	0	1
11	2	20	4	5	5	1	1
12	2	10	3	20	5	0	2
13	2	15	2	15	10	1	3
14	2	15	2	15	10	1	2
15	2	10	3	10	10	1	2
16	2	20	2	10	15	0	3

Female		Car		Train			
Choice task	Block	Time	Fuel & toll	Time	Wait	Transfers	Fare
1	1	20	4	10	15	1	1
2	1	10	2	5	5	0	3
3	1	10	4	20	5	0	1
4	1	20	2	10	15	0	3
5	1	20	2	15	10	1	3
6	1	25	4	20	15	0	1
7	1	20	3	5	5	1	2
8	1	15	2	15	10	1	2
9	2	25	2	20	5	0	3
10	2	15	4	20	10	0	1
11	2	25	4	10	15	1	1
12	2	15	3	15	10	1	2
13	2	15	3	5	5	1	2
14	2	25	2	5	10	0	3
15	2	10	3	15	10	1	2
16	2	10	3	10	15	0	2

Table 8.2: Heterogeneous design with multiple population segments

```
design ? mode choice example
;alts(male_low)    = car, train   ;alts(female_low)  = car, train
;alts(male_high)   = car, train   ;alts(female_high) = car, train
;rows = 16
;eff = all(mnl,d)
;fisher(all) = des1(male_low[0.35],  female_low[0.25])
             + des2(male_high[0.25], female_high[0.15])
;model(male_low):
U(car)   = con_car[0.3]                              ? constant for car
         + gender[0.5]        * GENDER.covar[1]      ? gender: 1=male, 0=female
         + ctime[-0.05]       * CTIME[10,15,20,25]   ? car driving time (min)
         + cost[-0.25]        * FUELTOLL[2,3,4]      ? fuel and toll cost ($)
         + cost_x_inc[0.0005] * FUELTOLL * INCOME.covar[30]
         /
U(train) = ttime[-0.04]       * TTIME[5,10,15,20]    ? train in-vehicle time (min)
         + wait[-0.06]        * WAIT[5,10,15]        ? waiting time (min)
         + trans[-0.5]        * TRANSFER[0,1]        ? number of transfers
         + cost               * FARE[1,2,3]          ? train fare ($)
         + cost_x_inc         * FARE * INCOME.covar[30]
;model(female_low):
U(car)   = con_car[0.3]
         + gender[0.5]        * GENDER.covar[0]
         + ctime[-0.05]       * CTIME[10,15,20,25]
         + cost[-0.25]        * FUELTOLL[2,3,4]
         + cost_x_inc[0.0005] * FUELTOLL * INCOME.covar[30]
         /
U(train) = ttime[-0.04]       * TTIME[5,10,15,20]
         + wait[-0.06]        * WAIT[5,10,15]
         + trans[-0.5]        * TRANSFER[0,1]
         + cost               * FARE[1,2,3]
         + cost_x_inc         * FARE * INCOME.covar[30]
;model(male_high):
U(car)   = con_car[0.3]
         + gender[0.5]        * GENDER.covar[1]
         + ctime[-0.05]       * CTIME[10,15,20,25]
         + cost[-0.25]        * FUELTOLL[2,3,4]
         + cost_x_inc[0.0005] * FUELTOLL * INCOME.covar[90]
         /
U(train) = ttime[-0.04]       * TTIME[5,10,15,20]
         + wait[-0.06]        * WAIT[5,10,15]
         + trans[-0.5]        * TRANSFER[0,1]
         + cost               * FARE[1,2,3]
         + cost_x_inc         * FARE * INCOME.covar[90]
;model(female_high):
U(car)   = con_car[0.3]
         + gender[0.5]        * GENDER.covar[0]
         + ctime[-0.05]       * CTIME[10,15,20,25]
         + cost[-0.25]        * FUELTOLL[2,3,4]
         + cost_x_inc[0.0005] * FUELTOLL * INCOME.covar[90]
         /
U(train) = ttime[-0.04]       * TTIME[5,10,15,20]
         + wait[-0.06]        * WAIT[5,10,15]
         + trans[-0.5]        * TRANSFER[0,1]
         + cost               * FARE[1,2,3]
         + cost_x_inc         * FARE * INCOME.covar[90]
$
```

Script 8.2: Design with more population segments

9

Agent-specific attribute levels

This chapter describes how to generate designs in which the attribute levels in choice tasks are tailored to the choice context of an agent. Each agent makes decisions within their own context, and therefore relevant attribute levels may vary from agent to agent. For example, for parcel delivery choice, delivery times may depend on whether one lives in a metropolitan area or rural area; for mode choice to work, travel times depend on how far one lives from the workplace; for car insurance choice, insurance premiums depend on the value of the car that one owns; for medication choice, risk of side effects may depend on patient characteristics, etc. Choice tasks with familiar attribute levels also tend to reduce hypothetical bias in choice experiments. Consequently, it is advised to employ attribute levels tailored to the particular circumstances the agent finds itself in.

9.1 Homogeneous design

In this section, it is explained how to generate a single (homogeneous) design for all agents, even if their attribute levels vary. A homogeneous design may be useful when the aim is to conduct a within-subject study, such that any differences found can be attributed to variations in the choice context.

In the absence of parameter priors, one could simply use design coding to generate a single orthogonal or efficient design. Once the design has been generated, one can replace (relabel) the design coding with the relevant attribute levels for each agent in the survey instrument.

Consider an experiment of public transport choice with two unlabelled alternatives in the context of travelling to work. Different agents have different commuting distances and therefore should be shown different attribute levels depending on their reported distance to work. Suppose that the analyst asks the agent in the survey how long their commute by public transport would take, see, for example, Figure 9.1. Based on their response, the distance class of the agent is classified as Short, Medium, or Long. For each distance class, one can pre-define sets of attribute levels for in-vehicle travel time, waiting time, and fare (ticket cost) as shown in Table 9.1. If an agent answers that they have a short commute distance, then they will be shown choice tasks with low in-vehicle travel times, low waiting times, and low fares.

Multiple choice question. Consider travelling to work by public transport. How long would this trip typically take?

- ◉ Short (less than 20 minutes in total)
- ○ Medium (about 20-60 minutes in total)
- ○ Long (more than 60 minutes in total)

Figure 9.1: Query about distance class in survey

Commuting distance	Level	Time	Wait	Fare
Short	0	4 min	2 min	$1
	1	8 min	5 min	$2
	2	12 min	8 min	$3
Medium	0	15 min	7 min	$2
	1	30 min	10 min	$4
	2	45 min	13 min	$6
Long	0	45 min	10 min	$3
	1	60 min	15 min	$6
	2	75 min	20 min	$9

Table 9.1: Predefined attribute levels for different distance classes

Script 9.1 shows an example of how such a design could be generated in Ngene, where the public transport alternatives `pt1` and `pt2` have attributes `TIME`, `WAIT`, `TRANSFERS`, and `FARE`. All attributes have been dummy coded, which is often useful when using noninformative priors, see Section 5.2. After design generation, all design-coded levels will need to be replaced with distance-specific attribute levels, except `TRANSFERS` since this attribute is not distance-specific and contains the actual number of transfers.

Table 9.2(a) presents the efficient design generated by Script 9.1. The relabelling of this design for the short distance class is shown in Table 9.2(b). For example, for short distances we relabel the in-vehicle travel time according to Table 9.1 as $0 \rightarrow 4$, $1 \rightarrow 8$, and $2 \rightarrow 12$.

```
design
;alts = (pt1, pt2)
;rows = 12
;eff = (mnl,d)
;model:     ? using design coding
U(pt1, pt2) = time.dummy[-]  * TIME[0,1,2]
            + wait.dummy[-]  * WAIT[0,1,2]
            + trans.dummy[-] * TRANSFERS[0,1]
            + cost.dummy[-]  * FARE[0,1,2]
$
```

Script 9.1: Single design for all distance classes using design coding

	Public transport 1				Public transport 2			
Choice task	Time	Wait	Transfers†	Fare	Time	Wait	Transfers†	Fare
1	0	2	1	2	2	0	0	1
2	1	2	0	0	2	0	1	2
3	0	2	0	1	2	1	1	0
4	1	0	0	2	0	2	1	0
5	1	0	1	0	2	1	0	1
6	2	1	0	2	0	0	1	1
7	1	1	1	1	0	2	0	0
8	2	0	0	0	1	1	1	1
9	2	1	0	1	1	0	1	2
10	0	1	1	0	1	2	0	2
11	2	2	1	1	0	1	0	2
12	0	0	1	2	1	2	0	0

† Contains actual levels.

(a) Original design using design coding

	Public transport 1				Public transport 2			
Choice task	Time	Wait	Transfers	Fare	Time	Wait	Transfers	Fare
1	4	8	1	3	12	2	0	2
2	8	8	0	1	12	2	1	3
3	4	8	0	2	12	5	1	1
4	8	2	0	3	4	8	1	1
5	8	2	1	1	12	5	0	2
6	12	5	0	3	4	2	1	2
7	8	5	1	2	4	8	0	1
8	12	2	0	1	8	5	1	2
9	12	5	0	2	8	2	1	3
10	4	5	1	1	8	8	0	3
11	12	8	1	2	4	5	0	3
12	4	2	1	3	8	8	0	1

(b) Relabelled design with actual levels for short distance class

Table 9.2: Relabelling attribute levels for specific distance classes

Choice task 1. Consider travelling to work by public transport. Which option do you prefer?	
Public transport A	Public transport B
4 minutes in-vehicle time 8 minutes waiting time 1 transfer $3.00	12 minutes in-vehicle time 2 minutes waiting time no transfer $2.00
○	○

(a) Short distance class

Choice task 1. Consider travelling to work by public transport. Which option do you prefer?	
Public transport A	Public transport B
45 minutes in-vehicle time 20 minutes waiting time 1 transfer $9.00	75 minutes in-vehicle time 10 minutes waiting time no transfer $6.00
○	○

(b) Long distance class

Figure 9.2: First choice task for different distance classes

The first choice task in this table is illustrated in Figure 9.2(a) for agents who reported a short distance to work, see Figure 9.1, while the first choice task for agents who reported a long distance to work is shown in Figure 9.2(b).

When parameter priors are available, a more efficient homogeneous design can be generated by explicitly considering different attribute levels during the design generation phase. Taking into account the same distance classes and attribute levels as shown in Table 9.1, a homogeneous design with informative priors could be generated using Script 9.2. In this script, three different models are specified, called `short`, `medium`, and `long` for each of the distance classes. Each model formulation in this script uses distance-specific levels for attributes `TIME`, `COST` and `FARE`. Parameter priors may differ across model specifications. It is important that the number of levels of each attribute is consistent across all model formulations.

To generate such a homogeneous design, the `fisher` property introduced in Chapter 8 can be used, which assumes joint estimation across all distance classes. In this case, on line 7 of Script 9.2, a homogeneous design called `des1` is generated jointly for all distance classes, with the assumption that 20% of the respondents have a short distance, 70% a medium distance, and 10% a long distance. Note that no constraints (neither conditional constraints nor check constraints, see Section 6) can be applied in combination with the `fisher` property.

The generated homogeneous design is shown in Table 9.3. Although it looks like there are three different designs, one for each distance class, they actually represent the same underlying homogeneous design. For example, the profiles for alternative “Public transport 1" in the first choice task are (8,5,0,2), (30,10,0,4), and (60,15,0,6) for `short`, `medium`, and `long`, respectively, which represent

```
design
;alts(short)  = (pt1, pt2)
;alts(medium) = (pt1, pt2)
;alts(long)   = (pt1, pt2)
;rows = 12
;eff = alldistances(mnl,d)
;fisher(alldistances) = des1(short[0.2], medium[0.7], long[0.1])
;model(short):
U(pt1, pt2) = time[-0.02]   * TIME[4,8,12]    ? in-vehicle travel time
            + wait[-0.03]   * WAIT[2,5,8]     ? waiting time
            + trans[-0.3]   * TRANSFERS[0,1]  ? number of transfers
            + cost[-0.2]    * FARE[1,2,3]     ? fare
;model(medium):
U(pt1, pt2) = time[-0.015]  * TIME[15,30,45]  ? in-vehicle travel time
            + wait[-0.02]   * WAIT[5,10,15]   ? waiting time
            + trans[-0.3]   * TRANSFERS[0,1]  ? number of transfers
            + cost[-0.15]   * FARE[2,4,6]     ? fare
;model(long):
U(pt1, pt2) = time[-0.01]   * TIME[45,60,75]  ? in-vehicle travel time
            + wait[-0.01]   * WAIT[10,15,20]  ? waiting time
            + trans[-0.25]  * TRANSFERS[0,1]  ? number of transfers
            + cost[-0.1]    * FARE[3,6,9]     ? fare
$
```

Script 9.2: Homogeneous design with informative priors

the middle levels for the time and fare attributes and the lowest level for the wait attribute. In other words, the three designs would be identical when converted to design coding.

9.2 Heterogeneous design

A heterogeneous design is generally more efficient than a homogeneous design, but will have less statistical power when making comparisons across distance classes. A heterogeneous design for the same public transport choice example can be generated by replacing the `fisher` property in Script 9.2 with the following syntax:

```
;fisher(alldistances) = des1(short[0.2]) + des2(medium[0.7]) + des3(long[0.1])
```

In this syntax, three entirely different designs are created, called `des1`, `des2`, and `des3`, again assuming joint estimation on data from the three distance classes. The resulting heterogeneous design is shown in Table 9.4. In contrast to the homogeneous design in Table 9.3, there is no similarity between the choice tasks for different distance classes. In the survey instrument, one would implement a separate choice experiment for each distance class, and based on the choice context of an agent, the agent is directed (branched) to the appropriate experiment.

9.3 Library of designs

Instead of simultaneously generating a design for all agents, one could create separate designs for different sets of attribute levels. We refer to this as a *library of designs*, see also Section 1.5.4. This is essentially a heterogeneous design in which each design is generated separately. The advantage of a library of designs over a heterogeneous design discussed in Section 9.2 is that it gives the analyst full flexibility to define attribute levels, specify utility functions, and apply constraints on attribute

Short	Public transport 1				Public transport 2			
Choice task	Time	Wait	Transfers	Fare	Time	Wait	Transfers	Fare
1	8	5	0	2	8	5	1	1
2	4	5	0	3	12	5	1	2
3	12	8	1	1	4	2	0	3
4	4	8	0	2	12	2	1	2
5	4	8	1	3	8	2	0	1
6	8	2	0	3	8	8	1	1
7	12	5	0	2	4	5	1	2
8	8	2	1	1	8	8	0	3
9	8	8	1	2	12	2	0	2
10	4	2	1	1	12	8	0	3
11	12	5	0	1	4	5	1	3
12	12	2	1	3	4	8	0	1

Medium	Public transport 1				Public transport 2			
Choice task	Time	Wait	Transfers	Fare	Time	Wait	Transfers	Fare
1	30	10	0	4	30	10	1	2
2	15	10	0	6	45	10	1	4
3	45	15	1	2	15	5	0	6
4	15	15	0	4	45	5	1	4
5	15	15	1	6	30	5	0	2
6	30	5	0	6	30	15	1	2
7	45	10	0	4	15	10	1	4
8	30	5	1	2	30	15	0	6
9	30	15	1	4	45	5	0	4
10	15	5	1	2	45	15	0	6
11	45	10	0	2	15	10	1	6
12	45	5	1	6	15	15	0	2

Long	Public transport 1				Public transport 2			
Choice task	Time	Wait	Transfers	Fare	Time	Wait	Transfers	Fare
1	60	15	0	6	60	15	1	3
2	45	15	0	9	75	15	1	6
3	75	20	1	3	45	10	0	9
4	45	20	0	6	75	10	1	6
5	45	20	1	9	60	10	0	3
6	60	10	0	9	60	20	1	3
7	75	15	0	6	45	15	1	6
8	60	10	1	3	60	20	0	9
9	60	20	1	6	75	10	0	6
10	45	10	1	3	75	20	0	9
11	75	15	0	3	45	15	1	9
12	75	10	1	9	45	20	0	3

Table 9.3: Homogeneous design for three distance classes

Short	Public transport 1				Public transport 2			
Choice task	Time	Wait	Transfers	Fare	Time	Wait	Transfers	Fare
1	12	5	1	1	4	5	0	2
2	4	8	0	1	8	2	1	3
3	12	2	0	2	4	8	1	3
4	4	8	1	3	8	2	0	1
5	4	2	0	3	12	8	1	2
6	8	5	1	3	12	5	0	1
7	8	5	0	2	12	5	1	1
8	12	2	1	3	4	8	0	1
9	12	2	0	1	8	8	1	3
10	8	5	0	1	4	5	1	3
11	8	8	1	2	12	2	0	2
12	4	8	1	2	8	2	0	2

Medium	Public transport 1				Public transport 2			
Choice task	Time	Wait	Transfers	Fare	Time	Wait	Transfers	Fare
1	45	5	1	4	15	15	0	4
2	30	15	1	2	15	5	0	6
3	15	15	1	6	45	5	0	2
4	30	15	0	4	30	5	1	4
5	15	10	1	4	45	10	0	4
6	30	15	0	2	30	10	1	6
7	45	10	0	6	30	10	1	2
8	30	10	0	2	30	5	1	6
9	15	5	0	6	45	15	1	2
10	45	5	1	2	15	15	0	6
11	15	5	1	4	45	15	0	4
12	45	10	0	6	15	10	1	2

Long	Public transport 1				Public transport 2			
Choice task	Time	Wait	Transfers	Fare	Time	Wait	Transfers	Fare
1	45	15	1	6	75	15	0	6
2	60	15	0	9	60	15	1	3
3	75	10	1	6	45	20	0	6
4	60	10	0	3	45	20	1	9
5	45	15	0	3	75	10	1	9
6	75	20	1	3	45	10	0	9
7	45	15	1	6	60	15	0	6
8	60	10	0	9	60	20	1	3
9	45	10	1	3	75	20	0	9
10	75	20	0	9	60	10	1	3
11	60	20	1	9	75	10	0	3
12	75	20	0	6	45	15	1	6

Table 9.4: Heterogeneous design for three distance classes

Open question. Consider your most recent trip to work by public transport. Please enter the details of this trip.

In-vehicle travel time	35	minutes
Waiting time	8	minutes
Number of transfers	1	
Fare	5.50	dollar

Figure 9.3: Query about recent trip characteristics in survey

level combinations (see Chapter 6). For these reasons, a library of designs is often the preferred approach.

Let us again consider the public transport choice experiment. Script 9.3 shows three scripts that need to be run separately, one for each distance class. The number of attribute levels can differ across classes; for example, attribute `TRANSFERS` in the script for long distance allows up to 2 transfers while in the other scripts it only allows at most 1 transfer. In addition, we added conditional constraints to allow only lower fares with shorter travel times and higher fares with longer travel times.

The generated library of designs is shown in Table 9.5. Depending on the commute distance reported by the agent earlier in the survey (for example, see Figure 9.1), the agent is shown the choice experiment based on the relevant design in the library.

9.4 Pivot designs

Instead of predefined attribute levels, one could create a *pivot design* containing relative or absolute pivots around the reference levels reported by the agent in the survey. Pivots are mainly useful for quantitative attributes. Based on reference levels and pivots, agent-specific attribute levels are calculated on the fly within the survey instrument.

In Ngene, pivot designs can be generated by first assigning reference levels for each attribute in the reference alternative using the suffix `.ref` and then specifying the other alternatives whereby attributes can have the suffix `.piv` to indicate pivots around the reference levels. A relative pivot is indicated with a percentage; for example, a pivot attribute level of `-25%` means that its level is 25 per cent less than the reference level, and `10%` means ten per cent more than the reference level. An absolute pivot is indicated with numbers; for example, a pivot attribute level of `-2` means that its level is two lower than the reference level, and `1` means one higher than the reference level.

Figure 9.3 shows an example of an open survey question where the agent is queried about a recent trip to obtain such reference levels. To avoid any typing mistakes by the agent in open text fields, one could instead use drop-down lists from which the agent can select.

Pivot designs are not without risk, and it is important to make sure that logic is implemented in the survey instrument to ensure that reported reference levels are valid and that on the fly calculated attribute levels are feasible. For example, suppose that the agent has entered a waiting time of 0. Then any relative pivots would again result in a zero waiting time (since x% on top of zero is still zero), which is undesirable. In addition, if the agent would have specified a low fare, or even 0 fare, then with absolute pivots the resulting fare may become negative.

```
design  ? Short distance
;alts = (pt1, pt2)
;rows = 12
;eff = (mnl,d)
;cond:
if(pt1.TIME=4, pt1.FARE<3), if(pt2.TIME=4, pt2.FARE<3),
if(pt1.TIME=12, pt1.FARE>1), if(pt2.TIME=12, pt2.FARE>1)
;model:
U(pt1, pt2) = time[-0.02]   * TIME[4,8,12]    ? in-vehicle travel time
            + wait[-0.03]   * WAIT[2,5,8]     ? waiting time
            + trans[-0.3]   * TRANSFERS[0,1]  ? number of transfers
            + cost[-0.2]    * FARE[1,2,3]     ? fare
$
```

```
design  ? Medium distance
;alts = (pt1, pt2)
;rows = 12
;eff = (mnl,d)
;cond:
if(pt1.TIME=15, pt1.FARE<6), if(pt2.TIME=15, pt2.FARE<6),
if(pt1.TIME=45, pt1.FARE>2), if(pt2.TIME=45, pt2.FARE>2)
;model:
U(pt1, pt2) = time[-0.015]  * TIME[15,30,45]  ? in-vehicle travel time
            + wait[-0.02]   * WAIT[5,10,15]   ? waiting time
            + trans[-0.3]   * TRANSFERS[0,1]  ? number of transfers
            + cost[-0.15]   * FARE[2,4,6]     ? fare
$
```

```
design  ? Long distance
;alts = (pt1, pt2)
;rows = 12
;eff = (mnl,d)
;cond:
if(pt1.TIME=45, pt1.FARE<9), if(pt2.TIME=45, pt2.FARE<9),
if(pt1.TIME=75, pt1.FARE>3), if(pt2.TIME=75, pt2.FARE>3)
;model:
U(pt1, pt2) = time[-0.01]   * TIME[45,60,75]    ? in-vehicle travel time
            + wait[-0.01]   * WAIT[10,15,20]    ? waiting time
            + trans[-0.25]  * TRANSFERS[0,1,2]  ? number of transfers
            + cost[-0.1]    * FARE[3,6,9]       ? fare
$
```

Script 9.3: Library of designs

Short	Public transport 1				Public transport 2			
Choice task	Time	Wait	Transfers	Fare	Time	Wait	Transfers	Fare
1	4	8	1	2	8	2	0	2
2	8	5	1	2	12	5	0	2
3	4	2	0	2	8	8	1	1
4	8	5	1	3	12	5	0	2
5	8	2	0	3	8	5	1	1
6	4	5	1	1	12	5	0	3
7	12	8	0	2	4	2	1	2
8	4	8	0	1	12	2	1	3
9	12	8	0	3	4	2	1	1
10	12	5	1	3	4	8	0	1
11	12	2	0	2	4	8	1	2
12	8	2	1	1	8	8	0	3

Medium	Public transport 1				Public transport 2			
Choice task	Time	Wait	Transfers	Fare	Time	Wait	Transfers	Fare
1	15	15	0	2	45	10	1	6
2	15	5	1	4	45	15	0	4
3	45	10	0	4	15	10	1	4
4	45	15	0	6	15	5	1	2
5	30	15	1	6	45	5	0	4
6	45	5	0	4	15	15	1	4
7	15	10	1	2	45	10	0	6
8	45	5	1	6	15	15	0	2
9	30	10	1	2	30	5	0	6
10	15	5	0	4	30	15	1	2
11	30	10	1	2	30	10	0	6
12	30	15	0	4	30	5	1	4

Long	Public transport 1				Public transport 2			
Choice task	Time	Wait	Transfers	Fare	Time	Wait	Transfers	Fare
1	45	15	2	6	75	15	0	6
2	75	20	1	9	45	10	2	3
3	75	15	1	9	45	20	1	3
4	75	10	2	6	45	15	0	6
5	60	20	0	9	60	10	1	3
6	60	10	0	6	60	20	2	3
7	45	20	0	3	60	15	2	6
8	60	20	1	3	60	10	1	9
9	45	15	1	3	75	10	1	9
10	60	15	2	3	60	15	0	9
11	45	10	2	6	75	20	0	6
12	75	10	0	6	45	20	2	6

Table 9.5: Library of designs for three distance classes

```
1  design  ? Public transport choice with reference alternative
2  ;alts = (ref, pt1, pt2)
3  ;rows = 12
4  ;eff = (mnl,d)
5  ;alg = mfedorov(stop = total(10000 iterations))
6  ;model:
7  U(ref) = time[-0.015] * TIME.ref[30]      ? in-vehicle travel time (min)
8         + wait[-0.02]  * WAIT.ref[10]      ? waiting time (min)
9         + trans[-0.3]  * TRANSFERS.ref[0]  ? number of transfers
10        + cost[-0.15]  * FARE.ref[4]       ? fare ($)
11        /
12 U(pt1, pt2) = time         * TIME.piv[-50%,0%,50%](3-5,3-5,3-5)
13             + wait         * WAIT.piv[-30%,0%,30%](3-5,3-5,3-5)
14             + trans        * TRANSFERS2[0,1](4-8,4-8)
15             + cost         * FARE.piv[-2,0,2](3-5,3-5,3-5)
16 $
```

Script 9.4: Pivot design around single set of reference levels

Pivot designs can be optimised around a single set of reference levels or can consider multiple sets of reference levels. These options are discussed in the following subsections.

9.4.1 Pivot designs optimised around a single set of reference levels

When only a single set of reference levels is used, these reference levels need to be as representative as possible for all agents so that the design is relatively efficient for each agent. An obvious choice would be to use reference levels based on average attributes values expected in the sample population.

Script 9.4 illustrates how to generate a pivot design for the public transport choice experiment where only a single set of reference levels is used. In lines 7—-10 of this script, the reference levels for `TIME`, `WAIT`, and `FARE` have been chosen as the midpoints of the attribute levels of the medium distance class, namely 30 minutes, 10 minutes, and $4, respectively. The reference value for `TRANSFER` was set to 0 as the expected most typical value. For illustration purposes, relative pivots were chosen for attributes `TIME` and `WAIT`, while absolute pivots are used for attribute `FARE`. Note that not all attributes need to be pivoted, as shown in this script for attribute `TRANSFER`. Since we use the modified Fedorov algorithm in this script, attribute level frequency constraints have been applied to all attributes. To calculate design efficiency, Ngene applies the pivots to the reference levels to obtain the actual attribute levels.

The generated pivot design is shown in Table 9.6, omitting the reference alternative. These attribute levels are expressed in pivots (except for the transfer attribute), and to emphasise this, we added a plus symbol (+) to positive pivots in this table. To apply the pivot design, consider the agent that reported the attribute levels of their recent trip shown in Figure 9.3. Table 9.7 shows the attribute levels for the two hypothetical public transport options that are generated on the fly within the survey instrument by applying the pivots in Table 9.6 to these reference levels. The first choice task in Table 9.7 is shown in Figure 9.4, where the levels of time attributes shown to the agent have been rounded to reduce cognitive burden.

Script 9.4 optimises the pivot design under the assumption that the reference alternative is shown in each choice task. Suppose that one would like to optimise the pivot design without showing the reference alternative. This can be achieved by specifying an auxiliary model; see Section 7.2. Script 9.5 generates a pivot design under the assumption that the reference alternative is *not* shown. Auxiliary model `withref` defines the reference levels, while the pivot design is optimised only for

	Public transport 1				Public transport 2			
Task	Time	Wait	Transfers†	Fare	Time	Wait	Transfers†	Fare
1	0%	-30%	1	-2	-50%	+30%	1	+2
2	+50%	+30%	1	-2	-50%	-30%	0	+2
3	+50%	+30%	0	-2	-50%	-30%	1	0
4	-50%	0%	1	-2	+50%	+30%	0	-2
5	0%	-30%	1	+2	+50%	+30%	1	-2
6	0%	-30%	1	+2	-50%	0%	1	-2
7	+50%	-30%	0	+2	0%	0%	1	-2
8	+50%	0%	1	-2	0%	0%	0	0
9	+50%	-30%	0	0	-50%	+30%	1	+2
10	-50%	0%	1	+2	0%	-30%	1	-2
11	-50%	+30%	0	0	+50%	-30%	1	0
12	-50%	+30%	1	0	+50%	-30%	0	+2

† Contains actual levels.

Table 9.6: Pivot design based on single set of reference levels

	Public transport 1				Public transport 2			
Task	Time	Wait	Transfers	Fare	Time	Wait	Transfers	Fare
1	35	5.6	1	3.5	17.5	10.4	1	7.5
2	52.5	10.4	1	3.5	17.5	5.6	0	7.5
3	52.5	10.4	0	3.5	17.5	5.6	1	5.5
4	17.5	8	1	3.5	52.5	10.4	0	3.5
5	35	5.6	1	7.5	52.5	10.4	1	3.5
6	35	5.6	1	7.5	17.5	8	1	3.5
7	52.5	5.6	0	7.5	35	8	1	3.5
8	52.5	8	1	3.5	35	8	0	5.5
9	52.5	5.6	0	5.5	17.5	10.4	1	7.5
10	17.5	8	1	7.5	35	5.6	1	3.5
11	17.5	10.4	0	5.5	52.5	5.6	1	5.5
12	17.5	10.4	1	5.5	52.5	5.6	0	7.5

Table 9.7: Attribute levels after applying pivots to reference levels in Figure 9.3

First choice task. Consider again your recent trip to work by public transport. For your next trip to work, which of the following options do you prefer?

Recent public transport	Public transport 1	Public transport 2
35 minutes in-vehicle time 8 minutes waiting time 1 transfer $5.50	35 minutes in-vehicle time 6 minutes waiting time 1 transfer $3.50	18 minutes in-vehicle time 10 minutes waiting time 1 transfer $7.50
○	○	○

Figure 9.4: Choice task with attribute levels pivoted around reported reference levels

```
1  design  ? Public transport choice without reference alternative
2  ;alts(withref)    = ref, pt1, pt2
3  ;alts(withoutref) = (pt1, pt2)
4  ;rows = 12
5  ;eff = withoutref(mnl,d)
6  ;model(withref): ? axuliary model with reference alternative
7  U(ref) = time[-0.015] * TIME.ref[30]      ? in-vehicle travel time (min)
8         + wait[-0.02]  * WAIT.ref[10]      ? waiting time (min)
9         + trans[-0.3]  * TRANSFERS.ref[0]  ? number of transfers
10        + cost[-0.15]  * FARE.ref[4]       ? fare ($)
11        /
12 U(pt1, pt2) = time         * TIME.piv[-50%,0%,50%](3-5,3-5,3-5)
13             + wait         * WAIT.piv[-30%,0%,30%](3-5,3-5,3-5)
14             + trans        * TRANSFERS2[0,1](4-8,4-8)
15             + cost         * FARE.piv[-2,0,2](3-5,3-5,3-5)
16 ;model(withoutref): ? model of interest without reference alternative
17 U(pt1, pt2) = time         * TIME.piv[-50%,0%,50%](3-5,3-5,3-5)
18             + wait         * WAIT.piv[-30%,0%,30%](3-5,3-5,3-5)
19             + trans        * TRANSFERS2[0,1](4-8,4-8)
20             + cost         * FARE.piv[-2,0,2](3-5,3-5,3-5)
21 $
```

Script 9.5: Pivot design without showing reference alternative

the model without the reference alternative `withoutref`, as indicated in line 5. We omitted the parentheses around the alternatives on line 2 to avoid unnecessary dominance checks with respect to the reference alternative.

9.4.2 Pivot designs optimised around multiple sets of reference levels

Instead of optimising a design pivot design only around a single set of (average/typical) attribute levels, one could optimise the design across multiple sets of reference levels. This is achieved using the `fisher` property that was introduced in Chapter 8.

Script 9.6 illustrates how to generate a homogeneous pivot design that is optimised across the three distance classes as specified in models `short`, `medium`, and `long`. Each class has the same pivot levels, but differs with respect to the reference levels. For example, model `short` has a reference level for `TIME` of 8 minutes, while models `medium` and `long` have reference levels of 30 and 60 minutes, respectively. The fact that this is a homogeneous design can be seen from line 7 in the `fisher` property, which indicates that only a single design called `des1` is being generated, simultaneously optimised for the three distance classes with provided weights. Table 9.8 shows the generated homogeneous pivot design, which can be used for all distance classes.

In Script 9.6 we can replace line 7 with the syntax below to generate a heterogeneous pivot design.

```
;fisher(alldistances) = des1(short[0.2]) + des2(medium[0.7]) + des3(long[0.1])
```

This will create three designs, called `des1`, `des2`, and `des3` for each of the three distance classes. The generated heterogeneous design is shown in Table 9.9, where agents of different distance classes are shown different sets of choice tasks.

```
design  ? Public transport choice example
;alts(short)  = (ref, pt1, pt2)
;alts(medium) = (ref, pt1, pt2)
;alts(long)   = (ref, pt1, pt2)
;rows = 10
;eff = alldistances(mnl,d)
;fisher(alldistances) = des1(short[0.2],medium[0.7],long[0.1])
;model(short): ? short distance
U(ref) = time[-0.015] * TIME.ref[8]       ? in-vehicle travel time (min)
       + wait[-0.02]  * WAIT.ref[5]       ? waiting time (min)
       + trans[-0.3]  * TRANSFERS.ref[0]  ? number of transfers
       + cost[-0.15]  * FARE.ref[2]       ? fare ($)
       /
U(pt1, pt2) = time         * TIME.piv[-30%,-10%,10%,30%,50%]
            + wait         * WAIT.piv[-50%,-25%,0%,25%,50%]
            + trans        * TRANSFERS2[0,1]
            + cost         * FARE.piv[-2,-1,0,1,2]
;model(medium): ? medium distance
U(ref) = time[-0.015] * TIME.ref[30]
       + wait[-0.02]  * WAIT.ref[10]
       + trans[-0.3]  * TRANSFERS.ref[0]
       + cost[-0.15]  * FARE.ref[4]
       /
U(pt1, pt2) = time         * TIME.piv[-30%,-10%,10%,30%,50%]
            + wait         * WAIT.piv[-50%,-25%,0%,25%,50%]
            + trans        * TRANSFERS2[0,1]
            + cost         * FARE.piv[-2,-1,0,1,2]
;model(long): ? long distance
U(ref) = time[-0.015] * TIME.ref[60]
       + wait[-0.02]  * WAIT.ref[15]
       + trans[-0.3]  * TRANSFERS.ref[1]
       + cost[-0.15]  * FARE.ref[6]
       /
U(pt1, pt2) = time         * TIME.piv[-30%,-10%,10%,30%,50%]
            + wait         * WAIT.piv[-50%,-25%,0%,25%,50%]
            + trans        * TRANSFERS2[0,1]
            + cost         * FARE.piv[-2,-1,0,1,2]
$
```

Script 9.6: Homogeneous pivot design around multiple sets of reference levels

	Public transport 1				Public transport 2			
Task	Time	Wait	Transfers†	Fare	Time	Wait	Transfers†	Fare
1	-10%	+25%	1	+2	+10%	-50%	0	-2
2	+50%	0%	0	-1	-30%	0%	1	+1
3	+10%	+50%	1	-2	+30%	-50%	0	+2
4	-10%	+25%	0	0	+30%	-25%	1	0
5	+10%	+50%	0	-1	+10%	-25%	1	+1
6	+30%	-50%	1	-2	-10%	+50%	0	+2
7	-30%	-50%	0	+1	+50%	+50%	1	-1
8	-30%	0%	1	0	+50%	+25%	0	-1
9	+30%	-25%	1	+1	-10%	+25%	0	0
10	+50%	-25%	0	+2	-30%	0%	1	-2

† Contains actual levels.

Table 9.8: Homogeneous pivot design based on multiple sets of reference levels

Short	Public transport 1				Public transport 2			
Choice task	Time	Wait	Transfers†	Fare	Time	Wait	Transfers†	Fare
1	-10%	0%	0	+1	+30%	0%	1	-1
2	+50%	-25%	1	0	-30%	+25%	0	0
3	-10%	+25%	0	+2	+10%	-50%	1	-2
4	+30%	-50%	0	+1	+10%	+50%	1	-1
5	+10%	-50%	1	-2	-10%	+50%	0	+2
6	+50%	-25%	0	+2	-30%	0%	1	-2
7	-30%	+25%	1	-2	+50%	-25%	0	+2
8	+10%	+50%	1	-1	+30%	-25%	0	+1
9	+30%	0%	1	-1	-10%	+25%	0	+1
10	-30%	+50%	0	0	+50%	-50%	1	0

† Contains actual levels.

Medium	Public transport 1				Public transport 2			
Choice task	Time	Wait	Transfers†	Fare	Time	Wait	Transfers†	Fare
1	+50%	-25%	0	0	-10%	+25%	1	+1
2	-10%	0%	1	+1	+30%	0%	0	-1
3	+10%	-50%	0	-2	-10%	+50%	1	+2
4	+50%	+50%	0	-1	-30%	-50%	1	0
5	+10%	-50%	1	+1	+10%	+50%	0	-1
6	-10%	+25%	0	+2	+30%	0%	1	-2
7	-30%	+25%	0	0	+50%	-25%	1	0
8	+30%	+50%	1	-1	+10%	-50%	0	+1
9	-30%	0%	1	-2	+50%	-25%	0	+2
10	+30%	-25%	1	+2	-30%	+25%	0	-2

† Contains actual levels.

Long	Public transport 1				Public transport 2			
Choice task	Time	Wait	Transfers†	Fare	Time	Wait	Transfers†	Fare
1	-30%	-50%	1	+1	+50%	+50%	0	-1
2	+50%	0%	1	-1	-30%	0%	0	+1
3	+10%	+25%	0	0	+10%	-25%	1	0
4	-30%	0%	1	+2	+50%	0%	0	-2
5	+10%	+25%	0	-2	+10%	-25%	1	+2
6	+30%	-25%	0	0	-10%	+25%	1	0
7	-10%	-50%	0	+1	+30%	+50%	1	-1
8	-10%	+50%	1	+2	+30%	-50%	0	-2
9	+30%	+50%	1	-1	-30%	-50%	0	+1
10	+50%	-25%	0	-2	-10%	+25%	1	+2

† Contains actual levels.

Table 9.9: Heterogeneous pivot design based on multiple sets of reference levels

References

Adamowicz, V., Boxall, P., 2001. Future Directions of Stated Choice Methods for Environment Valuation.

Adamowicz, V., Dupont, D., Krupnick, A., 2006. Willingness to Pay to Reduce Community Health Risks from Municipal Drinking Water: A Stated Preference Study. doi:10.1037/e516992012-005.

Arentze, T., Borgers, A., Timmermans, H., DelMistro, R., 2003. Transport stated choice responses: effects of task complexity, presentation format and literacy. Transportation Research Part E: Logistics and Transportation Review 39, 229–244. doi:10.1016/S1366-5545(02)00047-9.

Bateman, I.J., Carson, R.T., Day, B., Hanemann, M., Hanley, N., Hett, T., Jones-Lee, M., Loomes, G., 2002. Economic Valuation with Stated Preference Techniques: A Manual, in: Economic Valuation with Stated Preference Techniques. Edward Elgar Publishing.

Batley, R., Bates, J., Bliemer, M., Börjesson, M., Bourdon, J., Cabral, M.O., Chintakayala, P.K., Choudhury, C., Daly, A., Dekker, T., Drivyla, E., Fowkes, T., Hess, S., Heywood, C., Johnson, D., Laird, J., Mackie, P., Parkin, J., Sanders, S., Sheldon, R., Wardman, M., Worsley, T., 2019. New appraisal values of travel time saving and reliability in Great Britain. Transportation 46, 583–621. doi:10.1007/s11116-017-9798-7.

Bech, M., Kjaer, T., Lauridsen, J., 2011. Does the number of choice sets matter? Results from a web survey applying a discrete choice experiment. Health Economics 20, 273–286. doi:10.1002/hec.1587.

Bennett, J., Blamey, R., 2001. The Choice Modelling Approach to Environmental Valuation. Edward Elgar Publishing.

Bierlaire, M., 2020. A short introduction to PandasBiogeme. Technical Report Technical report TRANSP-OR 200605. Transport and Mobility Laboratory, ENAC, EPFL.

Black, I.R., Efron, A., Ioannou, C., Rose, J.M., 2005. Designing and Implementing Internet Questionnaires Using Microsoft Excel. Australasian Marketing Journal (AMJ) 13, 61–72. doi:10.1016/S1441-3582(05)70078-1.

Bliemer, M., Rose, J., Beck, M., 2018. Generating partial choice set designs for stated choice experiments, Santa Barbara CA, USA.

Bliemer, M.C.J., Collins, A.T., 2016. On determining priors for the generation of efficient stated choice experimental designs. Journal of Choice Modelling 21, 10–14. doi:10.1016/j.jocm.2016.03.001.

Bliemer, M.C.J., Rose, J.M., 2010. Construction of experimental designs for mixed logit models allowing for correlation across choice observations. Transportation Research Part B: Methodological 44, 720–734. doi:10.1016/j.trb.2009.12.004.

Bliemer, M.C.J., Rose, J.M., 2011. Experimental design influences on stated choice outputs: An empirical study in air travel choice. Transportation Research Part A: Policy and Practice 45, 63–79. doi:10.1016/j.tra.2010.09.003.

Bliemer, M.C.J., Rose, J.M., Chorus, C.G., 2017. Detecting dominance in stated choice data and accounting for dominance-based scale differences in logit models. Transportation Research Part B: Methodological 102, 83–104. doi:10.1016/j.trb.2017.05.005.

Bliemer, M.C.J., Rose, J.M., Hensher, D.A., 2009. Efficient stated choice experiments for estimating nested logit models. Transportation Research Part B: Methodological 43, 19–35. doi:10.1016/j.trb.2008.05.008.

Bliemer, M.C.J., Rose, J.M., Hess, S., 2008. Approximation of bayesian efficiency in experimental choice designs. Journal of Choice Modelling 1, 98–126. doi:10.1016/S1755-5345(13)70024-1.

Boxall, P., Adamowicz, W.L.V., Moon, A., 2009. Complexity in choice experiments: choice of the status quo alternative and implications for welfare measurement*. Australian Journal of Agricultural and Resource Economics 53, 503–519. doi:10.1111/j.1467-8489.2009.00469.x.

Boyle, K.J., Özdemir, S., 2009. Convergent Validity of Attribute-Based, Choice Questions in Stated-Preference Studies. Environmental and Resource Economics 42, 247–264. doi:10.1007/s10640-008-9233-9.

Brazell, J., Louviere, J., 1996. Length Effects in Conjoint Choice Experiments and Surveys: An Explanation Based on Cumulative Cognitive Burden, Univeristy of Sydney.

Brazell, J.D., Diener, C.G., Karniouchina, E., Moore, W.L., Séverin, V., Uldry, P.F., 2006. The no-choice option and dual response choice designs. Marketing Letters 17, 255–268. doi:10.1007/s11002-006-7943-8.

Burgess, L., Street, D.J., 2003. Optimal Designs for 2 k Choice Experiments. Communications in Statistics - Theory and Methods 32, 2185–2206. doi:10.1081/STA-120024475.

Burke, P.F., Eckert, C., Sethi, S., 2020. A Multiattribute Benefits-Based Choice Model with Multiple Mediators: New Insights for Positioning. Journal of Marketing Research 57, 35–54. doi:10.1177/0022243719881618.

Campbell, D., Boeri, M., Doherty, E., George Hutchinson, W., 2015. Learning, fatigue and preference formation in discrete choice experiments. Journal of Economic Behavior & Organization 119, 345–363. doi:10.1016/j.jebo.2015.08.018.

Caussade, S., Ortúzar, J.d.D., Rizzi, L.I., Hensher, D.A., 2005. Assessing the influence of design dimensions on stated choice experiment estimates. Transportation Research Part B: Methodological 39, 621–640. doi:10.1016/j.trb.2004.07.006.

Chorus, C., 2012. Random Regret Minimization: An Overview of Model Properties and Empirical Evidence. Transport Reviews 32, 75–92. doi:10.1080/01441647.2011.609947.

Chrzan, K., 2010. Using Partial Profile Choice Experiments to Handle Large Numbers of Attributes. International Journal of Market Research 52, 827–840. doi:10.2501/S1470785310201673.

Collins, A.T., Bliemer, M.C.J., Rose, J., 2014. Constrained stated choice experimental designs. URL: https://opus.lib.uts.edu.au/handle/10453/119855.

Cook, R.D., Nachtsheim, C.J., 1980. A Comparison of Algorithms for Constructing Exact D-Optimal Designs. Technometrics 22, 315–324. doi:10.1080/00401706.1980.10486162.

Cooper, B., Rose, J., Crase, L., 2012. Does anybody like water restrictions? Some observations in Australian urban communities. Australian Journal of Agricultural and Resource Economics 56, 61–81. doi:10.1111/j.1467-8489.2011.00573.x.

Czajkowski, M., Giergiczny, M., Greene, W.H., 2014. Learning and Fatigue Effects Revisited: Investigating the Effects of Accounting for Unobservable Preference and Scale Heterogeneity. Land Economics 90, 324–351. doi:10.3368/le.90.2.324.

Daly, A., Dekker, T., Hess, S., 2016. Dummy coding vs effects coding for categorical variables: Clarifications and extensions. Journal of Choice Modelling 21, 36–41. doi:10.1016/j.jocm.2016.09.005.

De Bekker-Grob, E.W., Bliemer, M.C.J., Donkers, B., Essink-Bot, M.L., Korfage, I.J., Roobol, M.J., Bangma, C.H., Steyerberg, E.W., 2013. Patients' and urologists' preferences for prostate cancer treatment: a discrete choice experiment. British Journal of Cancer 109, 633–640. doi:10.1038/bjc.2013.370.

De Bekker-Grob, E.W., Donkers, B., Jonker, M.F., Stolk, E.A., 2015. Sample Size Requirements for Discrete-Choice Experiments in Healthcare: a Practical Guide. The Patient 8, 373–384. doi:10.1007/s40271-015-0118-z.

DeShazo, J.R., Fermo, G., 2002. Designing Choice Sets for Stated Preference Methods: The Effects of Complexity on Choice Consistency. Journal of Environmental Economics and Management 44, 123–143. doi:10.1006/jeem.2001.1199.

Determann, D., Korfage, I.J., Lambooij, M.S., Bliemer, M., Richardus, J.H., Steyerberg, E.W., Bekker-Grob, E.W.d., 2014. Acceptance of Vaccinations in Pandemic Outbreaks: A Discrete Choice Experiment. PLOS ONE 9, e102505. doi:10.1371/journal.pone.0102505.

Dhar, R., 1997. Consumer Preference for a No-Choice Option. Journal of Consumer Research 24, 215–231. doi:10.1086/209506.

Dhar, R., Simonson, I., 2003. The Effect of Forced Choice on Choice. Journal of Marketing Research 40, 146–160. doi:10.1509/jmkr.40.2.146.19229.

Eagle, T.C., 1984. Parameter stability in disaggregate retail choice models: empirical evidence. Journal of Retailing 60, 101–123.

Farrar, S., Ryan, M., 1999. Response-ordering effects: a methodological issue in conjoint analysis. Health Economics 8, 75–79. doi:10.1002/(SICI)1099-1050(199902)8:1<75::AID-HEC400>3.0.CO;2-5.

Fayyaz, M., Bliemer, M.C.J., Beck, M.J., Hess, S., van Lint, J.W.C., 2021. Stated choices and simulated experiences: Differences in the value of travel time and reliability. Transportation Research Part C: Emerging Technologies 128, 103145. doi:10.1016/j.trc.2021.103145.

Green, P.E., Srinivasan, V., 1990. Conjoint Analysis in Marketing: New Developments with Implications for Research and Practice. Journal of Marketing 54, 3–19. doi:10.1177/002224299005400402.

Greiner, R., Bliemer, M., Ballweg, J., 2014. Design considerations of a choice experiment to estimate likely participation by north Australian pastoralists in contractual biodiversity conservation. Journal of Choice Modelling 10, 34–45. doi:10.1016/j.jocm.2014.01.002.

Haghani, M., Bliemer, M.C.J., Rose, J.M., Oppewal, H., Lancsar, E., 2021a. Hypothetical bias in stated choice experiments: Part I. Macro-scale analysis of literature and integrative synthesis of empirical evidence from applied economics, experimental psychology and neuroimaging. Journal of Choice Modelling 41, 100309. doi:10.1016/j.jocm.2021.100309.

Haghani, M., Bliemer, M.C.J., Rose, J.M., Oppewal, H., Lancsar, E., 2021b. Hypothetical bias in stated choice experiments: Part II. Conceptualisation of external validity, sources and explanations of bias and effectiveness of mitigation methods. Journal of Choice Modelling 41, 100322. doi:10.1016/j.jocm.2021.100322.

Hahn, G., Shapiro, S., 1967. Statistical Models in Engineering. Wiley.

Hansen, T.B., Lindholt, J.S., Diederichsen, A.C.P., Bliemer, M.C.J., Lambrechtsen, J., Steffensen, F.H., Søgaard, R., 2019. Individual preferences on the balancing of good and harm of cardiovascular disease screening. Heart (British Cardiac Society) 105, 761–767. doi:10.1136/heartjnl-2018-314103.

He, Y., Oppewal, H., 2018. See How Much We've Sold Already! Effects of Displaying Sales and Stock Level Information on Consumers' Online Product Choices. Journal of Retailing 94, 45–57. doi:10.1016/j.jretai.2017.10.002.

Hedayat, A.S., Sloane, N.J.A., Stufken, J., 1999. Orthogonal Arrays. Springer Series in Statistics, Springer, New York, NY. doi:10.1007/978-1-4612-1478-6.

Hensher, D.A., 2004. Accounting for stated choice design dimensionality in willingness to pay for travel time savings. Journal of Transport Economics and Policy 38, 425–446.

Hensher, D.A., 2006. How do respondents process stated choice experiments? Attribute consideration under varying information load. Journal of Applied Econometrics 21, 861–878. doi:10.1002/jae.877.

Hensher, D.A., 2010. Hypothetical bias, choice experiments and willingness to pay. Transportation Research Part B: Methodological 44, 735–752. doi:10.1016/j.trb.2009.12.012.

Hensher, D.A., Rose, J.M., Greene, W.H., 2015. Applied Choice Analysis. Second ed., Cambridge University Press. doi:10.1017/CBO9781316136232.

Hensher, D.A., Stopher, P.R., Louviere, J.J., 2001. An exploratory analysis of the effect of numbers of choice sets in designed choice experiments: an airline choice application. Journal of Air Transport Management 7, 373–379. doi:10.1016/S0969-6997(01)00031-X.

Hess, S., Choudhury, C.F., Bliemer, M.C.J., Hibberd, D., 2020. Modelling lane changing behaviour in approaches to roadworks: Contrasting and combining driving simulator data with stated choice data. Transportation Research Part C: Emerging Technologies 112, 282–294. doi:10.1016/j.trc.2019.12.003.

Hess, S., Palma, D., 2019. *Apollo*: A flexible, powerful and customisable freeware package for choice model estimation and application. Journal of Choice Modelling 32, 100170. doi:10.1016/j.jocm.2019.100170.

Huber, J., Zwerina, K., 1996. The Importance of Utility Balance in Efficient Choice Designs. Journal of Marketing Research 33, 307–317. doi:10.1177/002224379603300305.

Johnston, R.J., Boyle, K.J., Adamowicz, W.V., Bennett, J., Brouwer, R., Cameron, T.A., Hanemann, W.M., Hanley, N., Ryan, M., Scarpa, R., Tourangeau, R., Vossler, C.A., 2017. Contemporary Guidance for Stated Preference Studies. Journal of the Association of Environmental and Resource Economists 4, 319–405. doi:10.1086/691697.

Kessels, R., 2016. Homogeneous versus heterogeneous designs for stated choice experiments: Ain't homogeneous designs all bad? Journal of Choice Modelling 21, 2–9. doi:10.1016/j.jocm.2016.08.001.

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

Kessels, R., Jones, B., Goos, P., 2011. Bayesian optimal designs for discrete choice experiments with partial profiles. Journal of Choice Modelling 4, 52–74. doi:10.1016/S1755-5345(13)70042-3.

Kjær, T., Bech, M., Gyrd-Hansen, D., Hart-Hansen, K., 2006. Ordering effect and price sensitivity in discrete choice experiments: need we worry? Health Economics 15, 1217–1228. doi:10.1002/hec.1117.

Kontoleon, A., Yabe, M., 2003. Assessing the impacts of alternative 'Opt-out' formats in choice experiment studies: Consumer preferences for genetically modified content and production information in food. Journal of Agriculture Policy and Research 5, 1–43.

Liebe, U., Mariel, P., Beyer, H., Meyerhoff, J., 2021. Uncovering the Nexus Between Attitudes, Preferences, and Behavior in Sociological Applications of Stated Choice Experiments. Sociological Methods & Research 50, 310–347. doi:10.1177/0049124118782536.

Logar, I., Brouwer, R., Campbell, D., 2020. Does attribute order influence attribute-information processing in discrete choice experiments? Resource and Energy Economics 60, 101164. doi:10.1016/j.reseneeco.2020.101164.

MacCrimmon, K.R., Toda, M., 1969. The Experimental Determination of Indifference Curves. The Review of Economic Studies 36, 433–451. doi:10.2307/2296469.

MacDonald, D.H., Morrison, M.D., Rose, J.M., Boyle, K.J., 2011. Valuing a multistate river: the case of the River Murray*. Australian Journal of Agricultural and Resource Economics 55, 374–392. doi:10.1111/j.1467-8489.2011.00551.x.

Mariel, P., Hoyos, D., Meyerhoff, J., Czajkowski, M., Dekker, T., Glenk, K., Jacobsen, J.B., Liebe, U., Olsen, S.B., Sagebiel, J., Thiene, M., 2021. Environmental Valuation with Discrete Choice Experiments: Guidance on Design, Implementation and Data Analysis. SpringerBriefs in Economics, Springer International Publishing, Cham. doi:10.1007/978-3-030-62669-3.

May, K.O., 1954. Intransitivity, Utility, and the Aggregation of Preference Patterns. Econometrica 22, 1–13. doi:10.2307/1909827.

McFadden, D., 1973. Conditional Logit Analysis of Qualitative Choice Behavior, in: Frontiers in Econometrics. Academic Press, New York, NY, pp. 105–142.

Meißner, M., Oppewal, H., Huber, J., 2020. Surprising adaptivity to set size changes in multi-attribute repeated choice tasks. Journal of Business Research 111, 163–175. doi:10.1016/j.jbusres.2019.01.008.

Merkert, R., Bliemer, M.C.J., Fayyaz, M., 2022. Consumer preferences for innovative and traditional last-mile parcel delivery. International Journal of Physical Distribution & Logistics Management 52, 261–284. doi:10.1108/IJPDLM-01-2021-0013.

Meyer, R.J., Eagle, T.C., 1982. Context-Induced Parameter Instability in a Disaggregate-Stochastic Model of Store Choice. Journal of Marketing Research 19, 62–71. doi:10.2307/3151531.

Meyer, R.K., Nachtsheim, C.J., 1995. The Coordinate-Exchange Algorithm for Constructing Exact Optimal Experimental Designs. Technometrics 37, 60–69. doi:10.2307/1269153.

Meyerhoff, J., Oehlmann, M., Weller, P., 2015. The Influence of Design Dimensions on Stated Choices in an Environmental Context. Environmental and Resource Economics 61, 385–407. doi:10.1007/s10640-014-9797-5.

Mosteller, F., Nogee, P., 1951. An Experimental Measurement of Utility. Journal of Political Economy 59, 371–404.

Oehlmann, M., Meyerhoff, J., Mariel, P., Weller, P., 2017. Uncovering context-induced status quo effects in choice experiments. Journal of Environmental Economics and Management 81, 59–73. doi:10.1016/j.jeem.2016.09.002.

Ohler, T., Le, A., Louviere, J., Swait, J., 2000. Attribute Range Effects in Binary Response Tasks. Marketing Letters 11, 249–260. doi:10.1023/A:1008139226934.

Orme, B.K., 2019. Getting Started with Conjoint Analysis: Strategies for Product Design and Pricing Research. Fourth ed., Research Publishers LLC.

Ortúzar, J.d.D., Bascuñán, R., Rizzi, L.I., Salata, A., 2021. Assessing the potential acceptability of road pricing in Santiago. Transportation Research Part A: Policy and Practice 144, 153–169. doi:10.1016/j.tra.2020.12.007.

Penn, J.M., Hu, W., 2018. Understanding Hypothetical Bias: An Enhanced Meta-Analysis. American Journal of Agricultural Economics 100, 1186–1206. doi:10.1093/ajae/aay021.

Rolfe, J., Bennett, J., 2009. The impact of offering two versus three alternatives in choice modelling experiments. Ecological Economics 68, 1140–1148. doi:10.1016/j.ecolecon.2008.08.007.

Rose, J.M., Bliemer, M.C.J., 2013. Sample size requirements for stated choice experiments. Transportation 40, 1021–1041. doi:10.1007/s11116-013-9451-z.

Rose, J.M., Bliemer, M.C.J., Hensher, D.A., Collins, A.T., 2008. Designing efficient stated choice experiments in the presence of reference alternatives. Transportation Research Part B: Methodological 42, 395–406. doi:10.1016/j.trb.2007.09.002.

Rose, J.M., Hensher, D.A., Caussade, S., Ortúzar, J.d.D., Jou, R.C., 2009. Identifying differences in willingness to pay due to dimensionality in stated choice experiments: a cross country analysis. Journal of Transport Geography 17, 21–29. doi:10.1016/j.jtrangeo.2008.05.001.

Rose, J.M., Hess, S., 2009. Dual-Response Choices in Pivoted Stated Choice Experiments. Transportation Research Record 2135, 25–33. doi:10.3141/2135-04.

Rousseas, S.W., Hart, A.G., 1951. Experimental Verification of a Composite Indifference Map. Journal of Political Economy 59, 288–318. doi:10.1086/257092.

Ryan, M., Krucien, N., Hermens, F., 2018. The eyes have it: Using eye tracking to inform information processing strategies in multi-attributes choices. Health Economics 27, 709–721. doi:10.1002/hec.3626.

Scarpa, R., Drucker, A.G., Anderson, S., Ferraes-Ehuan, N., Gómez, V., Risopatrón, C.R., Rubio-Leonel, O., 2003. Valuing genetic resources in peasant economies: the case of 'hairless' creole pigs in Yucatan. Ecological Economics 45, 427–443. doi:10.1016/S0921-8009(03)00095-8.

Scarpa, R., Rose, J.M., 2008. Design efficiency for non-market valuation with choice modelling: how to measure it, what to report and why*. Australian Journal of Agricultural and Resource Economics 52, 253–282. doi:10.1111/j.1467-8489.2007.00436.x.

Scott, A., Vick, S., 1999. Patients, Doctors and Contracts: An Application of Principal-Agent Theory to the Doctor-Patient Relationship. Scottish Journal of Political Economy 46, 111–134. doi:10.1111/1467-9485.00124.

Street, D.J., Bunch, D.S., Moore, B.J., 2001. Optimal designs for 2 k paired comparison experiments. Communications in Statistics - Theory and Methods 30, 2149–2171. doi:10.1081/STA-100106068.

Street, D.J., Burgess, L., 2004. Optimal and near-optimal pairs for the estimation of effects in 2-level choice experiments. Journal of Statistical Planning and Inference 118, 185–199. doi:10.1016/S0378-3758(02)00399-3.

Street, D.J., Burgess, L., Louviere, J.J., 2005. Quick and easy choice sets: Constructing optimal and nearly optimal stated choice experiments. International Journal of Research in Marketing 22, 459–470. doi:10.1016/j.ijresmar.2005.09.003.

Sándor, Z., Wedel, M., 2001. Designing Conjoint Choice Experiments Using Managers' Prior Beliefs. Journal of Marketing Research 38, 430–444. doi:10.1509/jmkr.38.4.430.18904.

Sándor, Z., Wedel, M., 2002. Profile Construction in Experimental Choice Designs for Mixed Logit Models. Marketing Science 21, 455–475. doi:10.1287/mksc.21.4.455.131.

Sándor, Z., Wedel, M., 2005. Heterogeneous Conjoint Choice Designs. Journal of Marketing Research 42, 210–218. doi:10.1509/jmkr.42.2.210.62285.

Thurstone, L.L., 1931. The Indifference Function. The Journal of Social Psychology 2, 139–167. doi:10.1080/00224545.1931.9918964.

Train, K.E., 2009. Discrete Choice Methods with Simulation. Second ed., Cambridge University Press, Cambridge. doi:10.1017/CBO9780511805271.

Van Cranenburgh, S., Collins, A.T., 2019. New software tools for creating stated choice experimental designs efficient for regret minimisation and utility maximisation decision rules. Journal of Choice Modelling 31, 104–123. doi:10.1016/j.jocm.2019.04.002.

Van der Waerden, P., Borgers, A., Timmermans, H., 2004. The Effects of Attribute Level Definition on Stated Choice Behavior, Costa Rica.

Walker, J.L., Wang, Y., Thorhauge, M., Ben-Akiva, M., 2018. D-efficient or deficient? A robustness analysis of stated choice experimental designs. Theory and Decision 84, 215–238. doi:10.1007/s11238-017-9647-3.

Weller, P., Oehlmann, M., Mariel, P., Meyerhoff, J., 2014. Stated and inferred attribute non-attendance in a design of designs approach. Journal of Choice Modelling 11, 43–56. doi:10.1016/j.jocm.2014.04.002.

Weng, W., Morrison, M.D., Boyle, K.J., Boxall, P.C., Rose, J., 2021. Effects of the number of alternatives in public good discrete choice experiments. Ecological Economics 182, 106904. doi:10.1016/j.ecolecon.2020.106904.

Wittink, D., Huber, J., Zandan, P., Johnson, R., 1992. The Number of Levels Effect in Conjoint: Where Does It Come From and Can It Be Eliminated? In: Sawtooth Software Research Paper Series.

Wittink, D.R., Krishnamurthi, L., Reibstein, D.J., 1990. The Effect of Differences in the Number of Attribute Levels on Conjoint Results. Marketing Letters 1, 113–123. doi:10.1007/BF00435295.

Wu, F., Swait, J., Chen, Y., 2019. Feature-based attributes and the roles of consumers' perception bias and inference in choice. International Journal of Research in Marketing 36, 325–340. doi:10.1016/j.ijresmar.2018.12.003.